Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Language
Reasoning
Education

Problem Decomposition in Large Language Modeling Using Reasoning Experts

Master's Thesis

submitted by

Marcel Braasch

at the

Department of Computer Science

Advisors:	Jingwei Ni
	Kumar Shrihdar
Supervisor:	Prof. Elliot Ash
Examiner:	Prof. Georg Groh

October 3, 2024

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 01.09.2024

. . . . . .
Marcel Braasch

Abstract

Large language models have seen a steady rise in popularity in recent years showcasing impressive abilities in a wide range of tasks. Logical reasoning and planning are among those capabilities. Large models ($\geqslant$ 70B parameters) have proven to be good reasoners steadily beating strong human baselines. Smaller models ($<$ 70B parameters), on the other hand, lack behind. Real-life applications have hard constraints, which very large models violate. Models often need to be deployed on devices with limited resources. Additionally, users expect low latency in a model's response. Thus, there is an inherent desire to improve abilities of smaller models while meeting these requirements.

Current approaches for solving reasoning problems evolve around chain-of-thought techniques. Instead of generating only the final answer in and end-to-end manner, models are instructed to solve problems step-by-step. Each step generated thus builds upon the previous, improving performance drastically.

This work examines how to decompose the reasoning process generated by a language model's chain-of-thought process. We examine whether it is possible to split certain parts of the reasoning chain vertically and distribute it horizontally over specialized experts. This work introduces a new framework called Chain-of-Experts (CoE) that in its essence distributes the the reasoning processes horizontally over specialized models.

We evaluate a range of settings and techniques to improve model performance on reasoning tasks. This includes different CoE architectures, different data sets, including extensions and experiments with teacher forcing, sampling techniques as well as routing mechanisms. Furthermore, we conduct a series of ablation experiments analyzing a mixture-of-expert and setting and dependency graphs.

In this work we show that simple model decomposition likely is not sufficient to boost performance. More sophisticated architectures are required, which do show signs of effectivity, however, none of which showed *consistent* strong results.

Contents

Chapter 1

Introduction

Large language models have seen a steady rise in popularity in recent years showcasing impressive abilities in a wide range of tasks [35, 5, 10]. Logical reasoning and planning are among those capabilities. Large models ($\geqslant$ 70B parameters) have proven to be good reasoners steadily beating strong human baselines. Smaller models ($<$ 70B parameters), on the other hand, lack behind.

Real-life applications have hard constraints, which very large models violate. Models are often deployed on devices with limited resources. Additionally, users expect low latency. LLMs with a magnitude of billions of parameters are beginning to show reasoning capabilities of human-like ability [45]. Small-sized pre-trained language models, however, are lacking behind and known to be poor reasoners [41]. Thus, there is an inherent desire to boost SLM's reasoning abilities while meeting memory and latency requirements.

1.1 Motivation

Current approaches for solving reasoning problems evolve around chain-of-thought (CoT) techniques [45]. Instead of generating *only* the final answer, models are instructed to solve problems step-by-step. Each step builds upon the previous, improving performance drastically. This work examines how to decompose the reasoning process generated by a language model to increase its ability to solve difficult problems.

There is a whole body of work examining decomposition in machine learning and language modeling [43, 39, 21]. Shridhar et al.(2023), for example, show that, within a socratic framework it is beneficial to divide subquestion generation and reasoning step generation over two different models [39]. Further, Jain et al. (2023) show that especially the first step in a reasoning chain is crucial for solving a problem instance correctly [21]. If the model gets the first step wrong, it is almost impossible to recover from that mistake and most likely the model will fail to make a correct prediction. This works mainly builds on these observerations, trying fit the approach in a Mixture-of-Experts-like framework we call Chain-of-Experts. For a more in depth analysis on related work, please refer to Section 2.

1.2 Approach

In this work, we examine how to decompose the reasoning process generated by a language model's chain-of-thought process. We examine whether it is possible to split certain parts of the reasoning chain vertically and distribute it horizontally over specialized experts. This work introduces a new framework called Chain-of-Experts (CoE) that tests how certain parts of a reasoning processes can be solved by certain, specialized models.

At the moment, there are two main directions to improve a pre-trained language models' capabilities. Either, a model's abilities are improved through prompting techniques and in-context learning [40, 31, 13],

or fine-tuning [29, 55, 12]. In this work, we will make use of fine-tuning. To stay within memory limitations and be able to work with the hardware we have at hand, all models are implemented using LoRA [18]. Importantly to note, LoRA is a determining factor in this approach as we can quickly switch lightweight adapters to conduct certain parts of a chain without the need of having to train the full model.

We implement various strategies and configurations where each expert is responsible for a certain part of generating the reasoning chain and then passes on its result to the next expert. During inference, sub-results are chained and passed from model to model.

Further, we conduct a series of ablation studies analyzing the nature of reasoning prolems. Further, we conduct a series of ablation studies utilizing the Mixture-of-Experts (MoE) framework and dependency graphs, aiming to provide evidence how to train a CoE correctly.

1.3 Aim

The aim of this thesis is to study whether experts specialized on specific parts of the reasoning process, and chaining them, is beneficial. Throughout the entire thesis, we run experiments for various architectures and benchmark them against a strong baseline model. The overarching goal of this thesis is thus to find a method that improves the performance of small language models on reasoning tasks, without introducing too much complexity into the architecture. In the best case, the method developed will generalize to a range of reasoning tasks and models.

1.4 Overview

In Chapter 2 we give a brief overview of prior work, namely LoRA (Section 2.1), Mixture-of-Experts (Section 2.2), Reasoning in LLMs (Section 2.3), LLM Routing (Section 2.4), Problem Decomposition 2.5 and Teacher Forcing (Section 2.6. This Chapter shall serve as a headstart into the fundamentals of the techniques this work is based on.

In Chapter 3 we present the main contribution of this thesis. The problem statement is formulated out in Section 3.1.1 outlining the issue at hand and proposing ways to solve it. Then, the dataset and its variations are described in Section 3.1.2. In Section 3.1.3 the concept of Chain-of-Experts are defined and described in its architecture, training and inference procedure. A general definition of the CoE architecture is given, the baseline model is introduced in Section 3.1.5 and more sophisticated architectures are pictured in Sections 3.2 to 3.4. Lastly, in Section 3.5 our teacher forcing approach is shown. Lastly, we present Multiple-Choice Decoding in Section 3.6 and First Step Routing in Section 3.7. For each of the five architectures presented we show results dataset and conclude with a brief discussion, and if applicable, show the prompt.

In Chapter 4 we show two experimental settings trying to shed light on the reasoning process. In Section 4.1 we investigate a Mixture-of-Expert setting and in Section 4.2 we try to fundamentally understand better how reasoning problems are strucutred.

Finally, in Chapter 5 one can find a brief overview of the thesis, possible problems and limitations and next steps.

Chapter 2

Fundamentals

In this section an overview of current methods is presented. It shall serve as a headstart for the fundamentals of what this work builds on.

2.1 LoRA

LoRA, short for low-rank adaption, is a parameter efficient fine-tuning technique that allows for resource efficient fine-tuning of large language models [18]. Instead of conducting direct parameter updates of a model's dense layers, two rank decomposed matrices are updated, while keeping the actual layer's parameters frozen. During inference, the decomposed trained matrices are merged into their layer. This entails performing inference on fine-tuned base model with no additional overhead. Additionally, it enables fast task switching as simply subtracting of previous and addition of the newly desired weights is required.

2.2 Mixture-of-Experts

The Mixture-of-Experts (MoE) framework has been around for almost three decades [2, 1]. In essence, the framework aims to partition the problem space into subspaces and distribute it over specialized experts, supervised by a gating mechanism. Each specialized experts is thus only focus on their partition.

One can differentiate between mixtures of implicitly localized experts (MILE) and mixtures of explicitly localized experts (MELE) [30]. Many of current works can be attributed to MILE, where the problem space is implicitly partitioned by a dynamic gating mechanism of a singular large neural network [22, 49, 27, 37, 15]. In the context of LLMs, this often resonates with replacing the FFN in a transformer with a MoE layer. Other works focus on MELE, where the problem space is examined a priori and explicitly partitioned and distributed across experts. [39, 60, 20]. The choice of the gating mechanism is arbitrary and can range from binary, sparse, continuous, stochastic to deterministic functions [37].

Formally, in vector notation, a MoE can be expressed as

$$y = \mathbf{G}(\mathbf{x})\mathbf{E}(\mathbf{x}) \tag{2.1}$$

where $\mathbf{G}(\mathbf{x}) \in \mathbb{R}^{1\times n}$ is the output of a gating network and $\mathbf{E}(\mathbf{x}) \in \mathbb{R}^{n\times 1}$ is the output of the experts given input x. Since $\mathbf{G}(\mathbf{x})$ is typically a sparse vector, it follows that if entry i is 0, expert i does not need to be computed.

2.3 Reasoning in LLMs

Exploration and description of human reasoning has been around for ever since mankind thinks and reflects. Aristotle *already* described processes underlying human thought processes [3]. Since this work explores reasoning in the domain of LLMs this section will largely cover selected works that provide a kick-start into the domain.

Yu et al. (2023) provide an in-depth survey ocer current methods in natural language reasoning. They define what reasoning in the context of NLP is, why to use LLMs for reasoning, which methods are currently available and benchmarks and variations of reasoning tasks [52].

Besta et al. (2004) investigate variations of chain-of-thought prompting and generation topologies. They provide a broad analysis of reasoning topologies, how to construct and use them for inference and cover all concepts from chain-of-thought (CoT), tree-of-thought (ToT) to graph-of-thought (GoT) [4].

Yao et al. (2023) claim that human thought processes are often non-linear and a reasoning scheme supported by a graph of thoughts (not to be confused with a GoT sampling scheme) and predicted rationales may be a more suitable framework [51].

Faldu et al. (2021) take a similar approach and give a vast overview of both neural and non-neural approaches to solving math word problems (MWP). Especially note-worthy is that previous approaches have broadly relied on explicit structures to aid a systems, either defined by hand or generated automatically [16]

Wong et al. (2023) propose means to translate unstructured LLM output into structured and logic-based hierarchies that aim to give the reasoning- and solution-finding-process more robustness [48].

Huang et al. (2023) argue that currently to improve language model's reasoning capabilities, it is not very util to have a model critique and correct its own output, rather it hurts model performance [19]

Du et al. (2023), however, find that employing an agent-system that debates among themselves in order to find the best solution can be beneficial or LLM reasoning tasks [14].

Fu et al. (2023) show that specializing smaller language models for multi-step reasoning is a beneficial endeavour. Their work focuses on conduct a series of exeriments evaluating the trade-off between models' capabilities to perform general task versus tailoring the to very specific problems [17].

Lastly, there is a body of work exploring how CoT can aid reasoning in multi-modal models. Tasks like image segmentation or question-answering on a corpus of images have been shown to see benefits from CoT reasoning schemes [25, 57, 58].

2.4 LLM Routing

There is a large body of literature around routing LLMs in a MoE-fashion. The main idea is that difficulty of certain samples is judged and routed to a specific model to solve the sample accordingly. For almsot all approaches the main assumption is that for some samples it is sufficent to be solved by a weaker model, yielding a result of similar quality, at much lesser cost.

Ong et al. (2024) propose RouteLLM, which is a router trained on preference data from Chat Bot Arena battles enhanced by data augmentation [33].

Madaan et al. (2023) propose AutoMix, which uses a Partially Observable Markov Decision Process to route queries between different language models [28].

Mohammadshahi et al. (2024) propose Routoo, an architecture based on a a performance predictor and a cost-aware decoder, designed to optimize between variables such as performance, cost, and efficiency [32].

Shnitzer et al. (2023) propose a routing mechanism for large language models (LLMs) that leverages benchmark datasets to learn a binary classifier for predicting the best-performing model for a new task, effectively reducing the problem to a series of binary classification tasks [38].

Chen et al. (2023) propose FrugalGPT, a framework that reduces the cost of using large language models (LLMs) by employing strategies like prompt adaptation, LLM approximation, and LLM cascades to maintain performance while minimizing expenses [8].

Ding et al. (2024) propose HybridLLM, a routing framework that uses a BERT-style encoder to classify queries based on their difficulty and route them to either a small or large model, with the goal of reducing calls to the large model [11].

2.5 Problem Decomposition

Especially in question answering, decomposition has been studied extensively [23, 59]. Research differentiates between unsupervised, semi- and fully supervised decomposition methods [34]. For now, this is not as interesting for our case as the decomposition is already given in the dataset with its CoT decomposition, thus we will treat our following case as supervised problem decomposition.

Generally speaking, CoT and all related concepts can itself be regarded as problem decomposition already [45]. Instead of mapping the solution to a problem in a direct end-to-end manner intermediate steps are explicitly formulated out, decomplexifying and breaking the problem apart.

A popular method to understand decompisition better is the socratic questioning theme [39, 54, 7]. In addition to conducting a series of reasoning steps, for each step a Socratic question is formulated guiding the model toward generating the next step and producing an improved final solution.

2.6 Teacher Forcing

While training machine learning models, lots of research effort has gone into investigating distorting gold-label training data by noise [40, 24]. In sequential modeling, especially recurrent neural networks, teacher forcing (TF) has been a common practice to attain noisy-labels helping models to generalize better and become more robust [47, 26, 36].

2.7 Software and Parameters

To train the models the Huggingface API was used. All scripts used are provided here. To fine-tune models we rely on Huggingface's LoRA implementation. All models used and tested on are instruction-tuned language models ranging from 3 to 7 billion parameters. We opt for instruction-tuned models as these typically perform better across most tasks and usually are the ones used in real-world applications. We do not use quantized models as their models fit on a single RTX 3090 and 4090, which are the GPUs utilized throuth this thesis.

For model training the following parameters were used. Batch size 8, learning rate (LR) $2e-4$, warmup steps 5, trained with Bfloat16, weight decay 0.01, LR scheduler type "constant", training for 1 epoch. Even though a linear LR is more commonly used, a constant LR rate is crucial for comparability. Sample sizes differ between models. In the case of a linear LR scheduler the LR will adopt relative to the sample amount. For LoRA, the following parameters were used. Rank 16, Alpha 32, as target modules all linear layers except the LM head are trained, dropout and bias 0.

Surely, there are better parameters to increase performance but finding a better model was not the goal of this thesis. Importantly to note, models are only trained on gradients of the target sequence only. The frontal part, i.e., the question and previous steps when decomposed, are ignored.

Chapter 3

Chain-of-Experts

3.1 Introduction

3.1.1 Problem Statement

Current approaches for solving reasoning problems evolve around chain-of-thought techniques [57, 56, 44]. Instead of generating only the final answer, models are instructed to solve problems step-by-step, explicitly formulating out their ”thoughts”, where each step builds upon the previous. Typically, one model generates the entire reasoning chain and arrives at a solution.

This work examines various directions of enhancing reasoning abilities of SLMs. It makes use of specialized experts that each solve a subspace of the entire problem space. We explore a range of possibilities to divide the problem space of reasoning chains and distribute it over specialized experts.

Prior evidence suggests that especially the first step of the reasoning chain is crucial to arrive at a correct answer [21]. If the first step is wrong, the model is almost certain to not solve the problem correctly. We hypothesize that splitting the steps and distributing it over specialized experts may yield gains over having one general model solve the entire problem.

3.1.2 Data

The approach described will be developed along and evaluated on the GSM8K dataset [9], which serves as a common ground for understanding LLM reasoning within the NLP community [14, 17, 4].

Gold Label GSM8K / GSM8K-GT is the ground truth GSM8K dataset comprised of 8.5k high-quality and linguistically diverse grade school math word problems requiring basic arithmetic operations to reach a final solution. It has 7473 training examples and 1319 test examples. There is no dev set provided. To solve a problem 2 to 8 steps are needed. Most problems, a middle schooler could confidently solve with ease.

Distilled GSM8K / GSM8K-DL is a distilled version of GSM8K obtained from Llama 3 8B. The model was asked to generate answers for the training dataset. Samples which the model predicted correctly where used as part of the new dataset. Augmenting or enriching a dataset with LLMs annotation has become a common practice lately and is known to boost performance significantly [17].

Socratic GSM8K / GSM8K-SC is the Socratic version of GSM8K. For each reasoning step conducted, a Socratic question is asked before the step is conducted. This expands the idea of thinking step by step by ”thinking” even more granularly. In Figure 3.1 one can observe an example question in its normal and socratic form.

MetaMath GSM8K / GSM8K-MM is an augmented version of GSM8K taken from MetaMath [53]. MetaMath paraphrases questions in different styles to improve performance across mathematical tasks. Importantly to note, we only test on the GSM8K-augmented samples in the MetaMath dataset for computational reasons resulting in 240K training samples.

Question:
There are 25 roses in a garden. There are 40 tulips. There are 35 daisies. What percentage of flowers are not roses?

Answer:
How many flowers are there? ** There are 25+40+35=<<25+40+35=100>>100 flowers total.
How many flowers are not roses? ** There are 40+35=<<40+35=75>>75 flowers that are not roses.
What percentage of flowers are not roses? ** Therefore, (75/100)*100=<<(75/100)*100=75>>75% of the flowers are not roses.

Figure 3.1: Example from the GSM8K dataset. The question is at the top and the answers to be generated by the LLM at the bottom. In red, one can see the reasoning steps given in every GSM8K dataset. The questions indicated in blue are the Socratic questions which occur in the GSM8K-SC dataset. In case of the non-Socratic dataset the blue part would simply not be available.

Data formatting

For each of the strategies explained in Section 3 data needs to be formatted a certain way, depending on the model at hand. We use a certain template that supports handling complex configurations. Each sample is comprised of a dictionary with four entries, that is, `source_question`, `source_steps`, `target_steps` and `target_result`. Depending on what each model is supposed to learn, datapoints are formatted accordingly using this data template, being incorporated into the prompt shown below.

This datastructure serves as important constant during the entire experimantation as we only need to format the dataset accordingly and can easily train the simpler models shown in Sections 3.3 and 3.4.

```
{
   "source_question": "Kelly booked a three week vacation to visit relatives. The first […]",
   "source_steps": "Three weeks is 7*3=<<7*3=21>>21 days.\nShe traveled one day to her
   grandparents' house, one day to her brother's house, two days to her sister's house, and two
   days back home for a total of 1+1+2+2=<<1+1+2+2=6>>6 days.\nShe spent 5 days at her
   grandparents' house and 5 days at her brother's house, for a total of 5+5=<<5+5=10>>10 days.",
   "target_steps": "Thus, of the 21 vacation days, the number of days remaining to spend with her
   sister was 21-6-10=<<21-6-10=5>>5 days.",
   "target_result": "5"
}
```

Figure 3.2: A training example how model M_2 in the previous example could be trained. As described in Section 2.7, the model is exclusively trained on the `target_steps` and `target_result`. The `source_question` and the `source_steps` are getting formatted into the prompt accordingly.

3.1.3 Definition

A Chain-of-Experts (CoE) model $\mathcal{C}$ can be expressed as

$$\begin{aligned} \mathcal{C}_n(x) &= M_n(M_{n-1}(\dots(M_1(x)))) \\ \Leftrightarrow \qquad &= M_1 \circ M_2 \circ \dots \circ M_n \end{aligned} \tag{3.1}$$

where x is the input and models M_1 to M_n are the expert models. The input x will first be passed into model M_1 which passes its output as input to model M_2. Which expert is responsible for which subspace, or how many steps, is defined separately for each of the strategies.

Assume a problem where maximally $n = 8$ reasoning steps $\{s_1, \ldots, s_n\}$ are required to reach a final solution. The question now becomes how to assign a number of maximally n experts where each solves a certain subspace of those n steps.

From a combinatorial perspective, there is a large amount of combinations one could try. If we allow an expert to only solve adjacent steps (so e.g., it could solve $\{s_1, s_2, s_3\}$ but not $\{s_1, s_3\}$) we have exponential complexity with $2^n = 256$ different combinations. If we loosen that criterion complexity becomes the Bell number with $\mathcal{B}_8 = 4140$ different partitions.

3.1.4 Prompt

If not indicated otherwise, across the work the Alpaca fine-tuning prompt was used [42]. In Figure 3.10 one can see what it looks like.

```
Below is an instruction that describes a task, paired with an input that provides
further context. Write a response that appropriately completes the request.

### Instruction:
Solve the following math word problem step-by-step.

### Input:
{source_question}

### Response:
{source_steps}
{target_steps}
Final answer: {target_result}{eos_token}
```

Figure 3.3: The Alpaca prompt used in this work. The instruction is already filled according to solving GSM8K. Also note that, if available, `source_steps` will be formatted into the prompt during training. `target_steps` are always available, thus will be formatted in the prompt as well. If the sample at hand has a `target_result`, it will be included, too. The `eos_token` will always be part of the prompt, indicating termination.

3.1.5 Baseline

The singular model will serve as the baseline throughout the entire thesis. Strictly speaking, the baseline is a CoE such that $\mathcal{C}(x) = M_1(x)$ where $\mathcal{C}$ only consists of one expert and this expert is responsible for generating all reasoning steps. In Figure 3.4 the procedure can be observed visually.

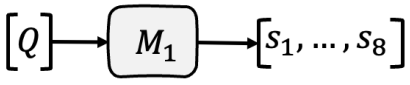

Figure 3.4: The baseline model which consists of only a singular model, receiving the question as input and generating the entire sequence.

3.2 Full Chain

Architecture For the Full Chain (FC) architecture we define exactly as many experts as the maximum of reasoning steps needed. In other words, one expert is responsible for exactly one reasoning step. In Figure 3.5 one can observe this process visually.

Inference During inference M_1 receives the question Q and generates step s_1. M_2 receives the question Q, s_1 previously produced by model M_1 and generates s_2. This continues until one expert decides it has reached a solution such that the chain does not need to be passed on.

Training The training procedure follows intuitively. Assume a training dataset $\mathcal{D} = \{\{q_i, x_i\}\}_{i=1}^{n}$, where q_i is the question i, and $x_i = \{s_{i1}, \ldots, s_{ik_i}\}$ are the steps for question i at step j, k_i is the maximum amount of reasoning steps for sample i and n is the number of examples. For a FC with an amount of models as there are maximum steps, for model M_m (i.e., the m-th model in the chain), the input data becomes $q_i, x_i \in \mathcal{D}$ where $s_m \in x \wedge \forall s_k \in x : k \leqslant m$. The target then becomes step s_m. Said simply, model m is exclusively trained to generate step m. If at step j a sample is concluded, the conclusion will be appended to the sample. This allows all models in the chain to make an autonomous decision whether it has reached the final answer or not.

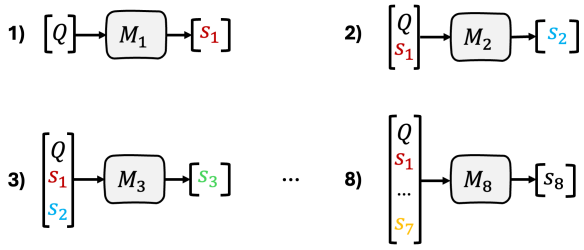

Figure 3.5: A full model pass of the FC strategy. The question Q is passed into model M_1 and generates step s_1 (red). Model M_2 receives the question and the generated previous step and produces step s_2 (blue). This continues until one of the models in the chain decides to conclude.

3.3 Two-Expert Chain

3.3.1 Architecture

Introducing exactly one model for one step in the reasoning introduces a lot of complexity into the model. In the case of GSM8K the dataset is comprised of problems where 2 to 8 reasoning steps are needed to solve a problem. This means that one has to train and do inference on eight experts. To decomplexify this, we implement a simpler architecture in a similar fashion. We reduce the number of experts from N to 2 by defining a cut c, where expert M_1 solves the problem until step s_c, model M_2 continues at $c + 1$ and concludes until the end. In Figure 3.6, similarly to the previous one, the procedure can be observed visually.

Inference During inference, M_1 receives the question Q and generates step s_1 until step s_c. Then, M_2 receives the question Q, steps s_1 to s_c previously generated by model M_1 and generates the rest of the solution.

Training The training procedure follows as for the FC strategy. We assume a dataset $\mathcal{D}$ as described above. For model M_1, the input data is only the question q and the model is trained to generate $\{s_1, \ldots, s_c\}$ where c denotes the cut-off. Similarly, for model M_2, the input will be the question q alongside steps $\{s_1, \ldots, s_c\}$. The model is then trained to generate steps $\{s_{c+1}, \ldots, s_n\}$. If for a sample i the solution is already reachable in $i \leqslant c$ steps, model M_1 concludes the sample completely. This allows for any of the models in the chain to make an autonomous decision whether it has reached the final answer or not.

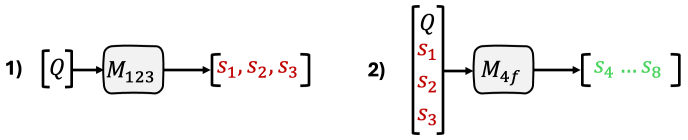

Figure 3.6: A full model pass of the $\mathbf{TEC_n}$ strategy for $n = 3$. The question Q is passed into model M_{123} and generates steps s_1, s_2 and s_3 (red). Model M_{4f} (f = final) receives the question and the generated previous steps and produces step s_4 until the end (green).

3.3.2 Results

In Table 3.1 one can see the results of the TEC approach. Experiments are conducted with three cuts, at 1, 2 and 3. Further we evaluate on 4 datasets (see Section 3.1.2).

	Phi-3-Mini	Gemma 2B	Mistral 7B	DS 7B
	GSM8K-GT			
Baseline	**0.7102**	**0.3303**	**0.4017**	**0.4899**
TEC_1	0.6984	0.3033	0.3233	0.4735
TEC_2	0.7005	0.2939	0.3243	0.4746
TEC_3	0.6993	0.3019	0.3492	0.4842
	GSM8K-DL			
Baseline	0.7249	**0.2610**	**0.4854**	**0.5713**
TEC_1	0.7107	0.2387	0.3888	0.5492
TEC_2	0.7293	0.2352	0.4038	0.5407
TEC_3	**0.7310**	0.2531	0.4180	0.5534
	GSM8K-SC			
Baseline	0.6896	**0.2669**	**0.3820**	**0.4923**
TEC_1	0.6805	0.2411	0.3116	0.4508
TEC_2	**0.6923**	0.2359	0.3351	0.4835
TEC_3	0.6834	0.2462	0.3713	0.4761
	GSM8K-MM			
Baseline	-	-	**61.3091**	-
TEC_1	-	-	55.8251	-
TEC_2	-	-	55.8251	-
TEC_3	-	-	55.8251	-

Table 3.1: Comparison of TEC-models in three different configurations compared to a full baseline model.

The results shown in 3.1 indicate that the TEC strategy underperforms compared to the baseline. In most cases, the baseline model outperforms all TEC configurations across all datasets. The TEC approach

yields only marginal improvements in specific configurations, such as a 1% gain with the Phi-3-Mini model in TEC, but overall it fails to perform effectively across model sizes and datasets. Notably, the GSM8K-MM results show that TEC underperforms significantly.

There are several reasons why the approach might not be a suitable solution for improving the reasoning of SLMs.

1. **Locality of Experts** Each expert only handles a subset of reasoning steps, which limits their view of the overall problem. This segmentation of reasoning can lead to errors when intermediate steps are inaccurate, as later experts depend on incomplete or incorrect context.

2. **Increased Complexity** Dividing the reasoning chain into multiple models introduces more complexity and overhead in both training and inference without substantial performance gains. Importantly to note, we used opted for a model with two experts because of exactly this overhead. Still, with two experts, complexity in the design of the architecture increased significantly.

An interesting direction to take here could be dynamic expert assignment. Rather than using fixed cuts, one could implement a more adaptive model where experts dynamically adjust to handle varying levels of reasoning complexity.

3.4 Overlap-Split Chain

3.4.1 Architecture

Training local experts may bring some issues with it. While each model is focused on their respective part of the reasoning chain, by design, it loses information about the surrounding steps. To overcome this, we test another strategy that tries to find a trade-off between globality and locality. The idea is to train models on all reasoning steps until the end but only use the frontal part of the generation during inference, the rest will be cut off.

In $\mathbf{OSC_{s,g,m}}$ the parameter s determines how many steps are wished to be passed to the next expert *after clipping*, g denotes the amount of steps it generates, and m denotes how many models there are chained in this strategy. Also note that $s < g$.

The parametrization itself controls the trade-off between locality and globality. Technically speaking, all previously introduced configuration are special cases of $\mathbf{OSC_{s,g,m}}$. The baseline model, i.e., $\mathbf{OSC_{1,1,1}}$, is comprised of one model, generating one step, passing on one step to the next model. The FC model, i.e., $\mathbf{OSC_{1,1,8}}$, passes $s = 1$ step to the next model, is trained on generating $g = 1$ steps and comprised of $m = 8$ models. Further, it holds that $\mathbf{OSC_{s,g,2}} = \mathbf{TEC_n} \iff s = g = n$.

Inference During inference, M_1 receives the question Q and generates step s_1 to and s_n. Then, only the first m steps generated by the model are kept and passed to the next model. M_2 then generates steps s_2 to s_n and the procedure repeats until one of the models in the chain concludes.

Training Models M_i are trained on the entire sequence until step n. The main difference compared to the other approach is how inference is conducted, leaving barely any modifications to the training data. For the $\mathbf{OS_{1,3,4}}$-model the procedure can be observed visually in Figure 3.7.

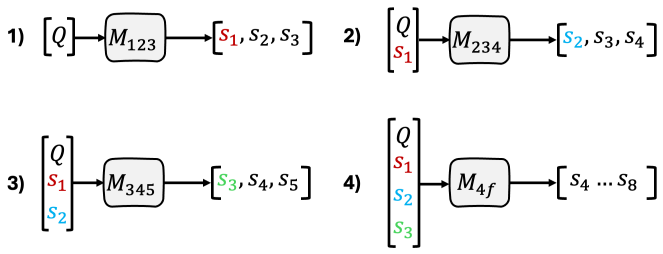

Figure 3.7: A full model pass of a $\mathbf{OS_{1,3,4}}$-model. The question Q is passed into model M_{123} and generates steps s_1, s_2 and s_3 (red, $n = 3$). Model M_{234} receives only the question and step s_1 ($m = 1$) and generates steps s_2, s_3, s_4. This happens $m - 1$ times until one model in the chain concludes.

3.4.2 Results

The OSCs are trained with three configurations. All models implement $m = 2$ experts, and are asked to generate until the end $m = 8$. The cut n is tested at 1, 2 and 3. In Table 3.1 one can observe the results tested on three datasets.

	Phi-3-Mini	Gemma 2B it	Mistral 7B it	DS 7B Chat
	GSM8K-GT			
Baseline	**0.7102**	**0.3303**	**0.4017**	**0.4899**
$OSC_{1,8,2}$	0.6993	0.2582	0.3630	0.4795
$OSC_{2,8,2}$	0.6957	0.2349	0.3210	0.4714
$OSC_{3,8,2}$	0.6834	0.2391	0.3428	0.4696
	GSM8K-DL			
Baseline	0.7249	0.2610	**0.4854**	**0.5713**
$OSC_{1,8,2}$	**0.7307**	**0.3154**	0.4309	0.5588
$OSC_{1,8,2}$	0.7254	0.2839	0.4165	0.5466
$OSC_{3,8,2}$	0.7289	0.2901	0.4331	0.5574
	GSM8K-SC			
Baseline	0.6896	**0.2669**	**0.3820**	**0.4923**
$OSC_{1,8,2}$	0.6849	0.2597	0.3541	0.4693
$OSC_{2,8,2}$	**0.6905**	0.2024	0.3124	0.4714
$OSC_{3,8,2}$	0.6773	0.2293	0.3450	0.4728

Table 3.2: Comparison of OSC-models in three different configurations compared to a full baseline model. Only Phi-3-Mini accuracy improves by about 1% absolute gain. For the other models the method was not effective.

The results in Table 3.2 indicate that the OSC strategy does not outperform the baseline in most cases. Despite its architectural flexibility, it fails to consistently improve reasoning in smaller LLMs across all datasets. There are, however, a few notable points worth discussing.

1. **Marginal Improvements in Certain Configurations** In the GSM8K-DL dataset, the $OSC_{1,8,2}$ configuration shows a slight improvement for Phi-3-Mini over the baseline, achieving 73.07% accuracy compared to 72.49%. However, these gains are marginal and do not generalize across models or datasets. On the GSM8K-GT dataset, OSC performs worse than the baseline for all tested models and configurations, highlighting the approach's limited applicability.

2. **Global-Local Trade-off** The OSC framework introduces a balance between locality (experts handling a small subset of steps) and globality (retaining awareness of broader steps). However, in practice, this trade-off doesn't seem to yield substantial benefits, as dividing reasoning across models with overlap doesn't enhance performance as expected. For example, the $OSC_{1,8,2}$ strategy performs better than the other OSC configurations, suggesting that having smaller overlap regions may help, but even this approach struggles compared to the singular baseline.

Analogous to the suggestion for the TEC, instead of using a fixed overlap between experts, a dynamic mechanism could adjust the size of the overlap based on the complexity of the problem. For example, simpler problems might need less overlap, while more complex problems could benefit from larger overlaps between experts. Adaptive Expert Assignment:

3.5 Teacher forcing

3.5.1 Architecture

Assume a sequential CoT model $\mathcal{C}$, defined as in Eq. (3.1). Each model $M_i \in \mathcal{C}$ is trained on an input-output pair (x_{i-1_g}, x_{ig}). In a smooth and de-noised setting both the input x_{i-1_g} and output x_{ig} signal stem from the ground truth data, hence the g indicating ground truth labels. Teacher forcing (TF), in essence, aims to replace the ground truth input signal with the previous unit's output during training. I.e., we exchange the non-noisy ground truth input x_{i-1_g} with the (possibly) noisy output produced by M_{i-1}, i.e., $(x_{i-1_{M_{i-1}}})$. In Figure 3.8 one can see this procedure visualized.

To implement this strategy there are a few design choices to make. A priori, it is not clear whether simply replace the input with its noisy counterpart or wheter one should just copy, modify, add it to the dataset. Further, it is not clear where to set the replacement/adding percentage, therefore we test in 5% steps, from 0% to 30%.

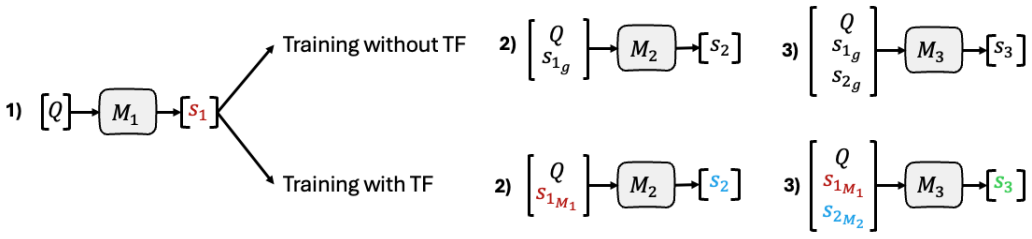

Figure 3.8: Example for TF implemented on the FC strategy. Model M_1 produces step s_1 indicated in red. In normal training (top), the model M_2 is trained on the ground truth step 1 s_{1_g}. This continues such that all models M_m are trained independently. In a teacher forcing setting (bottom), model M_2 receives as an input step 1 generated by M_1, so $s_{1_{M_1}}$ instead of s_{1_g}. The target s_2, however, for both cases remains the ground truth.

3.5.2 Results

The approach is evaluated on **GSM8K-GT** and **GSM8K-DL**. The model used here are deepseek-llm-7b-base (7B parameters). Results are not averaged and shown for one run only as only later during the work the this robustness factor was regarded.

The approach is tested against two models, that is, TEC_3 as evaluated in the previous sections and $OSC_{1,3,8}$. The OSC model evaluated here can be regarded as a mixture of the FC model described in Section 3.2 and the splitting strategy described in Section 3.4.

We employ two different TF approaches that is Added (A) and Replaced (R). The Added strategy, as the name suggests, adds x% of samples to the training dataset. The Replaced strategy replaces the samples. Importantly to note, for high percentages, like 25% and 30%, data for later models becomes scarce as the the distribution is heavily centered around 2- and 3-step examples. Mix in percentage is then clipped to the maximum percentage still available at this point.

As can be seen in Table 3.3 the teacher forcing approach does not seem suitable to improve reasoning in smaller LLMs, at least for this particular set of models.

Data	Model	TF	BL	0	5	10	15	20	25	30
GSM8K-GT	TEC_3	A	0.2796	**0.3139**	0.2805	0.2866	0.2775	0.2896	0.2805	0.2873
		R	0.2796	**0.3139**	0.2881	0.2858	0.2835	0.2835	0.2835	0.2828
	$OSC_{1,3,8}$	A	0.2796	**0.2881**	0.2782	0.2874	0.2851	0.2853	0.2926	0.2798
		R	0.2796	**0.2881**	0.2805	0.2828	0.2828	0.2858	0.2820	0.2805
GSM8K-DL	TEC_3	A	0.3510	**0.3798**	0.3730	0.3601	0.3647	0.3548	0.3533	0.3609
		R	0.3510	**0.3798**	0.3639	0.3639	0.3617	0.3533	0.3634	0.3578
	$OSC_{1,3,8}$	A	0.3510	**0.3586**	0.3594	0.3586	0.3472	0.3617	0.3548	0.3541
		R	0.3510	**0.3586**	0.3525	0.3497	0.3497	0.3525	0.3510	0.3374

Table 3.3: Teacher forcing with mix-in of x%. TF stands for the teacher forcing strategy chosen, where A and R indicate the added and replaced strategy respectively.

The results from the Teacher Forcing (TF) experiments, as shown in Table 3.3, suggest that the approach provides marginal or no improvement over the baseline models. While the initial idea was that introducing progressively more noisy input (i.e., replacing the ground truth with generated steps) could help the model learn to generalize better under realistic conditions, the data does not support this hypothesis for the current setup and models. Several factors could explain these findings:

1. **Marginal Improvements** While certain configurations showed slight improvements—such as a 3% accuracy gain for TEC on GSM8K-GT at the 5% mix-in rate—these improvements were inconsistent and did not generalize across different datasets or TF strategies. For example, at higher percentages of TF, performance either stagnated or dropped, suggesting that the models were unable to handle increasing levels of noisy input effectively.

2. **Limited Gains from Replacing vs. Adding** Both the "Added" and "Replaced" strategies produced only marginal differences in performance, with neither method providing a clear advantage. In some cases, adding noisy samples to the dataset resulted in minor improvements (as seen in the 5% mix-in for TEC), but replacing ground truth samples with noisy ones did not lead to any significant performance boost, and often led to worse results.

3. **Early Stopping or Data Scarcity** Particularly for higher mix-in rates (e.g., 25% and 30%), data scarcity became an issue, as the distribution of reasoning steps heavily skewed towards simpler examples with fewer steps. This could have introduced bias, reducing the models' ability to generalize across more complex problems with multiple reasoning steps.

3.6 Multiple-Choice Decoding

Naively splitting a reasoning chain and just distributing it over a CoE detoriorates performance, or at least does not improve it. Another strategy utilized in this work is Multiple-Choice Decoding (MCD). In this setting, multiple viable reasoning steps are generated seperately and the best solution is chosen to proceed on. Simply speaking, we do not just sample the entire reasoning chain and choose the best, but sample certain steps to proceed on.

In the previous setting, one model generates up to n steps of a reasoning process and then *naively* passes it on the the next. Between models, there is little room for variation, and hence little room for improvement. Additionally, it may be that experts are too focused on local signals and lose sight of the overall, global picture of the problem.

3.6.1 Architecture

In the MCD setting, we generate a set of viable intermediate steps and have the subsequent model judge which is the best. Generally speaking, the framework is built up the following way. We train a reasoner model $\mathcal{R}_i$ to conduct step i, and sample n times from it. The reasoner $\mathcal{R}$ acts like an up-projection or encoder from $1 \rightarrow n$. Further, we train a judge model $\mathcal{J}$ to judge which is the best out of the n reasoning steps given by the previous reasoner. The judge acts like a decoder down-projecting from $n \rightarrow 1$.

Importantly to note, the judge $\mathcal{J}$ decoding the previous step can be regarded like a multiple-choice task. This is the main advantage that shall motivate this method. Whether a reasoning step is correct or not is often decided by nuances in wording. In Figure 3.9 one can see such small differences in wording that have a large impact on the semantics and overall correctness of solving the problem.

Hypothesize We hypothesize that for SLMs it is hard to make out these differences on the fly while generating. Choosing from a set of given solutions is typically much easier to solve than working on a problem from scratch [6, 46]. If one model is solely focused on finding all viable steps, then the other only decodes, finds the best one, and proceeds with its part of the process. Futhermore, we hypothesize that this method is signficantly more sample efficient. Instead of sampling the entire chain (avg. 140 tokens) one only has to sample the first step (avg. 35 tokens) which accounts for 4 times less tokens to sample.

For a fair comparison of the method this means that if we train the models to sample the first step 16 times, a strong baseline would be a model that samples the entire chain $(140/35 =)$ 4 times and decides which is the correct solution by simple majority voting.

Question: Gene is sewing a quilt out of old souvenir t-shirts. He has one shirt from each vacation he has been on. Every shirt is its own quilt block. Each row is made of blocks from a different year of vacations. He goes on four vacations a year and has been vacationing since he was 23 years old. He is now 34. How many quilt blocks does he have in total?

❌ **Possibility 1 for s_1:** Gene has been on 34 - 23 = <<34-23=11>> 11 vacations.
✅ **Possibility 2 for s_1:** Gene has been vacationing for 34 - 23 = <<34-23=11>> 11 years.

Figure 3.9: Often, only nuances in wording decide whether a step is correct or not. While they are nuances in wording, on a semantic level, they can make the difference between solving or not solving a problem. Choosing which is the correct one among a set of suggestions eases the task for the model.

Definition Formally, in an MCD setting, an expert E_i can be expressed as

$$E_i = \begin{cases} \mathcal{R}_1, & \text{if } i = 1. \\ \mathcal{J}_{n-1}, & \text{if } i = n. \\ \mathcal{J}_{i-1} \circ \mathcal{R}_i, & \text{otherwise.} \end{cases} \tag{3.2}$$

where the full chain resambles $\mathcal{C}$ as in Eq. 3.1

$$\begin{aligned} & \mathcal{C}_n(x) = M_1 \circ M_2 \circ \ldots \circ M_n \\ \Leftrightarrow \quad & = E_1 \circ E_2 \circ \ldots \circ E_n \\ \Leftrightarrow \quad & = \mathcal{R}_1 \circ \mathcal{J}_1 \circ \mathcal{R}_2 \circ \ldots \circ \mathcal{J}_{n-1} \end{aligned} \tag{3.3}$$

with $n >= 2$ being the amount of reasoner-judge-pairs to utilize.

$\mathbf{TEC_1}$ with MCD If we focus on a model with two experts split after step 1, we follow the TEC_1 structure (see Section 3.3). Our model then becomes $C_2 = E_1 \circ E_2 = \mathcal{R}_1 \circ \mathcal{J}_1$ with the addition that judge $\mathcal{J}_1$ not only decodes and produces step 2 but generates the entire solution. Whether $\mathcal{J}$ should be one or two models is not clear and can be regarded as design parameter.

3.6.2 Prompt

We follow the Alpaca fine-tuning prompt as shown in Section 3.1.4. However, in this particular setting we want to instruct the model to make a selection from the given first steps. To achieve this behavior, we instruct the model to "copy, create from or get inspired from the given first step suggestions" or if "none are good create a new one".

```
Below is an instruction that describes a task, paired with an input that provides
further context. Write a response that appropriately completes the request.

### Instruction:
Solve the following math word problem step-by-step. Copy, create from or get inspired
from the given first step suggestions given in the "### Input"-part. If none are good
create a new first step. Pay close attention to the semantic nuances in differences
between the suggestions and be specific in your wording.

### Input:
Question: {source_question}

### Response:
<step 1>: {step 1}
<step 2>: {step 2}
...
<step n>: {step n}
<final answer>: {target_result}{eos_token}
```

Figure 3.10: The prompt used for the MCD approach. It follows the same prompt as shown in Section 3.1.4, however, adds flexibility to the model to make autonomous decisions. Furthermore, special `<step>` and `<final answer>` tokens are added to increase conistency in answers.

3.6.3 Temperature Experiments

Moving on to sampling the first step we need to understand better which temperature works best. Thus, we calculate a series of statistics. For this particular experiment Mistral-7B Instruct v0.3 is conducted.

Accuracy vs. temperature In Figure 3.11 on can see first step (F) and end result (E) accuracy with oracle and majority voting sampled for temperatures between 0.1 and 1.0 at steps of 0.1. For the majority voted first step (light green), unsurprisingly, accuracy slightly increases with increasing temperature and takes a dip at a temperature of 1.0. The dip likely comes from results being too creative (or not coherent) and

not delivering correct reasoning paths anymore. For the oracle first step (blue), unsurprisingly, first step accuracy continuously improves. The majority voted end result (red) increases until a temperature of 0.5, then almost drops back to where it was at a temperature of 0. The oracle end result consistently improves until it does not anymore at a temperature of around 0.8.

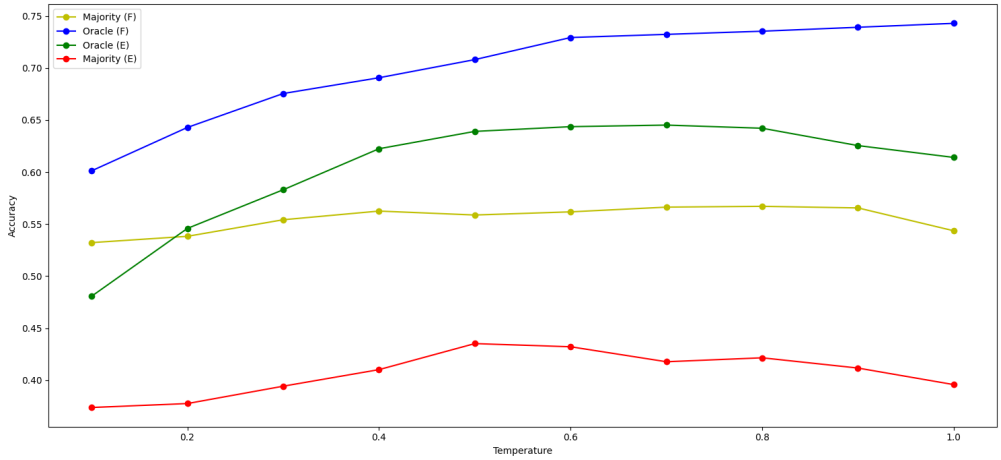

Figure 3.11: Majority and oracle voted first (F) and end result (E) accuracies for varying temperatures. Accuracies keep improving until a temperature of 0.5 and 0.6 and drop after a temperature of around 0.8 and 0.9.

The results suggest that temperatures of 0.9 and higher are not suitable as good sampling temperatures. Across three out of four metrics at 0.8, latest at 0.9 the performance takes a dip. Further, there is a rapid increase in improvement until a temperature of 0.4 and 0.5. Afterwards, performance either stagnates or even decreases. This suggests that a temperature of around 0.5 seems suitable to obtain samples that yield promising reasoning paths.

Diversity vs. Temperature In addition to that we wanted to investigate the diversity of the sampling process and how it might influence the final outcome. This trait of the sampling process is important as it is not expedient to sample multiple times while attaining the same samples. Diversity is just as important as a factor as performance. As can be observed in Figure 3.12 diversity of first steps as well as end results consistently increase, which is not too surprising.

What is interesting to pay attention to, though, is the superlinearity one can observe in the plot. For the first step diversity, the largest increase in diversity can be seen around a temperature of 0.3 and 0.6. For the diversity of the final result, until a temperature of 0.7 the increase is slighlty superlinear until the curve adheres back to the linear line. For diversity, too, the plots suggest that the most interesting and useful temperature to sample from is somewhere between 0.3 and 0.7.

3.6.4 Results

To examine the observations further we run a series experiments. As the previously shown results where conducted on Mistral-7B Instruct v0.3, we now run experiments on this model, too. In Table 3.4 one can see results on the test set. As suspected, a temperature of 0.5 yields strong results 2.88% above baseline.

To verify whether this might a promising direction to move toward, we scale experiments to the instruction-tuned Qwen2-series with 0.5B, 1.5B and 7B parameters [50]. Results can be obserbed in Table 3.5. Unfortunately, the outcome is not as convincing as the previous results on Mistral-7B. For the 0.5B-version one

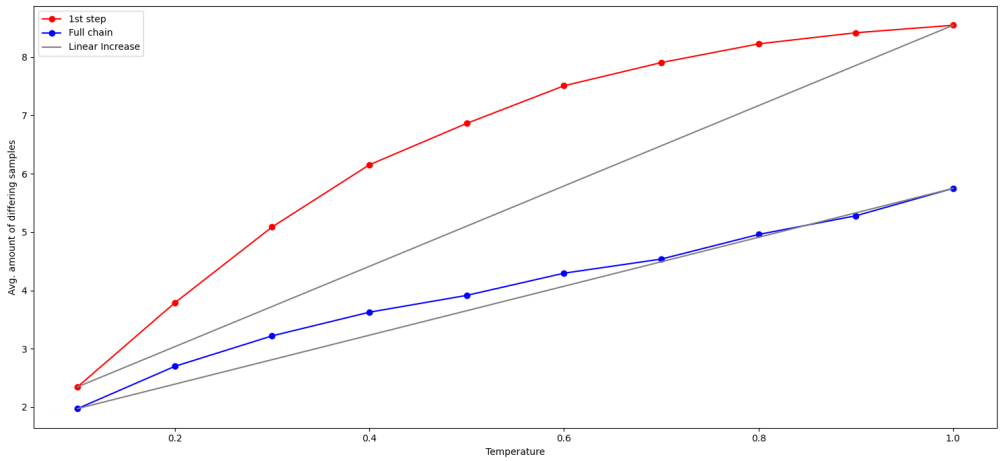

Figure 3.12: As the temperature increases, the amount of differing first step and end results increases as well. Notably, the first step diversity increase significantly faster compared to the end result diversity.

	Temperature			
Baseline	**0.4**	**0.5**	**0.6**	**0.7**
0.4003	0.4018	0.4291	0.4003	0.4041

Table 3.4: MCD results on Mistral-7B-instruct. As suspected, results for a temperature of 0.5 improved the results significantly with an absolute gain of 2.88%.

can see an absolut gain of about 1%. For the 1.5B-version the approach did not yield better results at all and for the 7B-version only slight improvements of $< 1\%$ can be observed.

	0.5B	**1.5B**	**7B**
Temp. / BL	27.52	**50.19**	75.13
0.3	27.07	47.08	**75.81**
0.4	28.27	49.05	74.98
0.5	28.12	48.14	74.07
0.6	26.46	48.52	74
0.7	**28.5**	47.31	74

Table 3.5: MCD results on Qwen2-X-Instruct. Slight improvements can be seen, however, it seems that the method does not deliver drastic changes.

The results from Table 3.4 suggest that for the Mistral-7B model, MCD yielded a significant improvement of 2.88% when the temperature was set to 0.5. This gain highlights that under specific conditions, the MCD approach *can* enhance performance. The increase in accuracy indicates that when the model is allowed to generate diverse reasoning steps at an optimal temperature, it can effectively identify the best solution, leading to better overall results.

However, when testing the method on the Qwen2 series of models, the results were less conclusive. The 0.5B model saw a modest improvement of about 1%, but the 1.5B model did not show any improvement, and the 7B model only exhibited slight gains of less than 1%. These results suggest that while MCD can be effective, its impact may depend on the specific architecture, training data, and fine-tuning process of the underlying model.

While the approach showed promising results in certain configurations, its overall effectiveness varied across different models and datasets. There are, however, a few notable points worth discussing

1. **Importance of temperature and sampling** The temperature experiments show the importance of tuning the sampling process. As shown in Figure 3.11, there is a clear relationship between the temperature parameter and the diversity of the generated reasoning steps. A temperature of 0.5 seemed to strike the right balance between diversity and accuracy. However, there is a high chance that the best temperature varies highly across models. The diversity analysis in Figure 3.12 further supports this. As temperature increases, diversity also increases, but too much diversity at high temperatures (e.g., 0.9 and above) leads to a degradation in performance. This suggests that controlled exploration of reasoning paths is beneficial, but excessive creativity can harm the model's ability to maintain coherence in problem-solving.

2. **Trade-offs Between Reasoning and Judging** The success of the MCD approach relies on the interaction between the reasoner and judge models. By decoupling the generation of reasoning steps (handled by the reasoner) from the evaluation of those steps (handled by the judge), the MCD framework would allow for a more structured and focused problem-solving process. However, this clearly introduces additional complexity. Higher complexity does not always garantuee better performance, plus increases difficulties around implementation.

The inconsistent performance across different models suggests that the MCD approach is not universally applicable. Factors such as model architecture, dataset characteristics, and the complexity of the reasoning tasks all likely play a role in determining the success of MCD. Additionally, the increased computational cost of generating and evaluating multiple candidate steps may not always justify the gains in performance, especially for larger models where reasoning abilities are already strong.

A potential future direction could involve adaptive sampling mechanisms where the number of generated steps or the temperature is dynamically adjusted based on the model or e.g., the problem instance. Moreover, integrating more sophisticated scoring mechanisms for the judge model could lead to better decision-making.

While the MCD approach shows promise, especially for initial exerperiments shown with Mistral-7B, its effectiveness varies and may require further refinement to be broadly applicable. Optimal temperature settings, careful balancing of reasoning and judging, and adaptive sampling strategies could enhance its performance in future iterations.

3.7 First Step Routing

Since naively passing one step to the other model (cf. Sections 3.3 - 3.5) and sampling the first step multiple times to select from it (cf. Section 3.6) did not yield significant results we analyze whether it may be beneficial to route the first step. The principle idea is the following. We assume that there are problem instances, where first step guidance can signifcant improve results, and problem instances where it is simply not needed. Significant performance gains may be possibly while cost barely increases. Thus, the difficulty of this problem boils down to a classification problem. Finding those instances that *do* need that first step guidance is the objective of this section.

In Section 2.4 we briefly analyze prior work investigating LLM routing. Almost all works focus heavily on building a routing mechanism that decides which instances shall be solved by a weaker model and which shall be solved by a strong model. Unlike these works, this work explores routing the first step only instead of the entire sample.

The family of models used here are Qwen2-instruct models for the sizes 0.5B, 1.5B and 7B. We chose this family of models as relationships and differences between model sizes can be neatly analyzed. Prior to setting up the actual experiments we conduct a series of preliminary experiments to shed light on a few determining factors to solve this problem.

3.7.1 Interpolating Small to Large Models

As our first step, we split the chain and distribute it horizontally across two experts E_1 and E_2. We assume E_1 to be the more capable and costly model, and E_2 to be the weaker and less capable model. Given a problem instance, E_1 always solves until step $\mathbf{s_x}$. E_2 then receives the instance and steps solved until step $\mathbf{s_x}$ and concludes until the end. In our specific case, we are especially interested in E_1 solving the first step, however, we investigate the split x at different points to grasp the entire picture. Conducting the priorly explained, leads to an interpolation between those models. The more of the frontal part of the reasoning problem is shifted toward E_1, the more capable model, the more performance converges toward the more capable model E_2's performance, likewise the cost. Accuracies for the interpolations are shown in Table 3.6 and visualized in Figure 3.13.

	7B to 0.5B	**1.5B to 0.5B**	**7B to 1.5B**
Small	27.52	27.52	50.19
$\mathbf{s_1}$	36.92	34.34	56.03
$\mathbf{s_2}$	50.26	41.47	62.32
$\mathbf{s_3}$	61.87	45.64	68.00
$\mathbf{s_4}$	68.46	48.52	72.10
$\mathbf{s_5}$	72.02	49.81	73.77
Large	75.13	50.19	75.13

Table 3.6: Performance for the interpolation between a small and a large models. Model sizes are indicated at the top expressing that in the case of *large to small*, the *large* model generates until step $\mathbf{s_x}$ and passes its result to continue on to the *small* model.

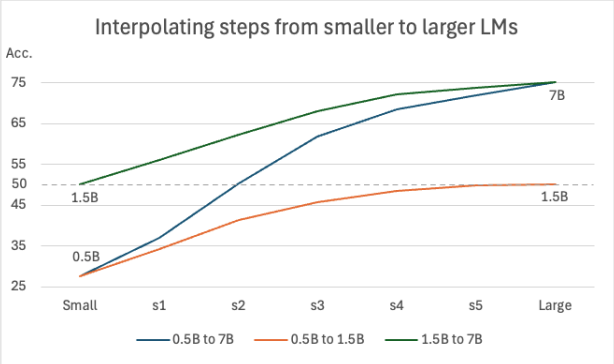

Figure 3.13: Visualization of interpolation from Table 3.6.

3.7.2 Estimating Model Cost

While, unsurprisingly, interpolating reasoning chains between models works, it does not inherently yield a large benefit in terms of cost-value ratio. To examine this further we plot cost against accuracy for all the models and their interpolations. For a fair comparison between models we calculate how long each

model takes to generate all 1319 samples of the GSM8K test set. We do not run tests from scratch to account for overhead and differences in hardware. Instead, for each model and each step we 1) estimate how many tokens will be generated and 2) obtain the amount of tokens each model can generate per second. Mathematically, this can be expressed as

$$c(m) = \sum_{i=1}^{n} s_i(m) \times v(m) \tag{3.4}$$

where $c(m)$ denotes the cost of generating a dataset with model m, n is the amount of reasoning steps for the dataset at hand, $s_i(m)$ is the step amount for step i using model m and $v(m)$ is the speed at which model m generates, i.e the tokens per second. Importantly to note, in the following experiments $v(\text{Avg.})$ will be used for a more robust estimator. Please refert to Section 3.7.1 as a reminder how to how we distribute steps across two models. With this, Equation (3.4) becomes

$$c(m_\text{l}, m_\text{s}) = \sum_{i=1}^{k} s_i(m_\text{l}) \times v(m_\text{l}) + \sum_{i=k+1}^{n} s_i(m_\text{s}) \times v(m_\text{s}) \tag{3.5}$$

where m_l denotes a large model generating until step k, and model m_l taking over from step $k+1$ until n.

Token amount per step

The amount of tokens generated by each model and for each step can be estimated by training a model on the train set and doing inference on the test set. From here, we simply count the total amount of tokens generated per step. Estimated tokens are averaged over all three models to obtain a more robust estimator $v(\text{Avg.})$. In Table 3.7 one can see these numbers for all three Qwen2 models.

i	s_i(0.5B)	s_i(1.5B)	s_i(7B)	s_i(Avg.)
1	46509	48431	46545	47162
2	46312	48178	46609	47033
3	33038	36280	35965	35094
4	18624	20664	22539	20609
5	8014	9272	12528	9938
6	3336	4440	5492	4423
7	1478	2279	2146	1968
8	520	872	847	746

Table 3.7: Amount of generated tokens for all steps of the GSM8K test set for the 0.5B, 1.5B and 7B Qwen2-instruct models. Additionally an average over all three models is calculated which serves as the estimator for later calculations.

Tokens per second

Alibaba benchmarked all Qwen2 models and makes results publicly available. As reference values we consistently used the numbers indicated with *vLLM* and an input length of *6144*, as indicated in the documentation. As of writing this, the 0.5B model can generate $v(0.5\text{B}) = 256.16$ t/s, the 1.5B model can generate $v(1.5\text{B}) = 166.23$ t/s and the 7B model is capable of generating $v(7\text{B}) = 76.4$.

3.7.3 Interpolation Accuracy vs. Cost

Now that we know the amount of tokens generated for each step and tokens per second for each model we can accurately estimate the cost for each model, when previous steps are substituted with more capable but costlier models. Interestingly, when interpolating naively it becomes clear that accuracy versus cost behaves linearly, yielding little benefit for this method. The goal thus will be to increase accuracy while keeping costs caused by a larger model's first steps comparably low. Practically all works described in Section 2.4 aim for optimizing this. In Figure 3.14 one can see the result for steps substituted by a larger model.

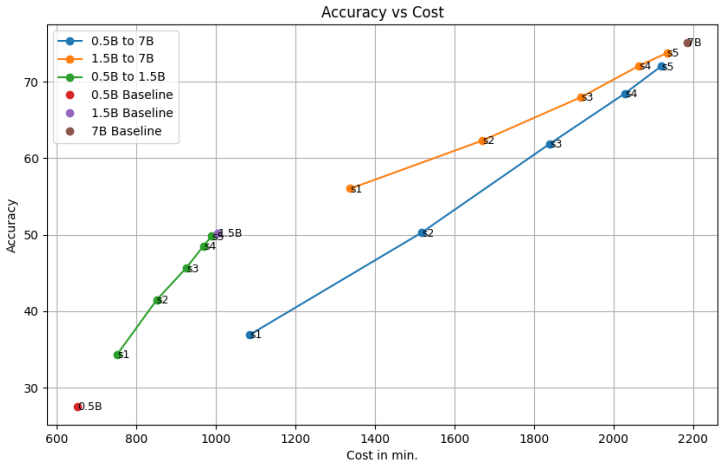

Figure 3.14: Model interpolations costs plotted against their accuracy. The lines between the three baseline models indicate how costs evolve from the smaller to the larger model. sx indicates that until step x steps where generated from the larger model.

First Step Oracle Routing

To verify whether this may be a good direction to move forward we check whether costs can be saved by routing the first steps to a larger model if the smaller model failed through an oracle. The oracle pursues two different strategies. Strategy 1 routes the question to generate the first step if we knew that the *end result* will be predicted wrong by the smaller model, indicated by (R). Strategy 2 routes the question to generate the first step if we knew the *first step* will predicted wrong by the smaller model, indicated by (F).

In the plot in Figure 3.15 the outcome of these strategies can be observed. It becomes visible that oracle first step routing for the 0.5B model (blue) using the 1.5B model (orange) increases baseline by about 10% (brown) if the *end result* was wrong and by about 5% (green) if the *first step* was wrong, with minimally increased costs. Doing oracle first step routing for the 0.5B model (blue) using the 7B model (green) performance slightly drops if the first step was wrong (pink) and is on par with linear interpolation if the *end result* was wrong (red). Doing oracle routing for the 1.5B model using the 7B model increases baseline (orange) by about 12% absolute with only 10% relative more cost if the *end result* was wrong (purple) and by about 4% absolute with only 5% relative more cost if the *first step* was wrong (gray).

3.7.4 Architecture

There are various settings and parameters as to what the architecture and the training procedure for the first step routing could look like. More specifically the router and training data are at the core of what remains

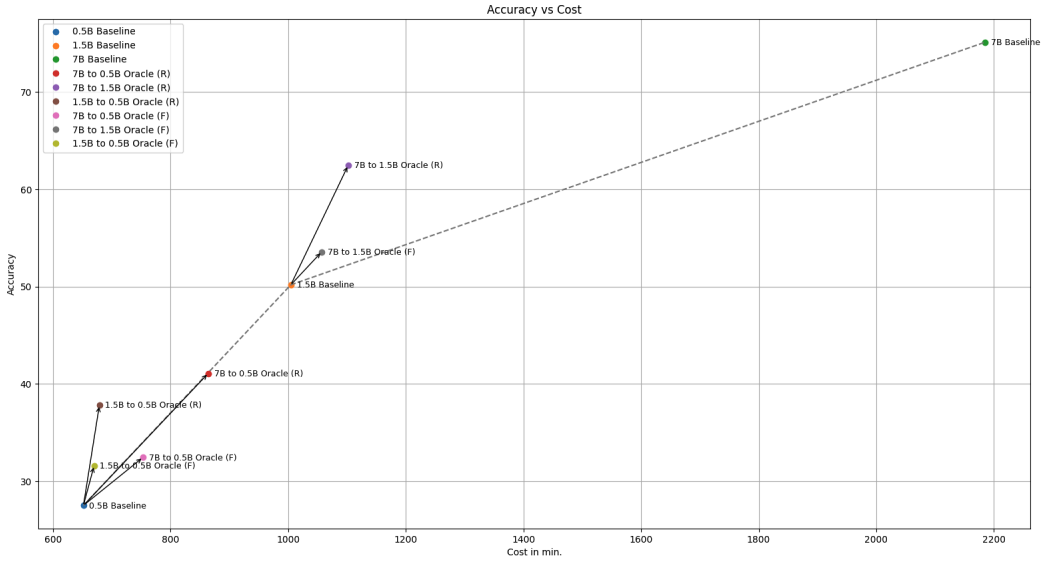

Figure 3.15: The grey shaded lines between baseline models denote the theoretical linear interpolation between models. Models above the curve perform better than baseline, below worse. *large to small* refers to the large model generating step 1 if the oracle decides for a sample that the end result (R, strategy 1) or the first step (F, strategy 2) will be wrong.

to be explored. In all scenarios, though, we want to utilize a router $\mathcal{R}$ which decides whether the first step of an instance should be routed to a strong model $\mathcal{S}$ or a weak model $\mathcal{W}$.

Router

The router needs to optimize for strong performance while maintaining low cost when utilized. While a large model typically yields better results than a small model, it is also more expensive to operate. A strong lightweight model likely is what one wants to opt for. It is far from clear what exactly the router looks like, however, a range of attempts have been described previously. Most methods either describe a deterministic or heuristic-based function [11], a probabilistic function which is *not* a language model, e.g. a Markov process [33, 38, 28], or a neural model, typically a language model [32, 11].

Data

How to obtain training data partially depends on the router. If we choose a router that solely relies on the input signal itself, or a heuristic, we do not need to train the router. If we want to train a more sophisticated model, we do need training data. There are various ways to obtain this type of data. Ong et al. (2024) use human preference data from Chatbot Arena [33], Mohammadshahi et al. (2024) [32] accumulate data points across four question answering datasets and heuristically decide for difficult and easy instances. Ding et al. (2023) [12] obtain training data by passing instances into a small and large language and those that differ the most are considered difficult.

Training

As our router model $\mathcal{R}$ we use Qwen2-7B-instruct. At first, we wanted to use the presumably most powerful model at our disposal. Likely, this model is not the optimal in terms of cost for its performance, but if

experiments go well, one can always shift toward more lightweight options. If the large model fails to predict difficulty well, almost certainly a smaller one will too.

The training data set is defined as $\mathcal{D} = \{x, y\}_{i=1}^{n}$ where x is a question and $y \in \{$<hard>, <easy>$\}$. Put simply, we want to train the model to conduct binary classification on whether an instance is hard or easy. To obtain training data we train a weak model $\mathcal{W}$, in our case Qwen2-0.5B-instruct, on the training dataset and do inference on it. For each inferred training sample we check whether $\mathcal{W}$ predicted correctly or not. If yes, the training sample will be tagged as `<easy>`, if no as `<hard>`. Now, we have at our disposal a binary training dataset $\mathcal{D}$ according to some weak model $\mathcal{W}$. The dataset $\mathcal{D}$ can now be used to train the router $\mathcal{R}$. Whether to finetune $\mathcal{R}$ on the tag y only or the entire sequence solution generating sequence is a design choice. In our case, we finetune the tag *including* the rest of the prediction, assuming to obtain a richer training signal.

Inference

Now that we have a trained router $\mathcal{R}$ we can conduct inference on the test set. For each sample, we generate the tag. If the sample is tagged as `<easy>`, it will be routed to be completely solved by $\mathcal{W}$. If the sample is predicted as `<hard>`, $\mathcal{R}$ will route the sample to the strong model $\mathcal{S}$, which will predict step one of the reasoning chain. Once concluded, $\mathcal{W}$ will take over and finish the prediction, with first-step guidance from $\mathcal{S}$.

3.7.5 Results

For this particular setting we report a result of 28.35% with costs at 685 compared to a baseline 27.67% with costs at 651, resulting in a model that performs below the linear baseline. The result can be observed in Figure 3.16

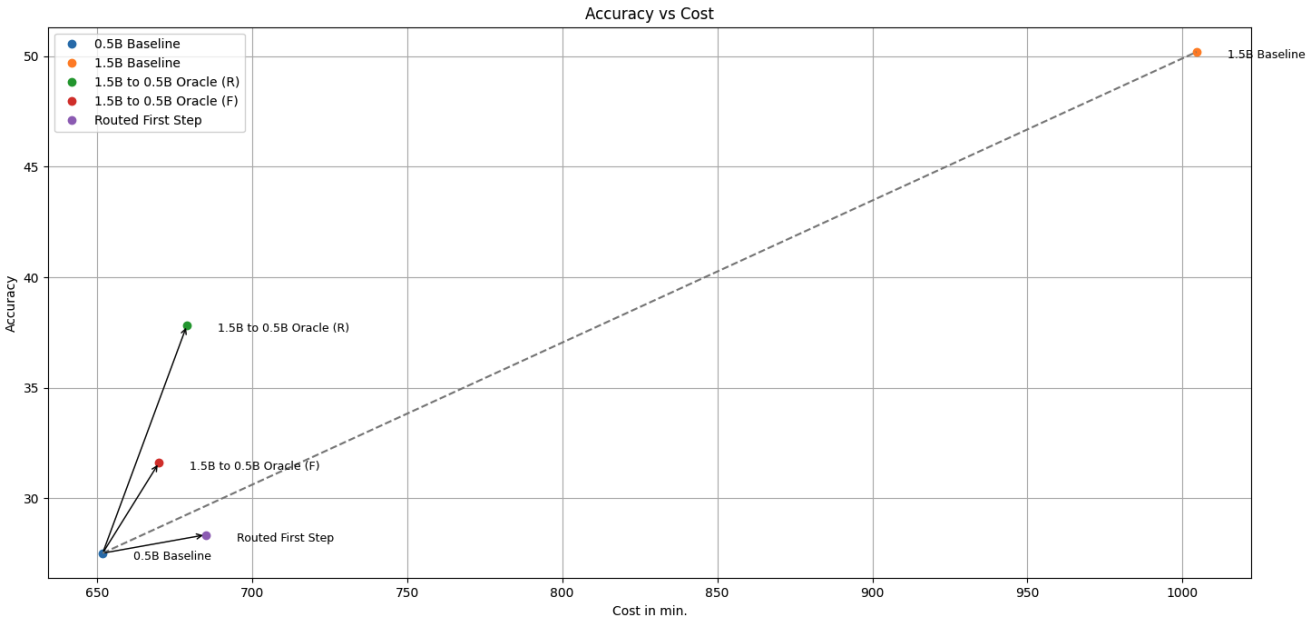

Figure 3.16: Acc. vs. cost change when using a first step router when trained on `<easy>` and `<hard>` tags.

The results from the first-step routing experiment, as illustrated in Figure 3.16, suggest that the proposed method does not yet outperform linear interpolation between models in terms of cost-effectiveness. The

model trained on <easy> and <hard> tags achieved an accuracy of 28.35%, compared to the baseline of 27.67%, with a cost increase from 651 to 685. This results in a marginal improvement in accuracy but a disproportionate increase in computational cost. Our key take-aways are the following.

1. **Marginal Gains at Higher Costs** While the oracle-based first-step routing experiments (see Figure 3.15) show that selective routing can indeed lead to performance gains, our trained router $\mathcal{R}$ failed to achieve similar gains. The small accuracy improvement comes at a relatively high cost, indicating that the routing model might not be making optimal decisions. This suggests that our current training method for $\mathcal{R}$, based on binary classification into <easy> and <hard> categories, may not capture the complexity of the task sufficiently well.

2. **Training Data** One potential issue lies in the way the training data was labeled. The approach of using predictions from a weak model $\mathcal{W}$ to label instances as <easy> or <hard> may be too simplistic. This binary labeling does not account for the nuance in problem difficulty and could lead to suboptimal routing decisions. More sophisticated labeling strategies, such as incorporating a probability-based or confidence-based approach, might yield better results. In Section 2.4 one can see various strategies proposed by other works.

Chapter 4

Ablation Studies

4.1 Mixture-of-Experts Experiments

Besides the CoE architecture we conducted a series of experiments in a classic Mixture-of-Experts (MoE) architecture. The approach differs from the the CoE architecture in the way experts are trained. Previously, examples were divided after a certain amount of steps. For this setting the dataset is simply divided by the amount of steps needed to solve the problem.

In these experiments, we especially investigate 2-, 3- and 4-step problems closer. Models are trained on certain splits of the data where only examples with certain step amounts are seen during training. If a model is called *"2"*, for example, it means that it was trained *only* on 2 step examples. If it is called *"45678"* it means it has seen 4 to 8 step examples during training. Unlike CoE, these experiments divide the actual problem space and do not distribute the problem step-wise over the chain.

4.1.1 Difficult examples help solve easy problems

In the first experimental setting, samples of higher complexity are increasingly added to the models. We begin with a simpler model like a model "2" and continuosly increase the training data set by 3, 4, 5, etc. samples. Looking at Figure 4.1 one can derive one main observation. Across 2-, 3- and 4-step problems performance increases if more complicated trained examples ($>$ 5 steps) are added to training data. This suggests that a model trained on the entire data will always perform better than a specialized expert 2-step-expert only trained on 2-step-problems. Difficult problems help solve easy problems, however, volatility in runs increases and finding a good model becomes harder. This result is rather unsurprising and confirms our intuition. Interestingly, what can be seen for 4-step-problems gives rise to conducting the next series of experiments. For both models tested, it seems like removing the 2- and 3-step-problems benefits the model. In the following we will shed more light on this.

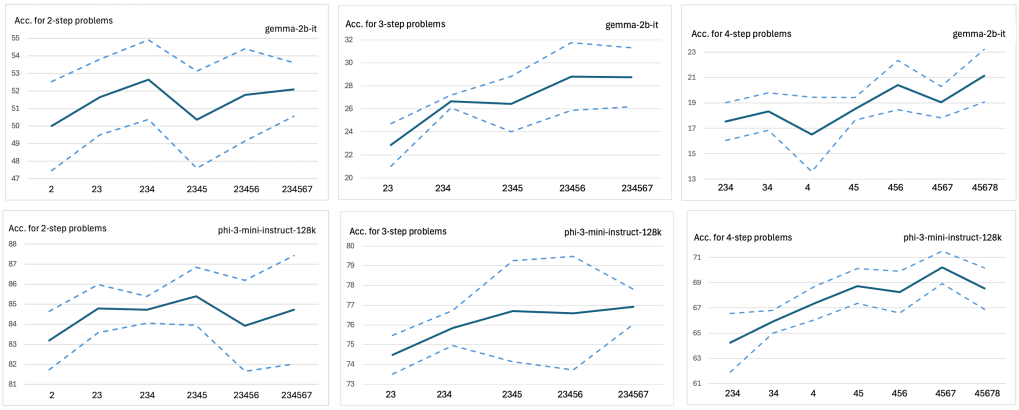

Figure 4.1: Increasing training sets by the samples which are greater than the their the actual problem step-size increases the model performance for both models and all three problem difficulties. This suggests that cutting presumably irrelevant examples does not seem like a good practice. Longer more difficult examples help solve easier ones, which does not seem too surprising.

4.1.2 Easy examples hurt difficult problems

The previous experiments suggest that removing easy problems, from a solver that is supposed to solve hard problems, benefits the model. We test this hypothesis in various settings. In Figure 4.2 one can observe that removing the 2- and 3-step problems compared to keeping them benefits the model in solving 4-step problems. Thus, despite more training data, models seems to perform better when the simpler instances are removed from the training data.

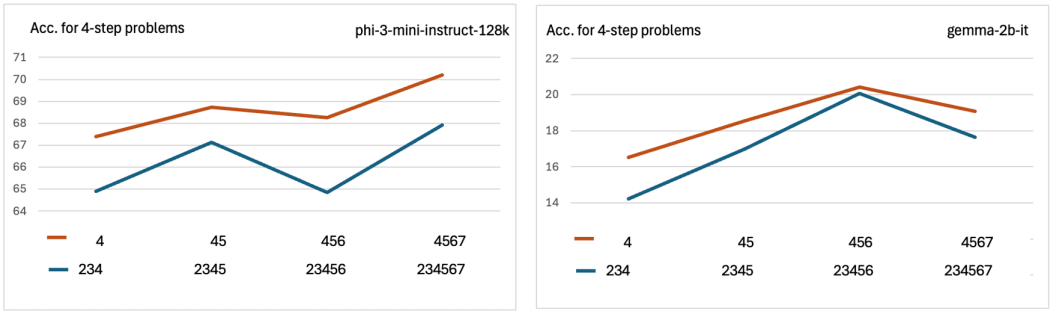

Figure 4.2: Taking away 2 and 3 step examples to solve 4-step problems increases performance. Shorter and easier problems hurt model performance in solving the difficult ones.

4.1.3 Full MoE Experiments

We extend this setting to four models on the entire problem space. For each of the four base models, 8 experts are trained such that expert n sees all x-step training examples where $x \in \{n, \ldots, 8\}$. In other words, for each expert n, all training examples that have less then n steps are cut. The experts are benchmarked against a baseline model that is trained on the full dataset. Importantly to note, these experiments only conduct the expert steps and do not regard the gating mechanism, assuming access to an oracle. In Figure 4.3 the results can be observed visually. The absolute differences can be found in Table 4.1

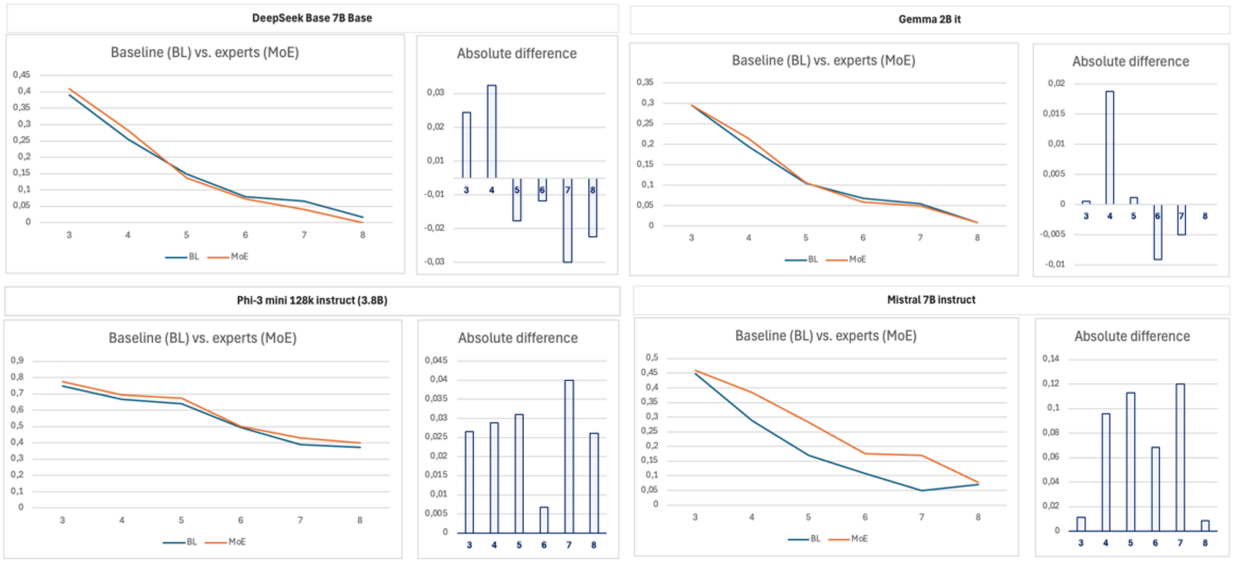

Figure 4.3: Comparison between a fully-trained baseline model and MoE with oracle gating. For Mistral 7B and Phi-3 there are significant gains, for DeepSeek Base 7B and Gemma 2B almost no improvement is visible.

	Gemma 2B	Deepseek 7B	Mistral 7B	Phi-3-Mini
Baseline	0.2728	0.3462	0.3804	0.7051
MoE	**0.2766**	**0.3547**	**0.4285**	**0.7252**

Table 4.1: For all four models, MoE with an oracle outperformed baseline. For Mistral 7B and Phi-3 there are significant gains of **+4.81%** and **+2.02%**, respectively.

4.2 Reasoning Dependency Graphs

In order to understand better why the methods shown in Sections 3.3 to 3.5 do not work well we conduct an error analysis and try to understand better how reasoning problems are structured. For this we sketched out 10s of examples as dependency graphs to reveal structural patterns.

A reasoning dependency graph (RDG) is a directed acyclic graph (DAG) that models humans' reasoning process for arithmetic problems. The graph consists of nodes that can be numbers or math operations. Further, edges model the operation nodes' input and output. A supernode then consists of at least 4 nodes. At least two nodes are the inputs, processed by an operation node and the output is modelled by a node, too. These four nodes, called supernode, make up one reasoning step. Reasoning steps are typically overlapping as the output of one usually is the input for the other.

4.2.1 Definition

Formally, we define a RDG as a DAG $\mathcal{G} = (V = \{A, B\}, E)$ with edges E and nodes V, where $A \cup B = V$ such that $A \cap B = \varnothing$. A denotes the set of aritmetic operations $\{+, -, \times, /, \text{mod}, >, <, \ldots\}$ and $B = \mathbb{R}$ is the set of all entities and their quantities. Within an RDG, the in-degree $d^-(v)$ of a node $v \in A$ (an operation) must be at least 2 and $\forall(w, v) \in E$ it must be that $w \in B$ (an entity).

4.2.2 Structural Analysis

In Figures 4.5 to 4.7 we show selected examples, the dependency graph of their solutions and the textual solution. The aim of this ablation study was to investigate the structure of problem instances. The issue arising is that problem instances are highly non-linear. However, our approach tries to improve model performance linearly, which may lead to a discrepency, hence lower performance. One can see that many instances can indeed not be modelled by a linear RDG but are made up of reather complicated, intertwined relations. At least 50% of GSM8K are made up of non-linear instances. A more in-depth analysis is not conducted.

Linear Instances

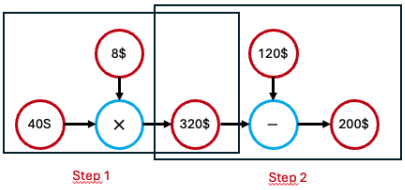

Figure 4.4: **Question:** "John worked for 40 hours at a rate of 8 dollars per hour. After receiving his earnings, he had to pay 120 dollars for equipment rental fees. How much money does John have left after the deductions? **Answer:** John worked for 40 hours at a rate of $8 per hour. Therefore, his total earnings are calculated as $40\,\text{hours} \times 8\,\$/\text{hour} = 320\,\text{dollars}$. John has to pay $120 for equipment rental fees. After the deduction, the remaining amount is $320\,\text{dollars} - 120\,\text{dollars} = 200\,\text{dollars}$.. Thus, John has $200 left after the deduction.

Parallel Instances

Put the two parallel instances in one picture and move text to appendix.

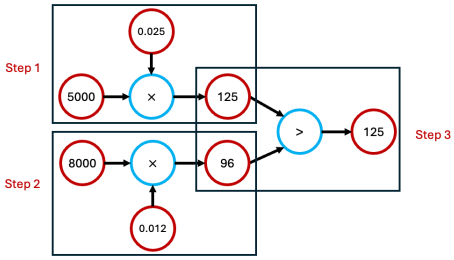

Figure 4.5: **Question:** A merchant wants to make a choice of purchase between 2 purchase plans: jewelry worth $5,000 or electronic gadgets worth $8,000. His financial advisor speculates that the jewelry market will go up 2.5% while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much profit would this be? **Answer:** If he purchases jewelry, he will make a profit of 2.5% which is $5000× (2.5/100) = $125. If he purchases electronic gadgets, he will make a profit of 1.2% which is $8000 × (1.2/100) = $96. If he wants to maximize profit, since $125 ¿ $96, he will choose to purchase jewelry, thereby making a profit of $125.

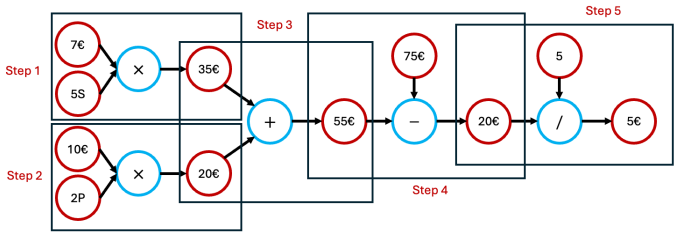

Figure 4.6: **Question:** Ann's favorite store was having a summer clearance. For $75 she bought 5 pairs of shorts for $7 each and 2 pairs of shoes for $10 each. She also bought 4 tops, all at the same price. How much did each top cost? **Answer:** She bought 5 shorts at $7 each so 5 × 7 = $35. She bought 2 pairs of shoes at $10 each so 2 × 10 = $20. The shorts and shoes cost her $35 + $20 = $55. We know she spent $75 total and the shorts and shoes cost $55 which left a difference of $75 - $55 = $20. She bought 4 tops for a total of $20 so $20 / 4 = $5.

Nested Instances

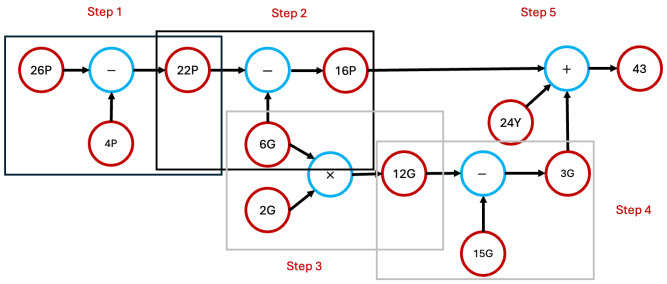

Figure 4.7: **Question:** In a truck, threre are 26 pink hard hats, 15 green hard hats, and 24 yellow hard hats. If Carl takes away 4 pink hard hats, and John takes away 6 pink hard hats and twice as many green hard hats as the number of pink hard hats that he removed, then calculate the total number of hard hats that remained in the truck. **Answer:** If there were 26 pink hard hats and Carl took away 4 pink hard hats, the number of pink hard hats that remained is 26 - 4 = 22. John also took away 6 pink hard hats, leaving 16 pink hard hats in the truck. If John also took twice as many green hard hats as pink hard hats, he took 2 × 6 = 12 green hard hats. The total number of green hard hats that remained in the truck is 15 - 12 = 3. In the truck, after some are taken, there were 3 green hard hats + 16 pink hard hats = 19 hard hats in the truck. Altogether, 19 green and pink hard hats + 24 yellow hard hats = 43 hard hats remained in the truck.

Chapter 5

Conclusion

In this work we implemented and experimented with sequential achitectures chaining together expert models. Each expert model was fine-tuned and trained to focus on a certain part of the solution generating process using LoRA. We tried out a range of settings. Initially, we started with a simple two-model architecture, extended to overlapping the problem context and trained the architectures using teacher forcing. We then moved on to extending the TEC architecture to a judge-reasoner-architecture that aims to exploit variability in the responses models can produce. Lastly, we investigated whether routing the first step to a more capable model may yield benefits.

These are our key take-aways. We mainly comment on architectures that are almost certainly not a good fit, and point to uncertainties and promising next steps.

1. **Too simple** We show that a simple splitting of the reasoning chain and distribution over experts without further extension may be too simple of a model (TEC and OSC). Also, extending it by teacher forcing did not yield signifcant results. Even if results looked a bit more promising the methodology entails lots of engineering work questioning the cost-performance trade-off.

2. **Non-linear nature of the reasoning process** The reasoning process seems non-linear by nature. Possibly, naively splitting the generation process across linear experts simply does not model and distribute information well.

3. **Non-locality of the reasoning process** The reasoning process seems non-local by nature. Possibly, by naively splitting the generation process across linear experts we models lose important global information. This may also imply that the training data is not rich enough for a local expert to perform well on its subtask.

4. **Decoding may work well** For Mistral-7B the MCD approach showed strong results. For the Qwen2 family the approach did not. Possibly one could investigate further why for the Qwen2-family the approach does not work well anymore. Generally, relations around sample efficiency, temperature and a language model's ability to reason well may be interesting to look into further.

5. **First step routing may work well** There is a large body of literature around LLM routing. Due to time constraints we did not further validate the idea, however, as can be seen in the discussion of experiments, it may be a promising direction. Likely, there are many other ways of predicting the difficulty of a problem, besides of what is available at the moment.

A promising direction for future work lies in the refinement of dynamic expert routing and the exploration of more adaptive architectures that can capture the non-linear and global dependencies of reasoning

tasks. While current methods like TEC and OSC are too rigid, strategies like MCD and selective first-step routing show potential but require further optimization. Broadly speaking, key challenges include improving computational efficiency, enhancing expert coordination, and developing more robust methods for problem decomposition. Addressing these issues could significantly advance the capabilities of small models in reasoning tasks without overcomplicating the architecture.

Bibliography

[1] Ethem Alpaydin and Michael I Jordan. Local linear perceptrons for classification. *IEEE Transactions on Neural Networks*, 7(3):788–794, 1996.

[2] Ran Avnimelech and Nathan Intrator. Boosted mixture of experts: An ensemble learning scheme. *Neural computation*, 11(2):483–497, 1999.

[3] Robert C Bartlett, Susan D Collins, et al. *Aristotle's Nicomachean ethics*. University of Chicago Press, 2011.

[4] Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwasniewski, Jürgen Müller, et al. Demystifying chains, trees, and graphs of thoughts. *arXiv preprint arXiv:2401.14295*, 2024.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Nixon Chan and Peter E Kennedy. Are multiple-choice exams easier for economics students? a comparison of multiple-choice and "equivalent" constructed-response exam questions. *Southern Economic Journal*, 68(4):957–971, 2002.

[7] Edward Y Chang. Prompting large language models with the socratic method. In *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0351–0360. IEEE, 2023.

[8] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

[12] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[13] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[14] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

[15] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

[16] Keyur Faldu, Amit Sheth, Prashant Kikani, Manas Gaur, and Aditi Avasthi. Towards tractable mathematical reasoning: Challenges, strategies, and opportunities for solving math word problems. *arXiv preprint arXiv:2111.05364*, 2021.

[17] Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. Specializing smaller language models towards multi-step reasoning. In *International Conference on Machine Learning*, pages 10421–10430. PMLR, 2023.

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[19] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

[20] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[21] Kushal Jain and Kumar Shridhar. First step advantage: Importance of starting right in multi-step reasoning. *arXiv preprint arXiv:2311.07945*, 2023.

[22] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[23] Yichen Jiang and Mohit Bansal. Self-assembling modular networks for interpretable multi-hop reasoning. *arXiv preprint arXiv:1909.05803*, 2019.

[24] Davood Karimi, Haoran Dou, Simon K Warfield, and Ali Gholipour. Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis. *Medical image analysis*, 65:101759, 2020.

[25] Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. Lisa: Reasoning segmentation via large language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9579–9589, 2024.

[26] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[27] Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Junwu Zhang, Munan Ning, and Li Yuan. Moe-llava: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*, 2024.

[28] Aman Madaan, Pranjal Aggarwal, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*, 2023.

[29] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

[30] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42:275–293, 2014.

[31] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

[32] Alireza Mohammadshahi, Arshad Rafiq Shaikh, and Majid Yazdani. Routoo: Learning to route to large language models effectively. *arXiv preprint arXiv:2401.13979*, 2024.

[33] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2024.

[34] Ethan Perez, Patrick Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. Unsupervised question decomposition for question answering. *arXiv preprint arXiv:2002.09758*, 2020.

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[36] Antônio H Ribeiro, Koen Tiels, Luis A Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In *International conference on artificial intelligence and statistics*, pages 2370–2380. PMLR, 2020.

[37] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[38] Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023.

[39] Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7059–7073, 2023.

[40] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *IEEE transactions on neural networks and learning systems*, 2022.

[41] Alessandro Stolfo, Zhijing Jin, Kumar Shridhar, Bernhard Schölkopf, and Mrinmaya Sachan. A causal framework to quantify the robustness of mathematical reasoning with language models. *arXiv preprint arXiv:2210.12023*, 2022.

[42] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[43] Denis Tarasov and Kumar Shridhar. Distilling llms' decomposition abilities into compact language models. *arXiv preprint arXiv:2402.01812*, 2024.

[44] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[45] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[46] Tracey Wilkinson and Hannah Shaw. Are spot test multiple choice questions easier to answer than short answer questions? *The FASEB Journal*, 29:344–1, 2015.

[47] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[48] Lionel Wong, Gabriel Grand, Alexander K Lew, Noah D Goodman, Vikash K Mansinghka, Jacob Andreas, and Joshua B Tenenbaum. From word models to world models: Translating from natural language to the probabilistic language of thought. *arXiv preprint arXiv:2306.12672*, 2023.

[49] Lemeng Wu, Mengchen Liu, Yinpeng Chen, Dongdong Chen, Xiyang Dai, and Lu Yuan. Residual mixture of experts. *arXiv preprint arXiv:2204.09636*, 2022.

[50] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[51] Yao Yao, Zuchao Li, and Hai Zhao. Beyond chain-of-thought, effective graph-of-thought reasoning in language models. *arXiv preprint arXiv:2305.16582*, 2023.

[52] Fei Yu, Hongbo Zhang, Prayag Tiwari, and Benyou Wang. Natural language reasoning, a survey. *ACM Computing Surveys*, 2023.

[53] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

[54] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[55] Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[56] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[57] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023.

[58] Ge Zheng, Bin Yang, Jiajin Tang, Hong-Yu Zhou, and Sibei Yang. Ddcot: Duty-distinct chain-of-thought prompting for multimodal reasoning in language models. *Advances in Neural Information Processing Systems*, 36:5168–5191, 2023.

[59] Weiguo Zheng, Jeffrey Xu Yu, Lei Zou, and Hong Cheng. Question answering over knowledge graphs: question understanding via template decomposition. *Proceedings of the VLDB Endowment*, 11(11):1373–1386, 2018.

[60] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.