

TP4: Análise Quantitativa do Trade-off entre Especialização e Generalização em LLMs via Fine-Tuning

Lucas Castro de Souza ¹

¹Universidade Federal do Amazonas - PPGI

I. OBSERVAÇÕES

Algumas observações importantes sobre o trabalho:

- O repositório com o código do trabalho está disponível em [bla.com](#).
- Os checkpoints do LoRA e os resultados intermediários estão disponíveis em <https://drive.google.com/drive/folders/1VXzIdSLtCn5-leHEG6usvXw60Y5OzKKv?usp=sharing> (*Link*).
- O parâmetro variado no experimento foi o número de épocas de fine-tuning. Foram avaliados o modelo base (sem fine-tuning), o modelo fine-tuned por 3 épocas e o modelo fine-tuned por 5 épocas.
- O modelo foi fine-tunado utilizando o dev split do Spider. Eu só percebi após finalizar que a especificação pedia o train split.
- Para a avaliação na tarefa de Text-to-SQL, utilizei apenas as 150 primeiras instâncias do Spider dev split. Isso ocorreu porque eu estava na terceira (e última) conta do Colab e não seria possível avaliar todas as 1034 instâncias. Para executar com todas as instâncias, basta alterar o valor da variável `EVALUATION_ENTRIES_SIZE` em `common.py` para 1034.

II. METODOLOGIA

A seguir, está detalhada cada etapa do pipeline experimental deste trabalho, incluindo arquitetura dos scripts, o pré-processamento dos dados, configuração do modelo base, a configuração do treinamento com LoRA, e a implementação da métrica customizada de avaliação.

A. Arquitetura dos Scripts

O projeto foi organizado em seis scripts principais. Cada um é responsável por uma etapa específica do pipeline experimental, eles foram executados na mesma ordem em que são apresentados aqui. A seguir, uma breve descrição de cada script:

- **0_preparando_dados.py**: Responsável pelo pré-processamento dos dados. Este script carrega o Spider dataset, extrai e formata os exemplos para o formato de chat utilizado no fine-tuning e avaliação. Também prepara e salva o conjunto de questões do MMLU, garantindo a divisão correta por categorias.
- **1_fine-tuning.py**: Realiza o treinamento supervisionado do modelo base utilizando LoRA. Implementa o pipeline de tokenização, configuração dos hiperparâmetros do LoRA, argumentos de treinamento e execução do processo de fine-tuning, salvando checkpoints intermediários.
- **2_calculando_baseline.py**: Calcula o desempenho de baseline do modelo base (sem fine-tuning) na tarefa de Text-to-SQL. Utiliza exemplos 3-shot e executa as queries geradas, comparando com as respostas de referência para medir a acurácia inicial.
- **3_avaliacao_customizada.py**: Avalia os modelos fine-tuned utilizando a métrica customizada *ExecutionAccuracy* utilizando o framework DeepEval. Gera prompts, executa as queries SQL produzidas pelo modelo e compara os resultados com o ground truth, salvando checkpoints e resultados detalhados.

- **4_analise_mmlu.py**: Mede a generalização dos modelos (base e fine-tuned) no benchmark MMLU. Gera prompts 4-shot para questões de múltipla escolha, executa a inferência e calcula métricas de acurácia e desvio padrão por categoria.

Para garantir a reprodutibilidade dos experimentos, todos os seeds dos geradores de números aleatórios foram fixados em 42. Sempre que possível, opções determinísticas foram selecionadas nos frameworks utilizados para minimizar variações entre execuções.

B. Pré-processamento dos Dados

O pré-processamento dos dados é necessário para garantir a compatibilidade dos datasets com o modelo e os frameworks utilizados. Inicialmente, o Spider dataset foi carregado a partir do arquivo original `dev.json`. Para cada entrada, foi extraída a questão em linguagem natural, a query SQL de referência e o esquema do banco de dados correspondente. O esquema foi reconstruído em formato textual, listando todas as tabelas e colunas presentes no banco, de modo a fornecer ao modelo o contexto necessário para a geração da query. Veja a Figura 1 para um exemplo de entrada do Spider dataset após o pré-processamento.

```
{
  "db_id": "concert_singer",
  "question": "How many singers do we have?",
  "query": "SELECT count(*) FROM singer",
  "db_schema": "Database: concert_singer\nTable stadium
(Stadium_ID, Location, Name, Capacity, Highest, Lowest, Average)
\nTable singer (Singer_ID, Name, Country, Song_Name,
Song_release_year, Age, Is_male)\nTable concert (concert_ID,
concert_Name, Theme, Stadium_ID, Year)\nTable singer_in_concert
(concert_ID, Singer_ID)\n"
},
```

Figura 1: Exemplo de entrada do Spider dataset após pré-processamento

Cada exemplo foi então convertido para o formato de chat, seguindo o padrão de prompts utilizado por modelos da família Qwen. O prompt inclui um bloco `system` com a instrução geral, um bloco `user` contendo o esquema do banco e a questão, e um bloco `assistant` com a query SQL de referência (no caso do treinamento) ou vazio (no caso da inferência). A Figura 2 ilustra a função que cria o prompt de inferência para uma consulta do Spider.

```
def spider_to_sql_inference_prompt(entry):
    return (
        f"<|im_start|>system\n"
        f"You are an expert text-to-SQL assistant.<|im_end|>\n"
        f"<|im_start|>user\n"
        f"Given the following database schema:\n"
        f"{entry['db_schema']}\n"
        f"Generate the SQL query for the following question:\n"
        f"{entry['question']}<|im_end|>\n"
        f"<|im_start|>assistant\n"
    )
```

Figura 2: Função que cria um prompt de inferência para uma consulta do Spider.

Para a avaliação de generalização, foi utilizada uma amostra de 150 questões do benchmark MMLU, divididas igualmente entre as categorias STEM (elementary_mathematics), Humanidades (philosophy) e Ciências Sociais (management). As questões foram selecionadas de subcategorias específicas e processadas para o formato de prompt compatível, incluindo o enunciado da questão e as alternativas de resposta. A Figura 3 ilustra o formato de uma questão do MMLU após o processamento.

```

"question": "What is T-group training?",
"subject": "management",
"choices": [
  "A group whose aim is transformational change",
  "A group brought together to deliver training programmes",
  "Team training for the purposes of advancing technology",
  "Team building activities involving learning"
],
"answer": 3,
"category": "Ciências Sociais"

```

Figura 3: Formato de uma questão do MMLU após ser processada.

C. Configuração do Modelo Base

O modelo base utilizado neste experimento foi o Qwen/Qwen2-7B-Instruct. A escolha deste modelo se deve a ser um modelo de 7 bilhões de parâmetros e ter boa performance no benchmark MMLU-Pro.

Ele foi carregado de forma quantizada pois ele não cabe sem quantização na GPU grátis do Google Colab (T4 15GB), que foi utilizada para o treinamento e avaliação. Foi utilizado também, `float16` pois a GPU grátis do Google Colab não suporta `bfloat16`. Veja Figura 4 para detalhes da configuração de carregamento do modelo.

```

# Model loading configuration
model_kwargs = {
    "device_map": "auto",
    "torch_dtype": torch.float16,
}

if quantization == "4bit":
    model_kwargs["quantization_config"] = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_use_double_quant=False,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
    )

# Load model
model = AutoModelForCausalLM.from_pretrained(model_path, **model_kwargs)

```

Figura 4: Configuração de carregamento do modelo em modo quantizado.

D. Configuração do Treinamento com QLoRA

O fine-tuning foi realizado utilizando o método QLoRA (Quantized Low-Rank Adaptation). Foram adaptados apenas os módulos de atenção do modelo (`q_proj`, `k_proj`, `v_proj`, `o_proj`), reduzindo o número de parâmetros treináveis e acelerando o processo de fine-tuning. A configuração do LoRA adotada está detalhada na Figura 5.

E. Configuração do Treinamento

O treinamento foi conduzido por 5 épocas, com batch size efetivo de 8 (batch size 2 e *gradient accumulation steps* 4), taxa de aprendizado inicial de 2×10^{-4} , otimizador AdamW em modo 8-bit.

Os checkpoints do LoRA foram salvos a cada época para permitir a avaliação do finetuning em diferentes épocas, como pedido na especificação do trabalho. Veja a Figura 6 para detalhes da configuração do treinamento.

```
lora_config = LoraConfig(
    r=4,
    lora_alpha=8,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
```

Figura 5: Configuração LoRA.

```
training_args = TrainingArguments(
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    num_train_epochs=5,
    learning_rate=2e-4,
    fp16=True,
    bf16=False,
    group_by_length=True,
    output_dir=output_dir,
    save_strategy="epoch",
    eval_strategy="no",
    logging_strategy="steps",
    logging_steps=10,
    report_to="none",
    seed=SEED,
    gradient_checkpointing=True,
    optim="adamw_8bit",
    lr_scheduler_type="cosine",
    warmup_ratio=0.1,
    max_grad_norm=1.0,
)
```

Figura 6: Configuração do Treinamento.

F. Implementação da Métrica Customizada: ExecutionAccuracy

Para avaliar a geração de consultas SQL, foi implementada a métrica customizada **ExecutionAccuracy**, compatível com o framework DeepEval. Essa métrica executa tanto a query gerada pelo modelo quanto a de referência no mesmo banco SQLite, atribuindo score 1.0 se os resultados coincidirem (ignorando ordem e duplicatas) e 0.0 caso contrário ou em caso de erro de execução. Mensagens de erro são registradas para análise posterior. A Figura 7 mostra o núcleo da implementação.

```
actual_result = self.execute_sql_query(db_path, test_case.actual_output)
expected_result = self.execute_sql_query(db_path, test_case.expected_output)

if actual_result["success"] and expected_result["success"]:
    if set(list(actual_result["data"])) == set(list(expected_result["data"])):
        self.score = 1.0
        self.reason = "Correct result."
    else:
        self.score = 0.0
        self.reason = "Incorrect result."
```

Figura 7: Parte principal da função que avalia os resultados.

III. RESULTADOS

A. Desempenho na Tarefa Text-to-SQL (Spider)

A Tabela I apresenta as acurácias médias e desvios padrão dos modelos avaliados na tarefa de Text-to-SQL, considerando as 150 primeiras instâncias do Spider dev split.

Tabela I: Resultados na tarefa Spider (Text-to-SQL)

Modelo	Épocas	Acurácia	Desvio Padrão
Qwen2-7B-Instruct (base)	0	0.480	0.501
Qwen2-7B-Instruct-LoRA	3	0.793	0.406
Qwen2-7B-Instruct-LoRA	5	0.907	0.292

A variação percentual de acurácia entre o modelo base e o modelo fine-tuned por 5 épocas foi de:

$$\Delta_{\text{acurácia}} = \frac{0.907 - 0.480}{0.480} \times 100 \approx 89.0\%$$

Ou seja, o **fine-tuning proporcionou um aumento de aproximadamente 89% na acurácia na tarefa-alvo.**

B. Desempenho em Generalização (MMLU)

A Tabela II apresenta os resultados agregados e por categoria no benchmark MMLU, incluindo acurácia média e desvio padrão.

Tabela II: Resultados no MMLU (generalização por categoria)

Modelo	Agregada	STEM	Ciências Sociais	Humanidades
Base	0.300 ± 0.016	0.320 ± 0.466	0.280 ± 0.449	0.300 ± 0.458
LoRA (3 épocas)	0.373 ± 0.038	0.400 ± 0.490	0.400 ± 0.490	0.320 ± 0.466
LoRA (5 épocas)	0.393 ± 0.057	0.460 ± 0.498	0.400 ± 0.490	0.320 ± 0.466

A variação percentual de acurácia agregada entre o modelo base e o modelo fine-tuned por 5 épocas foi de:

$$\Delta_{\text{acurácia, MMLU}} = \frac{0.393 - 0.300}{0.300} \times 100 \approx 31.0\%$$

Ou seja, **houve um aumento de aproximadamente 31% na acurácia agregada no MMLU após o fine-tuning.**

A Tabela III detalha a variação percentual por categoria:

Tabela III: Variação percentual de acurácia por categoria (MMLU, base → LoRA 5 épocas)

Categoria	Base	LoRA 5 épocas	Variação (%)
STEM	0.320	0.460	+43.8%
Ciências Sociais	0.280	0.400	+42.9%
Humanidades	0.300	0.320	+6.7%

Observa-se que o ganho de acurácia foi mais expressivo nas categorias STEM e Ciências Sociais, enquanto em Humanidades o aumento foi modesto.

C. Análise de Erros

Apesar do aumento significativo de desempenho, o modelo fine-tuned ainda apresenta falhas em alguns casos. A seguir, foram destacados dois exemplos de erro do modelo após 5 épocas de fine-tuning na tarefa Text-to-SQL:

- **Input:** What is the count of the car models produced in the United States?

Output gerado:

```
SELECT count(*) FROM CAR_MAKERS AS T1 JOIN COUNTRIES AS T2 ON T1.Country =
T2.CountryId WHERE T2.CountryName = 'United States'
```

Output esperado:

```
SELECT count(*) FROM MODEL_LIST AS T1 JOIN CAR_MAKERS AS T2 ON T1.Maker =
T2.Id JOIN COUNTRIES AS T3 ON T2.Country = T3.CountryId WHERE
T3.CountryName = 'usa';
```

- **Input:** Which countries in europe have at least 3 car manufacturers?

Output gerado:

```
SELECT T1.CountryName FROM COUNTRIES AS T1 JOIN CONTINENTS AS T2 ON
T1.Continent = T2.ContId JOIN CAR_MAKERS AS T3 ON T1.CountryId =
T3.Country WHERE T2.Continent = 'Europe' GROUP BY T1.CountryName HAVING
COUNT(*) ≥ 3;
```

Output esperado:

```
SELECT T1.CountryName FROM COUNTRIES AS T1 JOIN CONTINENTS AS T2 ON
T1.Continent = T2.ContId JOIN CAR_MAKERS AS T3 ON T1.CountryId =
T3.Country WHERE T2.Continent = 'europe' GROUP BY T1.CountryName HAVING
count(*) ≥ 3;
```

Nestes exemplos, o modelo errou ao utilizar valores incorretos para os nomes de países ('United States' vs 'usa' e 'Europe' vs 'europe'). Isso provavelmente ocorreu porque, embora o esquema das tabelas seja fornecido, os valores possíveis para cada coluna não são explicitados no contexto do prompt.

IV. DISCUSSÃO

Era esperado que o modelo apresentasse desempenho inferior após o fine-tuning para Text-to-SQL devido ao fenômeno de "esquecimento catastrófico". No entanto, os modelos fine-tunados melhoraram em todas as três categorias avaliadas no MMLU. O maior ganho ocorreu em STEM, com variação de +43,8% em relação ao modelo base.

Alguns fatores podem explicar esses resultados:

- A tarefa de conversão de texto para SQL exige raciocínio lógico e compreensão estrutural. Isso pode ter favorecido a transferência de aprendizado para áreas com demandas cognitivas semelhantes, como STEM (matemática) e Ciências Sociais (economia), resultando em ganhos nessas categorias. Em Humanidades (filosofia), que demanda outros tipos de raciocínio (eu acho), o impacto foi menor.
- O fine-tuning foi realizado apenas com o dev split do Spider, que é menor que o conjunto de treino completo. Isso pode ter reduzido o risco de "esquecimento catastrófico", mesmo após 5 épocas, preservando a capacidade geral do modelo.

Os resultados indicam que é possível especializar LLMs para tarefas específicas sem comprometer a generalização, pelo menos em cenários de fine-tuning moderado e com datasets restritos. Isso tem implicações práticas importantes. Organizações podem adaptar modelos open-source para domínios específicos sem perder versatilidade em tarefas gerais. Técnicas como LoRA tornam esse processo viável mesmo com recursos computacionais limitados. Isso permite a personalização de LLMs em ambientes de produção. Tudo isso, claro, considerando que eu não fiz bobagem nos experimentos.