Análise Quantitativa do Trade-off entre Especialização e Generalização em LLMs via Fine-Tuning

André Okimoto, Guilherme Louro, Nicolas Mady
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)
Manaus, Brasil
{andre.okimoto, guilherme.louro, nicolas.gomes}@icomp.ufam.edu.br

I. Introdução

Este trabalho foi realizado para praticar os conhecimentos das aulas de fine-tuning e fazer uma avaliação deste processo em *Large Language Models* (LLMs). Foi feita a implementação do modelo **Mistral-7B-Instruct-v0.2**, treinamento com a base de dados **Spider Dataset** e duas avaliações, uma para a tarefa de Text-to-SQL e outra de Multitarefas (MMLU).

Alguns pontos importantes a se comentar antes de continuar:

1) Utilizamos a base Spider que está disponível na biblioteca *datasets* do python. Essa base possui os mesmos dados do Spider Dataset que há no site oficial da primeira versão.
2) Todos os scripts (notebook jupyter) foram feitos para rodar no google colaboratory, visto que não tínhamos máquina.
3) Nosso segundo modelo com fine-tuning, por algum motivo, estava dando erro durante a inferência (ele foi treinado normalmente). Não sabemos o que houve, principalmente porque os códigos foram os mesmos utilizados nos dois fine tunings. Chegou num ponto em que não conseguimos disponibilizar um segundo modelo funcional.
4) Para tentar compensar um pouco a falta do segundo modelo, nós testamos a tarefa Text-to-SQL também com o primeiro modelo com fine-tuning, mas usando 3-shot.

II. Metodologia

A metodologia deste trabalho foi dividida em quatro etapas principais:

1) **Fine-tuning do modelo de linguagem Mistral-7B na tarefa de Text-to-SQL** usando o dataset Spider, utilizando exclusivamente o *training split*, como requisitado.;
2) **Geração de consultas SQL a partir de questões em linguagem natural** utilizando o spider dev split (ou validation, na biblioteca datasets). Nessa etapa, foram utilizados o modelo base, fine-tuned e fine-tuned com fewshot. Geramos um jsonl com as consultas.;
3) **Avaliação da tarefa de Text-to-SQL** utilizando a nossa métrica customizada do deepeval
4) **Avaliação da regressão de capacidade geral** utilizando o benchmark MMLU com o modelo base e o modelo com fine-tuning.

O pipeline foi dividido em quatro etapas. A primeira etapa correspondeu à execução do fine-tuning do modelo Mistral-7B-Instruct-v0.2 sobre a tarefa-alvo, utilizando exclusivamente o conjunto de treinamento do dataset Spider. Esse conjunto foi carregado com a biblioteca datasets e os exemplos foram convertidos para o formato esperado pelo modelo, utilizando marcações do tipo [INST] e [/INST] envolvendo as perguntas em linguagem natural, seguidas da resposta esperada em SQL.

Para o fine-tuning, foi adotada a técnica de adaptação eficiente de parâmetros conhecida como LoRA (Low-Rank Adaptation). As configurações do LoRA estão disponíveis nas Tabelas I e II. A quantização foi realizada com uso de 4 bits no esquema nf4 e operações em bfloat16. O treinamento foi conduzido utilizando o SFTTrainer da biblioteca TRL, com uma época, taxa de aprendizado de 2e-4, batch size de 64 e acumulação de gradientes a cada quatro lotes. O modelo resultante do fine-tuning foi salvo localmente com o identificador mistral_spider_qlora.

Tabela I
Configurações do Adaptador LoRA utilizadas para o fine-tuning 1 do modelo

Parâmetro	**Valor**
`r`	16
`lora_alpha`	16
`target_modules`	`["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]`
`lora_dropout`	0.05
`bias`	`"none"`
`task_type`	`"CAUSAL_LM"`
`learning_rate`	$2e^{-4}$

A segunda etapa consistiu na geração de consultas SQL dos modelos na tarefa de Text-to-SQL. Foi utilizada a divisão de validação do dataset Spider, que não participou do treinamento. As perguntas foram submetidas ao modelo fine-tuned zero shot e few shot, além do modelo base com quantização de 8 bits, no mesmo formato de prompt utilizado anteriormente.

Tabela II
CONFIGURAÇÕES DO ADAPTADOR LORA UTILIZADAS PARA O FINE-TUNING 2 DO MODELO

Parâmetro	**Valor**
r	16
lora_alpha	16
target_modules	["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]
lora_dropout	0.05
bias	"none"
task_type	"CAUSAL_LM"
learning_rate	$2e^{-2}$

As respostas geradas, representadas por consultas SQL, foram salvas num arquivo jsonl para avaliação com as métricas no script Execution Accuracy.

A terceira etapa foi a avaliação do Text-to-SQL. Ela foi realizada por meio de uma métrica customizada chamada Execution Accuracy, implementada com base no framework DeepEval. Essa métrica compara a consulta SQL gerada pelo modelo com a consulta esperada (ground truth), executando ambas sobre um banco de dados SQLite correspondente ao identificador da questão (db_id). Cada caso de teste é avaliado individualmente e recebe pontuação em caso de acerto, sendo considerado correto se as respostas forem equivalentes do ponto de vista da execução. Os arquivos de entrada utilizados no teste foram gerados durante a inferência sobre o dataset Spider e estão organizados no formato jsonl, contendo os campos db_id, response_query, original_query e question.

Por fim, a quarta etapa teve como foco a análise da regressão de capacidade do modelo com fine-tuning. Essa análise foi conduzida utilizando o benchmark MMLU (Massive Multi-task Language Understanding), carregado pela Hugging Face. Foram selecionadas 150 questões, divididas igualmente entre três grandes domínios: ciências exatas (STEM), humanidades e ciências sociais. A avaliação foi realizada em modo 4-shot, conforme recomendado pelo benchmark, com o modelo recebendo quatro exemplos antes de responder a cada questão. As respostas fornecidas pelo modelo foram comparadas com o gabarito oficial, de forma que apenas a primeira letra da resposta gerada fosse a opção (por exemplo, se o modelo respondeu “B. 1994”, então foi considerado apenas o caractere “B” como resposta).

III. RESULTADOS

A. Text-to-SQL

Primeiramente, vamos avaliar na tarefa Text-to-SQL.

O modelo base com 3-shot conseguiu acertar 20 consultas de 200 do validation na nossa métrica. Em muitos exemplos, ele simplesmente não retornava nada. Alguns casos foram por puro desconhecimento dos campos das tabelas, como:

Consulta do modelo: *SELECT count(*) FROM singers;*
Consulta original: *SELECT count(*) FROM singer*

E teve poucos casos em que o modelo não seguiu o formato pedido ou que alucinou, como por exemplo na Figura 1.

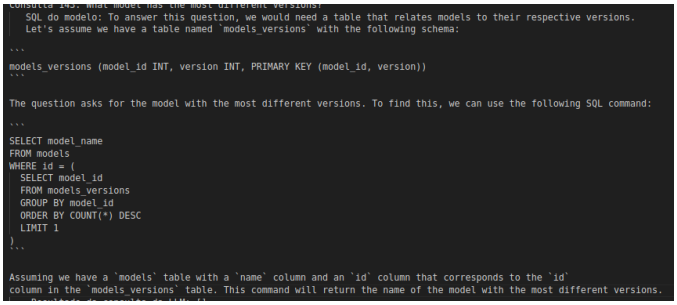

Figura 1. Caso em que o modelo gerou como se fosse uma conversa normal

Já no caso do primeiro modelo com fine-tuning, utilizando essas mesmas 200 amostras, ele conseguiu um score de 28 acertos. Mas há alguns problemas que ocorreram com ele. Primeiro, às vezes o modelo gerava texto até o final do máximo de tokens, como na Figura 2. Ele simplesmente gerava a consulta SQL, colocava o [/INST], e repetia a mesma consulta. Nesse ponto, nós "roubamos"um pouco, e tentamos limpar o texto dele, retirando a primeira consulta SQL gerada. Outra coisa é que esse modelo alucinou muito mais do que o modelo base. Veja algumas das consultas geradas, como a geração de um desenho de uma tabela (Figura 3). Mas também ele teve problemas do modelo base, que não conhecia direito os campos das tabelas (Figura 4).

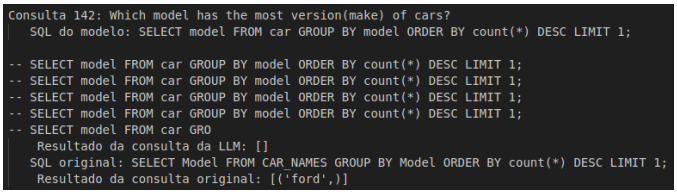

Figura 2. Exemplo de geração contínua de texto, mesmo tendo gerado uma consulta SQL.

Nós também testamos o modelo fine-tuned com fewshot, para compensar o problema que tivemos com o segundo modelo treinado. Ele acertou 20 das 200 consultas, empatando com o modelo base. Ele alucinou bem menos do que o fine-tuned zero shot, mas ainda tiveram algumas pérolas, como a Figura 5.

B. MMLU

Avaliamos o nosso modelo base e o primeiro modelo com fine-tuning no MMLU, com as mesmas 150 questões do teste para cada um. Utilizamos 4-shot e o mesmo prompt pros dois modelos.

Nossa forma de avaliação foi simplesmente pegar a primeira letra do prompt e considerá-lo como a resposta. Ou seja, se a primeira letra for ’A’, essa é a resposta dele.

O modelo base teve um desempenho de 94/150, enquanto que o fine-tuned obteve 66/150. Note que houve uma perda de generalização, com uma diferença de 18 questões acertadas. Uma coisa que foi possível reparar é que o modelo com fine-tuned colocava muitas respostas como opção "S", oriundo de

Figura 3. O modelo desenhou uma tabela usando texto.

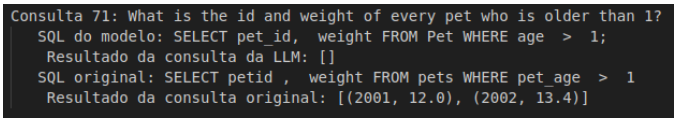

Figura 4. Caso em que o modelo acertou a estrutura, mas não conhecia os dados das tabelas para acertar totalmente.

prompts que tentavam fazer uma consulta SQL com SELECT, como na Figura 6.

IV. Discussão

Infelizmente não conseguimos comparar os hiperparâmetros para avaliar os trade-offs de fine-tuning, como mencionado anteriormente.

Sobre o ganho de especificação, se fosse para escolher entre o modelo base e o nosso fine-tuned, a melhor opção parece ser o modelo base, que teve um desempenho não muito diferente do fine-tuned, mas manteve sua generalização. O few-shot pareceu mais poderoso do que o fine-tuned nesse caso. Mas como testamos somente uma configuração de hiperparâmetros, talvez houvesse uma configuração melhor para conseguirmos resultados mais satisfatórios.

Uma coisa que poderíamos ter feito seria também disponibilizar informações das tabelas no prompt, para que ele errasse menos em relação a sintaxe das tabelas. Ainda sim, podemos afirmar que o uso de few-shot pode ser algo comercialmente bom, já que dependendo do tipo de atividade, não seria necessário fazer um fine-tuning satisfátorio. Bastasse utilizar

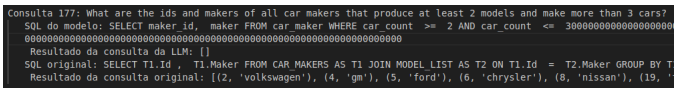

Figura 5. O modelo colocou pra ser menor que um número muito grande

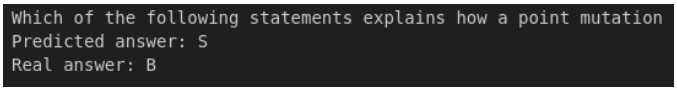

Figura 6. O modelo respondeu 'S', mas a resposta era letra 'B'

alguns exemplos bons para que o modelo já fosse bom naquela tarefa.