Base Model vs Lora Model: Analisando a melhora na task de text-to-sql e perda de capacidades gerais de modelos base e com adaptadores Lora

Caio Cunha, João Nogueira, Leonardo Irias
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)
Manaus, Brasil
{caio.cunha, joao.nogueira, leonardo.cunha}@icomp.ufam.edu.br

I. Introdução

Este relatório descreve a metodologia, os resultados e a discussão de um estudo comparativo entre o modelo `meta-llama/MetaLlama38BInstruct` em sua forma original e a mesma arquitetura após ajuste fino com adaptadores LoRA, mais especificamente QLoRA, na tarefa *text-to-SQL*. Além disso, investigamos o eventual esquecimento catastrófico introduzido pelo ajuste fino, avaliando o desempenho do modelo em tarefas de resposta a perguntas gerais.

II. Metodologia

A. Pré-processamento do Spider

A partir do repositório oficial do *Spider*, obtivemos os *splits* de treino (`train`) e desenvolvimento (`dev`). Com o `gera_base.ipynb` geramos dois arquivos no formato `.jsonl`—`train_formatted.jsonl` e `dev_formatted.jsonl`. Cada exemplo contém:

- o enunciado em linguagem natural;
- a consulta SQL de referência;
- um *prompt few-shot* com três exemplos resolvidos;
- o respectivo *schema* extraído de `tables.json`;
- os *tokens* especiais exigidos pela família Llama.

Ao incluir explicitamente os *schemas* no *prompt* tanto de treino quanto de teste, evitamos que o modelo gere consultas com sintaxe correta, porém com nomes de colunas inexistentes no banco.

B. Inferência do modelo base em text-to-SQL

Carregamos o Llama-3-8B quantizado e realizamos inferência sobre 200 instâncias do `dev_formatted.jsonl`, limitados pela capacidade de GPU no Google Colab. As saídas foram armazenadas e posteriormente avaliadas no `testa_modelo_base_SQL.ipynb`.

C. Avaliação do modelo base no MMLU

Para mensurar conhecimento geral, aplicamos o modelo às 150 questões MMLU selecionadas (STEM, Humanidades e Ciências Sociais). O *prompt* continha quatro exemplos (*4-shot*) e a questão-alvo, e os resultados foram salvos via `testa_modelo_base_MMLU.ipynb`.

D. Primeiro ajuste fino (fine-tuning 1)

O primeiro ajuste fino foi conduzido em ambiente quantizado com QLoRA, empregando `train_formatted.jsonl`. O número máximo de `steps` foi fixado em 50, substituindo as 1 750 iterações necessárias para uma época completa (inviável no cenário de Colab). A perda final foi de 0,144 após 27 min.

Tabela I
Hiperparâmetros do primeiro treinamento QLoRA

Parâmetro	Valor
Rank LoRA (r)	8
`lora_alpha`	16
`target_modules`	{`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`}
`lora_dropout`	0.05
`num_train_epochs`	1
`max_steps`	50
Otimizador	`AdamW` (`torch`)
Taxa de aprendizado	2×10^{-4}
`max_seq_length`	512

O código correspondente encontra-se em `fine_tuning_modelo.ipynb`.

E. Segundo ajuste fino (fine-tuning 2)

Mantivemos todos os hiperparâmetros do treinamento anterior, exceto `max_steps`, ampliado para 200. O treinamento demandou 2 h e atingiu perda final de 0,0416, também registrado em `fine_tuning_modelo.ipynb`.

F. Inferência dos modelos ajustados em text-to-SQL

Repetimos a avaliação da Subsec. B (*text-to-SQL*) empregando o modelo base mais cada adaptador QLoRA. As mesmas 200 instâncias foram processadas; as saídas encontram-se em `testa_modelo_lora_SQL.ipynb`.

G. Inferência dos modelos ajustados no MMLU

Da mesma forma, testamos o impacto do ajuste fino em conhecimento geral reaplicando o MMLU às 150 questões, agora com cada adaptador. A acurácia foi calculada a partir dos arquivos de saída gerados pelo `testa_modelo_lora_MMLU.ipynb`.

H. Implementação da métrica de avaliação

Para *text-to-SQL*, implementamos a métrica `ExecutionAccuracyMetric` no framework *DeepEval*. A métrica conecta-se dinamicamente ao banco SQLite de cada questão do *Spider*, executa com segurança a consulta gerada e a de referência, converte os resultados em `sets` Python e computa a pontuação com base em equivalência exata. Todo o fluxo é orquestrado por um *script* que associa perguntas, bancos e respostas, produzindo um relatório JSON com a acurácia global.

III. Resultados

A. Desempenho em text-to-SQL

Tabela II
Acurácia de execução (# consultas corretas) na tarefa *text-to-SQL*

Modelo	Consultas corretas / 200	Acurácia (%)
Llama-3-8B (base)	104	52.0
+ QLoRA #1 (50 steps)	103	51.5
+ QLoRA #2 (200 steps)	102	51.0

Observa-se que o modelo original obteve ligeira vantagem, apenas dois acertos a mais que a configuração mais extensa de ajuste fino. A diferença absoluta (1–2) indica que o treinamento realizado não resultou em melhoria mensurável e, de fato, introduziu uma pequena degradação. A Seção Discussão explora possíveis causas, incluindo o número reduzido de passos e a limitação computacional imposta pelo ambiente Colab.

Vendo os erros retornados na avaliação, podemos notar que o principal tipo de erro ocorre por "Sintaxe da query predita falhou: no such column: T2.Name", pois o modelo tenta acessar uma coluna que não existe na tabela. Esse erro provavelmente ocorre pois o modelo chama a coluna por um nome diferente do definido no schema da tabela.

B. Desempenho no MMLU

Tabela III
Acurácia no *MMLU* (150 questões)

2*Modelo (lr)3-5	Total (%)	Acurácia por domínio (# corretas / 50) STEM	Humanidades	C. Sociais
Llama-3-8B (base)	44.0	17 (34%)	17 (34%)	32 (64%)
+ QLoRA #1 (50 steps)	44.7	20 (40%)	17 (34%)	30 (60%)
+ QLoRA #2 (200 steps)	47.3	21 (42%)	22 (44%)	28 (56%)

Curiosamente, a configuração com maior número de `steps` (#2) superou tanto o modelo base quanto a versão com ajuste fino reduzido (#1), contrariando a hipótese de degradação crescente. Já o modelo sem ajuste foi o pior dos três. Não sabemos exatamente como explicar esse resultado, mas na seção Discussão falamos um pouco sobre o que pode ter acontecido.

IV. Discussão

A. Desempenho em text-to-SQL

Os resultados apresentados na Seção III-A indicam que o ajuste fino realizado com QLoRA praticamente não alterou a acurácia de execução: a variação máxima entre os três modelos foi de apenas dois acertos em duzentas consultas. Essa diferença mínima é compatível com flutuações estocásticas de geração e sugere que o treinamento, limitado a 50 e 200 `steps`, não foi suficientemente intenso para incorporar conhecimento adicional relevante.

Dois fatores operacionais provavelmente contribuíram:

1) **Comprimento do *prompt*.** O *schema* das tabelas ocupa grande parte do contexto. Com o limite de 512 `tokens` imposto durante o treinamento, porções do *prompt* poderiam ter sido truncadas, dificultando a aprendizagem. Tentativas de elevar `max_seq_length` para 1 024 tokens esbarraram em limitações de memória (erro CUDA) no ambiente Google Colab.
2) **Ausência de replicações.** Uma única rodada de inferência; executar pelo menos três repetições reduziria a variância inerente ao processo de amostragem, mas tal estratégia não foi viável sob as restrições de tempo e GPU.

Em síntese, um *fine-tuning* mal dimensionado tende não apenas a falhar em aprimorar o desempenho, como pode introduzir degradação na task.

B. Desempenho no MMLU

Conforme a Tabela III, o modelo submetido ao treinamento mais longo (`max_steps` = 200) superou as demais variantes, inclusive o modelo base, contrariando a hipótese de esquecimento catastrófico incremental. Embora o fenômeno não tenha sido investigado a fundo, temos a seguinte hipótese: o processo de *fine-tuning*, mesmo focado em *text-to-SQL*, envolve gerar respostas estruturadas a perguntas. Esse "treino para responder" pode ter proporcionado uma leve regularização que beneficiou o modelo em perguntas de conhecimentos gerais, sobretudo porque a perda total permaneceu baixa.

C. Implicações para o processo de fine-tuning

Os achados reforçam a necessidade de planejamento rigoroso para qualquer etapa de ajuste fino:

- **Dados.** Garantir que o conjunto de treinamento esteja balanceado e adequadamente formatado para evitar desperdício de contexto com informações redundantes.
- **Hiperparâmetros.** Definir número de passos, taxa de aprendizado e comprimento de sequência compatíveis com a capacidade de hardware, evitando tanto subtreinamento quanto *overfitting*.
- **Avaliação robusta.** Adotar réplicas de inferência e múltiplas métricas para detectar variações aleatórias e efeitos colaterais indesejados.

Sem tais cuidados, corre-se o risco de dedicar recursos computacionais a um processo que, em vez de aprimorar, degrada o modelo em tarefas críticas.