Develop an AI model answering coding-related multiple-choice questions (CodeMMLU Challenge)

Vo Hoai Trong

Introduction

The CodeMMLU challenge is to develop an AI model that can answer multiple choice questions related to programming ability. Given a text input (question and options) and an alphanumeric output representing the answer, this is a natural language processing problem. The challenge is held on the Kaggle - a platform that supports tools such as notebooks and GPUs to develop AI/ML applications.

After thoroughly analyzing the challenge requirements and dataset, I adopted a large language model (LLM) approach. By implementing and evaluating various advanced techniques, including model finetuned, retrieval-augmented generation (RAG), autonomous agents, and model parallelism, I systematically optimized my model's performance. Through extensive testing and submission evaluations, my approach yielded competitive results, demonstrating its effectiveness in tackling this complex NLP problem. The results are summarized in the Methodology section (3.5).

Methodology

1. Dataset and Analysis

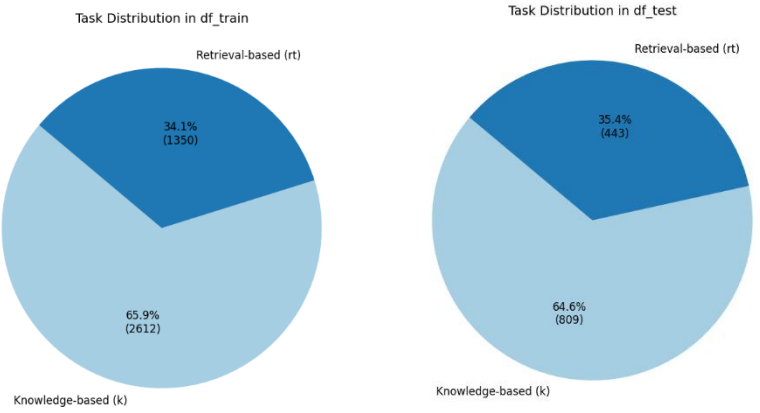
The dataset provided for the CodeMMLU Challenge consists of multiple choice questions covering a variety of programming topics. Each question has multiple answer choices labeled with alphanumeric identifiers. The training dataset also includes answer choices with letter identifiers.

Before developing the models, I conducted an initial analysis to understand the structure and distribution of the dataset:

```
df_train answer unique: ['C' 'A' 'B' 'D' 'ANSWER: C' 'ANSWER: B' 'ANSWER: D' 'ANSWER: A' 'E' 'G'
 nan 'ANSWER:  D']
```

	task_id	question	choices	answer	task_type	task_short
count	3963	3963	3963	3949	3963	3963
unique	3963	3826	3451	11	2	2
top	k10168	Question: What is Conditional Rendering?	['False', 'True']	A	k	Knowledge-based (k)
freq	1	7	22	988	2613	2613

	task_id	question	choices	task_type	task_short
count	1253	1253	1253	1253	1253
unique	1253	1242	1163	2	2
top	k10171	Question: Match the following:	['(1)', '(2)', '(3)', '(4)']	k	Knowledge-based (k)
freq	1	2	7	809	809



Picture 1. Dataset Description

It is easy to see that the training data has a bit of noise amd the amount of data provided is quite small. The difference in numbers between the training and testing sets is not much. This suggests that the approach for developing a new model is not feasible, requiring us to use pre-trained models or lightly finetune them. In addition, fine-tuning with a small amount of data may not be effective when the range of questions is too wide.

One approach mentioned in the research is to use MLP (Embedding) networks. However, the vectors from the Embedding model do not contain features to solve this problem, moreover, the labels are not fixed as A, B, C and D (sometimes only A, B or more E, F)

2. Kaggle Platform and Computational Resources

Since the challenge is hosted on Kaggle, what I can leverage is its cloud-based infrastructure for model development. Kaggle provides:

- GPU/TPU: 1 P100 GPU (16GB Vram) or 2 T4 GPUs (16GB x 2 Vram) for training and inferring models
- Notebooks: Pre-configured Jupyter-based environment with libraries like transformers, torch, and faiss, etc.
- Dataset storage & version control: Makes it easy to track changes in data and test models.

So we can inference or finetune models from 0 to 40B parameters applying Quantization, Lora, Data Parallel techniques...

3. Experimental Approaches

A series of experiments were performed and submitted to the system. Based on the scores of the models in the original data report (CodeMMLU), I decided to use the Qwen, Mistral model family. The models from small to large were tested with and without the techniques applied. All experiments used the same prompt template except for the Agent case.

3.1. Prompting

To make the model understand its task and generate the correct answer format, we need to prompt it.

The answer prompt I used is as follows:

```
You are a programming expert and will answer multiple-choice questions about code.

Read the following question and options carefully and select the **best** answer.


### Response Format:

- Reply with **only** the letter of the correct choice (A, B, C, or D).

- Do **not** provide explanations.


### {}


### Options:

{}


### Response:
```

The Agent verify prompt I used is as follows:

```
You are a programming expert and will verify the student's answer about code.
```

Read the following question and verify

Response Format:

- Reply with **only** TRUE or FALSE

- Do **not** provide explanations.

Question:

{}

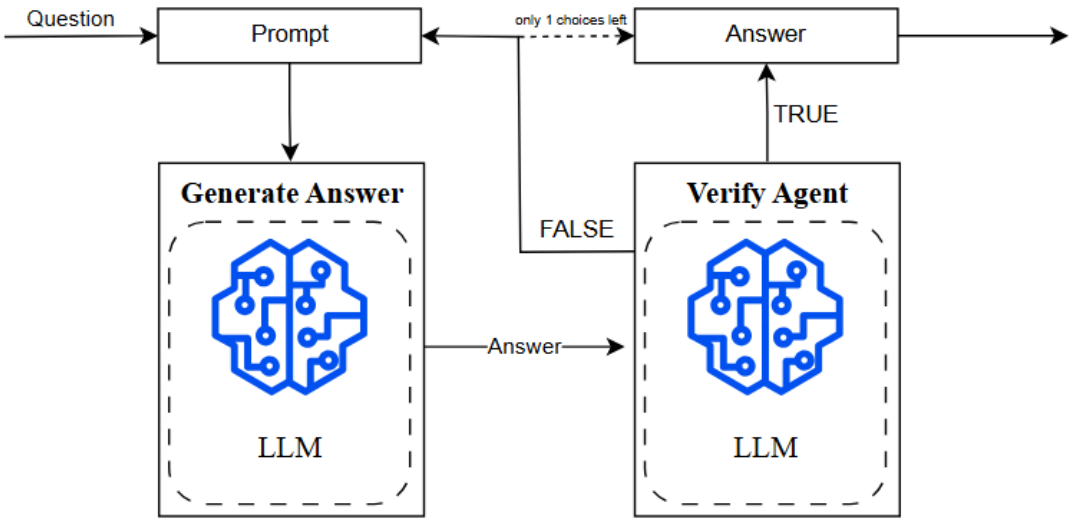
Answer:

{}

Response:

3.2. Agent flow

The idea here is to let the model reconsider whether its answer is right or wrong and respond with a different answer. Consider the Agent's workflow:



Picture 2. Agent Flow

Verify Agent acts as a judge that will eliminate the answers that it thinks are wrong until there is an answer that satisfies it or only the final answer remains. This allows the model to answer itself and verify its own answer to give the most correct answer. However, the limitation of this solution is that if the model is not good, the verification performance is poor, leading to incorrect results. In addition, the inference time increases a lot.

3.3. Finetuned and Inference

Applying Quantization and Lora techniques, even with the limited resources provided by kaggle, we can still finetune and inference the model in the range of 0 - 14B parameters. In addition, using advanced frameworks also greatly supports in saving resources, greatly speeding up training and inference.

The frameworks I used in the experiment are:

- Unsloth: Inference and Finetuned models under 14B parameters
- VLLM: Inference larger models

Finetune parameters:

Parameter	Value	Description
LoRA Parameters		
r	16	Rank of LoRA decomposition. Higher values increase expressiveness but use more memory.
target_modules	["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]	LoRA-applied transformer modules.
lora_alpha	16	Scaling factor for LoRA weights.
lora_dropout	0	Dropout rate for LoRA layers (0 is optimized).
bias	"none"	LoRA bias configuration ("none" is optimized).
use_gradient_checkpointing	"unsloth"	Reduces memory usage, useful for long sequences.
random_state	3407	

		Random seed for reproducibility.
use_rslora	False	Rank-stabilized LoRA (not used).
loftq_config	None	LoftQ quantization (not used).
Training Parameters		
max_seq_length	8092	Maximum sequence length for training.
dataset_num_proc	2	Number of parallel dataset processing workers.
packing	False	If True, speeds up training for short sequences.
Optimizer & Scheduler		
optim	"adamw_8bit"	Optimizer (memory-efficient AdamW).
weight_decay	0.01	L2 regularization term.
lr_scheduler_type	"linear"	Learning rate decay method.
learning_rate	2e-4	Base learning rate.
warmup_steps	5	Steps for linear learning rate warmup.
Batching & Training Steps		
per_device_train_batch_size	8	Batch size per GPU.
gradient_accumulation_steps	4	Steps before optimizer update.
num_train_epochs	1	Number of training epochs.
Precision & Logging		
fp16	not is_bfloat16_supported()	Enables 16-bit precision if BF16 is unsupported.
bf16	is_bfloat16_supported()	Enables BF16 precision if supported.
logging_steps	30	Logging interval.
Output & Reporting		
output_dir	"outputs"	Directory for saving model checkpoints.
report_to	"none"	Reporting framework (e.g., WandB).

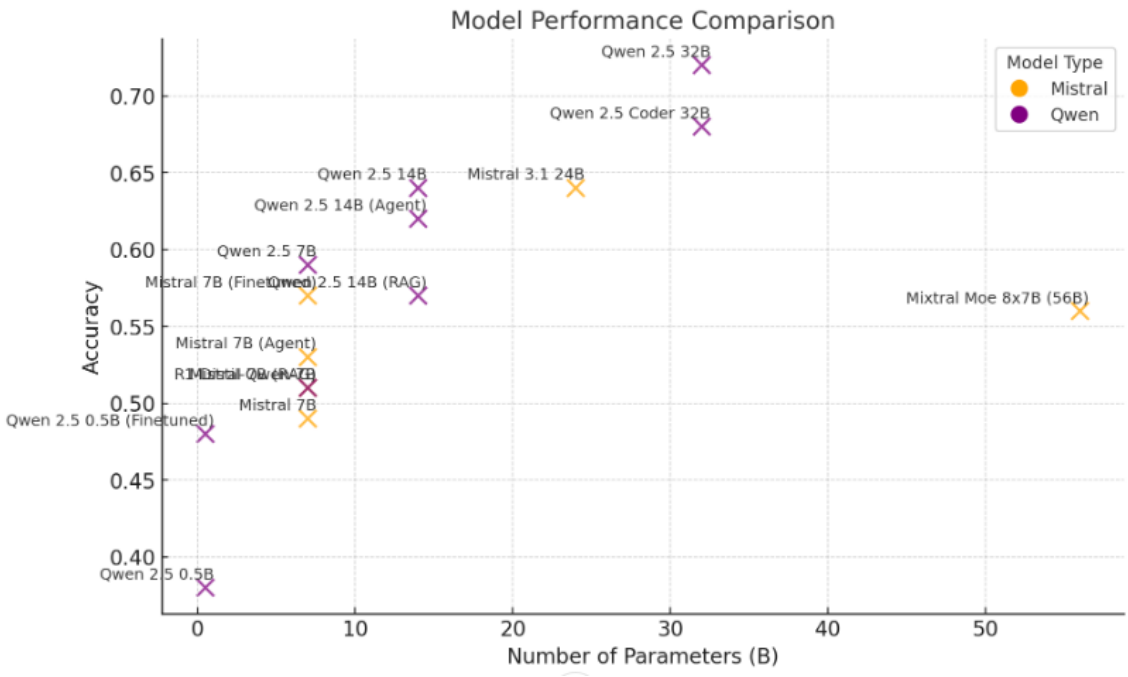
seed	3407	Training seed for reproducibility.

3.4. RAG (Retrieval-Augmented Generation)

With this technique, I will provide examples with answers relevant to the current question that the model needs to answer so that the model can make more accurate inferences. From the training data, I build a vector database with FAISS and query top_k when needed.

3.5. Results Achieved and Analysis

The complete test results are shown in the figure below:



Picture 3. Results Achieved

In general, model accuracy tends to increase as the number of parameters increases. However, more parameters are not necessarily better, as there are cases where a smaller model performs almost as well as a larger model.

About Mistral:

- Mistral 7B has an accuracy of 0.49, but when using techniques such as RAG (0.51), Agent (0.53) and Fine-tune (0.57), the accuracy gradually increases.

- Mistral 3.1 24B (0.64) outperforms Mistral 7B Fine-tuned (0.57), showing that increasing the number of parameters significantly improves performance.
- Mixtral MoE 8x7B (56B) only achieves 0.56, lower than Mistral 3.1 24B (0.64), possibly because Mixtral MoE does not utilize the full number of effective parameters or due to the characteristics of the MoE architecture.

About Qwen:

- Qwen 2.5 0.5B has an accuracy of 0.38, but finetuned it to 0.48.
- Qwen 2.5 7B (0.59) beats Mistral 7B Fine-tuned (0.57), suggesting that Qwen may have an architectural advantage.
- Qwen 2.5 14B (0.64) is on par with Mistral 3.1 24B (0.64), despite having fewer parameters.
- Qwen 2.5 14B with RAG (0.57) and Agent (0.62) shows improvement, but not as good as the original Qwen 2.5 14B (0.64). This may be because RAG or Agent are not well optimized for this problem.
- Qwen 2.5 32B achieved 0.72, the highest of all models, demonstrating a clear benefit from increasing model size.
- Qwen 2.5 Coder 32B (0.68) was slightly lower than Qwen 2.5 32B (0.72), possibly because Coder optimized for source code rather than general multiple choice questions.

At the same parameter level, Qwen seems to outperform Mistral. For example:

- Qwen 2.5 7B (0.59) > Mistral 7B Finetuned (0.57)
- Qwen 2.5 14B (0.64) = Mistral 3.1 24B (0.64) despite fewer parameters
- Qwen 2.5 32B (0.72) is the highest on the list

Finally

- The Qwen 2.5 32B is the strongest model, closely followed by the Qwen 2.5 14B and Mistral 3.1 24B.
- Finetuning is clearly effective for small models, especially with the Qwen 2.5 0.5B and Mistral 7B.
- RAG and Agent improve performance but not always better than the original model.
- Mistral works well but the Qwen 2.5 seems to be more optimized at the same parameter level.

Improvement Idea

1. **Fine-tune with task-specific data by augmenting data or creating data synthesis**

- Idea:

We only fine-tune the data in general, we can train each model on a more specialized data, focusing on possible mathematical tools such as programming, natural language, or multi-task problems. Data can be added from open source code or data augmentation and synthetic generation.

- Reason:

The results show that Qwen 2.5 0.5B Fine-tuned (0.48) > Qwen 2.5 0.5B (0.38), proving that fine-tuning can significantly improve the performance of small models. Qwen 2.5 Coder 32B (0.68) < Qwen 2.5 32B (0.72), possibly because the fine-tuned dataset is not suitable. If the right dataset is chosen, Coder can outperform.

2. **Knowledge Augmentation: Hybrid RAG + Fine-tuning**

- Idea:

Combine Retrieval-Augmented Generation (RAG) with fine-tuned models, instead of using RAG alone.

- Reason:

Mistral 7B RAG (0.51) > Mistral 7B (0.49) but Mistral 7B Fine-tuned (0.57) > RAG (0.51) → Fine-tuning still has the advantage.

Qwen 2.5 14B (0.64) > Qwen 2.5 14B RAG (0.57), maybe because the retrieval data is not suitable or too dependent on RAG.

If Fine-tuning is combined with RAG, it can help the model have both strong retrieval data and optimize internal knowledge.

3. **Mixture of Experts (MoE) Optimization**

- Idea:

Adjust routing strategy in Mixtral MoE to increase efficiency of experts.

- Reason:

Mixtral MoE 8x7B (56B) has 0.56, but still loses to Mistral 3.1 24B (0.64) despite having more parameters.

It could be due to the MoE not being optimized, leading to not being able to take full advantage of each expert's capabilities.

4. **Business Process Optimization (BPO) Fine-tuning**

- Idea:

Apply Business Process Optimization (BPO) with Reinforcement Learning from Human Feedback (RLHF) training to improve model performance and practicality in real-world applications.

- Reason:

The model may perform well on benchmarks but may not be optimal in practice. BPO helps the model focus on tasks with high business value, optimizing resources. Large models such as Qwen 2.5 32B (0.72) > Qwen 2.5 Coder 32B (0.68) show that fine-tuning in a specialized direction may not be effective without a good process.

5. **Model Voting & Ranking: Enhancing Performance Through Model Collaboration**

- Idea:

Instead of relying on just one model, we can deploy multiple models and use voting or ranking methods to choose the best answer.

- Reason

Take advantage of multiple models instead of just one. Reduce errors: If one model gets the answer wrong, another can compensate. Flexibility: Voting strategies can be changed depending on the type of task.