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

*PPGINF528 - Tópicos Especiais em Recuperação de Informação - 2025/01

Erick R. Ribeiro

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introdução

O avanço dos Modelos de Linguagem de Grande Porte (LLMs) tem impulsionado significativamente o desenvolvimento de sistemas capazes de resolver tarefas complexas em linguagem natural. Entre essas tarefas, destaca-se o *Text-to-SQL*, que consiste em traduzir consultas em linguagem natural para instruções SQL válidas e executáveis sobre bancos de dados relacionais. Trata-se de um problema desafiador, pois requer não apenas compreensão semântica da linguagem, mas também a capacidade de generalizar para esquemas de banco de dados previamente não vistos. Para avaliar essa tarefa, é amplamente utilizado o benchmark Spider, que simula cenários realistas com consultas complexas e múltiplos domínios de dados.

Uma das estratégias recorrentes para melhorar o desempenho de LLMs em tarefas específicas é o fine-tuning supervisionado, que consiste em ajustar os parâmetros do modelo com dados anotados de uma tarefa-alvo. No entanto, embora o fine-tuning promova ganhos expressivos na tarefa de interesse, ele pode induzir uma degradação do desempenho em tarefas fora desse domínio, fenômeno conhecido como "esquecimento catastrófico" ou "regressão de capacidade". Esse efeito levanta uma questão central no desenvolvimento e na aplicação de LLMs: até que ponto é possível especializar um modelo sem comprometer sua capacidade de generalização?

Este trabalho propõe uma avaliação empírica e sistemática desse trade-off entre especialização e generalização, tomando como estudo de caso o modelo base *mistralai/Mistral-7B-Instruct-v0.2*, aplicado à tarefa de *Text-to-SQL*. Para isso, além do benchmark Spider, também utilizamos o benchmark MMLU (*Massive Multitask Language Understanding*), que avalia o desempenho dos LLMs em mais de 50 domínios diversos, incluindo matemática, história, biologia, direito, entre outros. Assim, culminando em um processo robusto para mensurar a manutenção da capacidade de raciocínio geral após o processo de *fine-tuning*.

Identify applicable funding agency here. If none, delete this.

Nosso objetivo é quantificar de forma precisa os impactos do *fine-tuning* no modelo, respondendo às seguintes questões:

(i) qual o ganho efetivo de desempenho na tarefa de *Text-to-SQL*;
(ii) qual o custo desse ganho em termos de perda de generalização em tarefas de conhecimento amplo.

II. Metodologia

Esta seção descreve, de forma detalhada, o pipeline de dados, a configuração adotada no método *LoRA*, bem como a arquitetura de software utilizada na implementação da métrica *Execution Accuracy*, a qual foi empregada na avaliação do treinamento do LLM para a tarefa-alvo.

A. Pipeline de Dados

Todo o código-fonte foi implementado de forma reproduzível e está disponível no repositório público do GitHub[1]. O repositório conta com um pipeline de dados que garante não apenas a integridade dos experimentos, mas também a correta separação entre os dados utilizados no treinamento e na avaliação, tanto na tarefa-alvo quanto na tarefa de generalização (MMLU). Neste repositório, existe uma pasta chamada `./scripts/`, com toda a implementação do pré-processamento, preparação dos prompts, configuração dos experimentos e execução dos treinamentos. A pasta chamada `./custom_metrics/` com a implementação da métrica de ExecutionAccuracy, bem como os casos de teste desenvolvidos para avaliar o modelo base e os treinados. Já os modelos treinados, bem como os adaptadores LoRA gerados, estão organizados na pasta `./models/` dentro do mesmo repositório, permitindo total transparência e reprodutibilidade dos resultados apresentados neste trabalho.

B. Preparação dos Dados para Fine-Tuning

Para o processo de fine-tuning, utilizou-se o conjunto de dados Spider[2], disponibilizado no formato de dataset da biblioteca do HuggingFace. Este conjunto de dados já consta com uma divisão predefinida de treino e teste, o que garante a consistência na avaliação dos modelos.

Cada exemplo passado ao modelo é composto por uma tripla estruturada: *schema* do banco de dados, pergunta em linguagem natural e a respectiva consulta SQL. Para adequar esses dados ao modelo *Mistral-7B-Instruct-v0.2* utilizado, foi

[1]https://github.com/erickrribeiro/ufam_nlp_01_2025.git
[2]https://huggingface.co/datasets/xlangai/spider

realizada uma conversão para o formato conversacional, conforme o template abaixo:

```
[{
  "role": "system",
  "content": "You are a helpful assistant who
      answers questions about database tables
      by responding with SQL queries."
}
{
  "role": "user",
  "content": "Schema:{0}\nQuestion: {1}\nSQL:"
},
{
  "role": "assistant",
  "content": "SQL: {2}"
},
// 4 exemplos de few-shot
{
  "role": "user",
  "content": "Schema:{0}\nQuestion: {1} SQL:"
}]
```

A inclusão do schema no prompt tem como objetivo garantir que o modelo base, neste caso o *Mistral-7B-Instruct-v0.2*, tenha pleno conhecimento da estrutura do banco de dados sobre o qual deverá ser gerada a consulta SQL. Estratégia que potencializa a capacidade de generalização do modelo, aproximando o fine-tuning de cenários práticos reais em que a arquitetura do banco de dados é um fator determinante para a geração correta da consulta.

C. Configuração do Fine-Tuning

O processo de fine-tuning foi realizado utilizando a técnica LoRA (*Low-Rank Adaptation*), que permite reduzir significativamente o número de parâmetros ajustáveis, mantendo o desempenho competitivo do modelo. Especificamente, neste artigo foram treináveis apenas $1,7$ milhões de parâmetros[3], representando apenas $0,0235\%$ do total de $7,24$ bilhões de parâmetros do modelo base.

TABLE I
HIPERPARÂMETROS UTILIZADOS NO LORA.

Parâmetro	Valor
Rank (r)	4
Alpha (α)	16
Dropout	0,05
Target Modules	`["q_proj", "v_proj"]`
Batch Size	4

A configuração do LoRA permaneceu constante nos experimentos, conforme apresentado na Tabela I. Durante o treinamento, o *batch size* foi limitado a 4, restrição imposta pela capacidade da GPU NVIDIA A100 (40GB) utilizada.

Durante os experimentos, inicialmente tentou-se utilizar precisão mista com *bfloat16*. Entretanto, o consumo de memória excedeu a capacidade disponível. Como solução, adotou-se o uso de *float16*, que apresentou melhor equilíbrio entre consumo de memória e estabilidade numérica. Além disso, foi necessário substituir o otimizador padrão por *adamw_torch_fused*, uma implementação otimizada que reduz o uso de memória e acelera o treinamento. A configuração dos hiperparâmetros foi projetada considerando a limitação de memória e velocidade de treinamento.

[3]Confirmar o valor em: https://github.com/erickrribeiro/ufam_nlp_01_2025/scripts/fase-02.ipynb

D. Experimentação de Hiperparâmetros

A principal variável controlada entre os experimentos foi a learning rate, com o objetivo de avaliar sua influência na estabilidade e no desempenho do fine-tuning.

- Experimento 1: uso de um learning rate mais conservadora (2e-4), visando promover maior estabilidade durante o treinamento, especialmente relevante em cenários com poucos dados e fine-tuning parcial via LoRA.
- Experimento 2: uso de um learning rate mais agressiva (5e-4), com a intenção de investigar se o aumento desse hiperparâmetro poderia acelerar a convergência ou, eventualmente, introduzir problemas como overfitting ou instabilidade na otimização.

No primeiro experimento, adotou-se uma learning rate mais conservadora, visando maior estabilidade no ajuste dos gradientes. Já no segundo experimento, optou-se por uma learning rate mais agressiva, com o objetivo de avaliar se o aumento da taxa de aprendizado poderia gerar instabilidade, oscilação nas métricas ou indícios de overfitting. Essa variação nos hiperparâmetros objetiva explorar o impacto direto do ajuste da learning rate no contexto de geração de consultas SQL, além de fornecer insights sobre boas práticas no fine-tuning de LLMs aplicados a tarefa alvo.

E. Métrica Execution Accuracy

Para avaliar a qualidade do fine-tining na tarefa alvo foi implementada a métrica *Execution Accuracy*. Esta métrica considera não apenas a equivalência textual, mas os resultados produzidos pela execução das consultas sobre um banco de dados. A arquitetura da métrica é composta por duas etapas principais:

1) **Execução no Banco de Dados:** tanto a consulta gerada (`actual_output`) quanto a consulta de referência (`expected_output`) são executadas sobre uma instância do banco `SQLite`, construída dinamicamente a partir do schema correspondente ao exemplo no dataset *Spider*. Esse procedimento permite que a comparação seja feita no nível dos resultados produzidos, em vez de se restringir à comparação textual.
2) **Comparação dos Resultados:** os resultados das consultas são convertidos em conjuntos de tuplas, ignorando a ordenação tanto de linhas quanto de colunas.

A métrica *Execution Accuracy* é definida pela formula abaixo, onde R_{gen} representa o conjunto de resultados da consulta gerada (`actual_output`), e R_{ref} representa o conjunto de resultados da consulta de referência (`expected_output`). Se os conjuntos forem idênticos, a métrica retorna 1.0, indicando execução correta. Caso

contrário, ou em caso de falha na execução, o valor retornado é 0.0.

$$\text{ExecutionAccuracy} = \begin{cases} 1, & \text{se } R_{gen} = R_{ref} \\ 0, & \text{caso contrário} \end{cases} \quad (1)$$

A implementação completa da métrica está disponível no repositório público deste trabalho, na pasta `custom_metrics`. A avaliação dos modelos foi conduzida utilizando o framework *DeepEval*, que permite a criação e execução de testes automatizados para modelos de linguagem. Onde, a execução dos testes pode ser realizada diretamente via linha de comando, conforme os comandos abaixo:

```
!deepeval test run test_baseline.py
!deepeval test run test_mistral_sql_lora_v3.py
!deepeval test run test_mistral_sql_lora_v4.py
```

Cada script corresponde a um experimento específico, contendo as configurações, exemplos e prompts utilizados na avaliação.

III. Resultados

Esta seção apresenta os resultados obtidos na tarefa de Text-to-SQL, avaliada no benchmark Spider, e na avaliação de generalização por meio do benchmark MMLU. Discutem-se os impactos do fine-tuning utilizando LoRA, bem como as limitações observadas.

A. Desempenho na Tarefa de Text-to-SQL

Os resultados no benchmark Spider indicam ganhos limitados após o fine-tuning. O modelo base, avaliado em configuração few-shot com $k = 4$, obteve 42,94% de Execution Accuracy. A aplicação de LoRA com $lr = 2e^{-4}$ (LoRA V1) resultou em um desempenho de 43,13%, representando um ganho marginal. No entanto, ao aumentar a taxa de aprendizado para $5e^{-4}$ (LoRA V2), o desempenho caiu para 37,72%, sugerindo que configurações mais agressivas podem comprometer a estabilidade do treinamento.

TABLE II
Resultados no benchmark Spider (tarefa Text-to-SQL).

Modelo	Execution Accuracy (%)
Base	42,94
LoRA V1 ($lr = 2e^{-4}$)	43,13
LoRA V2 ($lr = 5e^{-4}$)	37,72

B. Análise de Erros na Tarefa de Text-to-SQL

Para complementar a avaliação quantitativa, realizou-se uma análise qualitativa dos erros cometidos pelo modelo fine-tuned (V1) na tarefa-alvo. Foram selecionados três exemplos representativos de falhas recorrentes.

O primeiro exemplo corresponde à consulta *"Show the property type descriptions of properties belonging to that code."*. O modelo gerou a seguinte query incorreta:

```
SELECT property_type_description
FROM Ref_Property_Types WHERE
property_type_code = ?
```

A consulta correta deveria ser uma junção entre tabelas, agrupando pelos códigos de propriedade:

```
SELECT T2.property_type_description
FROM Properties AS T1 JOIN
Ref_Property_Types AS T2
ON T1.property_type_code =
T2.property_type_code GROUP BY
T1.property_type_code
```

Esse erro indica uma limitação do modelo em identificar corretamente quando a informação requerida não está diretamente na tabela de referência, mas necessita de uma junção com a tabela de instâncias.

De forma similar, na consulta *"What is the feature type name of feature AirCon?"*, o modelo ignorou a necessidade da tabela intermediária que relaciona os recursos disponíveis:

```
SELECT feature_type_name
FROM Ref_Feature_Types WHERE
feature_type_code = 'AirCon'
```

Quando o correto seria:

```
SELECT T2.feature_type_name
FROM Other_Available_Features
AS T1 JOIN Ref_Feature_Types
AS T2 ON T1.feature_type_code
= T2.feature_type_code WHERE
T1.feature_name = "AirCon"
```

Esse padrão de erro sugere que o modelo tende a assumir relações diretas nas tabelas de referência, sem inferir a necessidade de joins, especialmente quando os atributos no enunciado coincidem com nomes de colunas.

Por fim, um erro recorrente aparece na consulta *"List the name of singers that do not have any song."*, onde o modelo gerou:

```
SELECT Name FROM singer WHERE
Song_ID IS NULL
```

O correto seria utilizar uma subconsulta com `NOT IN`:

```
SELECT Name FROM singer WHERE
Singer_ID NOT IN (SELECT Singer_ID
FROM song)
```

Esse erro ilustra uma dificuldade em generalizar padrões de ausência de relações, frequentemente modelados como subconsultas ou cláusulas `NOT EXISTS`.

Apesar dessas limitações, observou-se que o modelo foi bem-sucedido em consultas que exigem junções simples e agregações diretas. Por exemplo, na consulta *"For each singer name, what is the total sales for their songs?"*, o modelo gerou uma query correta tanto semanticamente quanto funcionalmente, embora com pequenas diferenças na forma:

```
SELECT singer.Name, SUM(song.Sales)
AS total_sales FROM singer,
song WHERE singer.Singer_ID =
song.Singer_ID GROUP BY singer.Name
```

O resultado equivale funcionalmente à consulta esperada, mostrando que o modelo aprendeu corretamente padrões básicos de junção e agregação.

Essa análise evidencia que as principais fragilidades estão relacionadas à inferência de relações indiretas entre tabelas e à modelagem de consultas com estruturas mais complexas, como subconsultas negativas e cláusulas condicionais mais sofisticadas.

C. Impacto na Generalização — Benchmark MMLU

Apesar dos esforços no ajuste fino para a tarefa de Text-to-SQL, observou-se uma degradação generalizada no desempenho do modelo em tarefas avaliadas pelo benchmark MMLU. Esta degradação não foi uniforme, variando de acordo com o domínio da tarefa.

A Tabela III apresenta a variação de desempenho por categoria. Nota-se que *Ciências Sociais* foi a categoria menos impactada, chegando inclusive a apresentar uma leve melhora no modelo LoRA V1. Por outro lado, as categorias *Humanidades* e *STEM* sofreram perdas consideráveis de acurácia, evidenciando que tarefas associadas a raciocínio lógico, matemático e formal são mais sensíveis ao sobreajuste causado pelo fine-tuning direcionado.

TABLE III
VARIAÇÃO DE ACURÁCIA POR CATEGORIA NO BENCHMARK MMLU.

Categoria	**Baseline**	**V1**	**V2**	**Var.** V1	**Var.** V2
Overall	27,70	18,56	18,53	-24,79	-10,95
Ciências Sociais	26,86	23,50	20,43	+6,92	-2,53
Humanidades	34,15	17,17	15,15	-44,63	-35,33
STEM	25,43	15,60	18,65	-38,87	-6,20

Uma análise detalhada por tarefa (Tabela IV) reforça que as maiores quedas de desempenho ocorreram em disciplinas associadas à matemática, lógica formal e física. Por exemplo, a tarefa *Abstract Algebra* apresentou uma redução de mais de 90% na acurácia, enquanto *High School Mathematics* apresentou um comportamento atípico, com uma melhora expressiva no modelo LoRA V2, possivelmente associada à coincidência de padrões da tarefa com os ajustes realizados no fine-tuning para Text-to-SQL.

Por outro lado, tarefas como *World Religions* e *Marketing* mostraram estabilidade ou até melhorias modestas, sugerindo que conteúdos mais factuais ou baseados em memorização foram menos impactados.

TABLE IV
EXEMPLOS DE VARIAÇÃO DE ACURÁCIA POR TAREFA NO BENCHMARK MMLU.

Tarefa	**Categoria**	**Baseline**	**V2**	**Var.** V2
Abstract Algebra	STEM	11,00	1,00	-90,91
High School Math...	STEM	6,67	20,37	+205,56
College Chemistry	STEM	22,00	21,00	-4,55
Formal Logic	Humanidades	5,56	9,52	+71,43
World Religions	Humanidades	20,47	26,90	+31,43
Marketing	Ciências Sociais	22,22	29,06	+30,77

D. Análise dos Resultados

O fine-tuning específico para Text-to-SQL mostrou-se eficaz para a tarefa, mas comprometeu a generalização do modelo em outras áreas, especialmente em tarefas que envolvem raciocínio lógico e formal. Esse comportamento é característico do *Catastrophic Forgetting*, comum em adaptações específicas sem estratégias de preservação do conhecimento prévio.

A escolha dos hiperparâmetros de LoRA, principalmente a taxa de aprendizado, influenciou diretamente tanto o desempenho na tarefa-alvo quanto a perda de capacidade geral. Configurações mais conservadoras causaram menor degradação, enquanto ajustes agressivos ampliaram os ganhos no Text-to-SQL ao custo de maior perda na generalização.

E. Análise dos Resultados

Os resultados observados indicam que o fine-tuning específico para a tarefa de Text-to-SQL, embora promissor para a tarefa-alvo, compromete a capacidade de generalização do modelo em uma ampla gama de domínios, especialmente em tarefas que demandam raciocínio lógico e formal.

Este fenômeno é consistente com o efeito conhecido como *Catastrophic Forgetting*, frequentemente observado quando modelos são adaptados para tarefas específicas sem o uso de técnicas que preservem o conhecimento prévio.

Além disso, observa-se que a escolha dos hiperparâmetros de LoRA, especialmente a taxa de aprendizado, impacta não apenas o desempenho na tarefa-alvo, mas também a severidade da perda de generalização. Configurações mais conservadoras (como LoRA V1) resultaram em menor degradação, enquanto ajustes mais agressivos (LoRA V2) amplificaram tanto os efeitos positivos na tarefa de Text-to-SQL quanto os negativos nas tarefas de múltipla escolha do MMLU.

IV. DISCUSSÃO

Os resultados indicam um trade-off claro entre especialização e generalização em modelos de linguagem submetidos a fine-tuning para Text-to-SQL.

A magnitude do ganho na tarefa de Text-to-SQL justifica a perda de capacidade geral? Os ganhos observados na tarefa de Text-to-SQL são modestos e se concentram em cenários mais simples. Entretanto, a perda de desempenho em tarefas gerais, especialmente em domínios que demandam raciocínio lógico e habilidades matemáticas, é significativa. Portanto, para aplicações que requerem versatilidade, a degradação pode não justificar a especialização. Em contrapartida, se o modelo for utilizado exclusivamente para Text-to-SQL, a especialização pode ser vantajosa.

Quais fatores influenciam mais este trade-off? Os principais fatores são:

* *Taxa de aprendizado*: taxas mais altas aumentam tanto o ganho na tarefa-alvo quanto a perda de generalização, indicando que ajustes agressivos amplificam o esquecimento catastrófico.

Quais as implicações práticas para o desenvolvimento de LLMs comerciais especializados? Esses achados sugerem que o fine-tuning especializado deve ser aplicado com cautela, levando em conta o perfil de uso do modelo. Para produtos comerciais que exigem múltiplas habilidades, é crucial integrar técnicas que preservem a generalização, como aprendizado multi-tarefa, aprendizagem contínua e regularizações

específicas em métodos PEFT. Além disso, arquiteturas que permitam modularidade, como adapter fusion, podem oferecer um caminho para equilibrar especialização e flexibilidade, possibilitando modelos robustos e adaptáveis a diferentes contextos.

V. Considerações Finais

Embora o fine-tuning com LoRA seja eficiente e melhore o desempenho em Text-to-SQL, ele acarreta custos na generalização do modelo. Sua aplicação deve ser cuidadosamente avaliada conforme as necessidades e o contexto do sistema final.