TP4 Análise Quantitativa

1st Igor Lima
Instituto de Computação
Universidade Federal do Amazonas
Manaus, Amazonas
igor.lima@icomp.ufam.edu.br

***Abstract*—A especialização de Modelos de Linguagem de Grande Escala (LLMs) para tarefas de nicho, como a tradução de texto para SQL (Text-to-SQL), apresenta um trade-off fundamental entre o ganho de proficiência na tarefa-alvo e a potencial degradação de suas capacidades de conhecimento geral. Este trabalho investiga essa dinâmica através do fine-tuning do modelo unsloth/Llama-3.2-3B-Instruct-bnb-4bit no complexo dataset cross-domain Spider, utilizando a técnica de Low-Rank Adaptation (LoRA). A performance foi quantificada por meio de uma metodologia de avaliação dupla: (1) uma métrica customizada de Acurácia de Execução (Execution Accuracy) em um framework de testes automatizado com pytest e deepeval para a tarefa de Text-to-SQL, e (2) uma avaliação de regressão de capacidade no benchmark MMLU, abrangendo os domínios de STEM, Humanidades e Ciências Sociais. Os resultados demonstram um ganho de performance substancial na tarefa de especialização, com a acurácia de execução saltando de um baseline de 3.48% para 7.25% no melhor modelo fine-tuned. Em contrapartida, foi observada uma modesta regressão de capacidade agregada de 2.00 pontos percentuais no MMLU. Notavelmente, a análise por categoria revelou que a perda de conhecimento foi concentrada em Humanidades (-8.00 pts), enquanto houve um ganho inesperado em domínios de STEM (+2.00 pts), sugerindo uma sinergia positiva entre tarefas lógicas. Conclui-se que o fine-tuning com LoRA é uma estratégia eficaz, onde o ganho expressivo na tarefa-alvo justifica a regressão de capacidade controlada, ressaltando a importância de pipelines de avaliação holística para o desenvolvimento de LLMs comerciais especializados.**

***Index Terms*—Isso é um Relatório de Faculdade.**

I. METODOLOGIA

Esta seção detalha o pipeline experimental empregado para o fine-tuning e a avaliação de modelos de linguagem na tarefa de tradução de texto para SQL (Text-to-SQL), bem como na análise subsequente de regressão de capacidade. O processo foi dividido em quatro fases principais: estabelecimento de um baseline de desempenho, especialização dos modelos através de fine-tuning, avaliação na tarefa-alvo com uma métrica customizada, e, por fim, avaliação de conhecimento geral com o benchmark MMLU.

A. Avaliação do Modelo Base (Baseline)

Esta seção descreve o processo experimental para o estabelecimento de um baseline de desempenho para um modelo de linguagem na tarefa de tradução de texto para SQL (Text-to-SQL). O objetivo foi quantificar a capacidade "out-of-the-box" de um modelo pré-treinado, sem qualquer fine-tuning, para servir como uma referência de comparação para especializações futuras.

O modelo de base selecionado para a avaliação foi o unsloth/Llama-3.2-3B-Instruct-bnb-4bit, uma variante do Llama 3.2 otimizada para inferência eficiente com quantização de 4-bits. A avaliação foi conduzida no ambiente Google Colab, utilizando uma GPU Tesla T4. O framework de software para esta fase, documentado no notebook *TP4-Baseline-execution.ipynb*, foi construído em Python, utilizando a biblioteca unsloth para o carregamento do modelo e transformers para a geração de texto.

O conjunto de dados empregado para a avaliação foi o split de desenvolvimento (dev.json) do benchmark Spider. Este conjunto contém 1.034 exemplos, cada um consistindo em uma pergunta em linguagem natural, uma consulta SQL correspondente (ground-truth) e o identificador do banco de dados (db-id) ao qual a pergunta se refere. Para cada um desses exemplos, foi adotada uma metodologia de few-shot prompting. Um template de prompt foi construído contendo três exemplos fixos de pares pergunta-SQL, projetados para contextualizar o modelo na tarefa. A pergunta de teste era então inserida neste template, e o prompt completo era submetido ao modelo para a geração da consulta SQL.

A avaliação da acurácia foi totalmente automatizada. Para cada consulta gerada pelo modelo, o script estabelecia uma conexão com o banco de dados SQLite correspondente, localizado no diretório database/. Tanto a consulta gerada quanto a consulta de referência foram executadas, e seus resultados foram comparados. A métrica de sucesso, denominada Acurácia de Execução (Execution Accuracy), foi rigorosa: um caso de teste era considerado um "sucesso" somente se o conjunto de resultados da consulta gerada fosse idêntico ao da consulta de referência. Qualquer divergência nos resultados ou erro de sintaxe durante a execução da consulta gerada era classificada como "falha". Ao final do processo, a taxa de acurácia de execução do modelo base foi calculada como a razão entre o número de sucessos e o número total de exemplos avaliados.

B. Especialização de Modelos com Fine-Tuning

Com o objetivo de especializar um modelo de linguagem na tarefa de Text-to-SQL, foi realizado um processo de fine-

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

tuning utilizando o train-spider.json como fonte de dados. O notebook TP4-finetunes.ipynb documenta este procedimento, que foi conduzido em um ambiente Google Colab com uma GPU Tesla T4.

1) Pipeline de Dados e Pré-processamento: O processo iniciou-se com a preparação dos dados de treinamento. Um total de 7.000 pares de pergunta-consulta foram extraídos do arquivo train-spider.json. Estes pares foram então processados por um script Python para se adequarem ao formato de conversação ShareGPT, sendo salvos em um arquivo qa-sharegpt.jsonl. Este formato estrutura cada exemplo como um diálogo contendo os papéis user (pergunta em linguagem natural) e assistant (consulta SQL correspondente). O dataset resultante foi então carregado e tokenizado, aplicando-se um chat-template específico do Llama 3. Uma etapa crucial deste pré-processamento foi a aplicação de uma máscara de loss, utilizando a função train-on-responses-only da biblioteca Unsloth. Esta técnica garante que o cálculo do gradiente durante o treinamento considere apenas a resposta do "assistant", forçando o modelo a aprender a gerar a consulta SQL de forma autônoma, sem tentar prever ou replicar a entrada do usuário.

2) Configuração e Treinamento com LoRA: O modelo base selecionado para a especialização foi o unsloth/Llama-3.2-3B-Instruct-bnb-4bit, uma variante otimizada para treinamento eficiente. O fine-tuning foi implementado com a técnica de Low-Rank Adaptation (LoRA), que insere matrizes de baixo ranque treináveis nos módulos de atenção do modelo, especificamente em q-proj, k-proj, v-proj, o-proj, gate-proj, up-proj, e down-proj. Esta abordagem de Parameter-Efficient Fine-Tuning (PEFT) permitiu a atualização de apenas 0.81% dos parâmetros totais do modelo.

Foram conduzidos dois experimentos de fine-tuning distintos, variando-se os hiperparâmetros para avaliar diferentes estratégias de otimização. O Modelo 1 foi treinado por uma única época com uma taxa de aprendizado mais alta, enquanto o Modelo 2 foi treinado por três épocas com uma taxa de aprendizado significativamente mais baixa e uma semente de aleatoriedade diferente. O otimizador adamw-8bit foi utilizado em ambos os casos para um uso de memória mais eficiente. Os adaptadores LoRA resultantes de cada treinamento foram salvos localmente nas pastas lora-model-version1 e lora-model-version2. A Tabela 1 detalha os hiperparâmetros específicos para cada um dos modelos.

TABLE I
HIPERPARÂMETROS DE FINE-TUNING PARA OS MODELOS

Hiperparâmetro	**Valor (Modelo 1)**	**Valor (Modelo 2)**
basemodel	unsloth/Llama-3.2-3B	unsloth/Llama-3.2-3B
r (Rank LoRA)	16	16
loraalpha	16	16
targetmodules	qproj, kproj, vproj, oproj, gateproj, upproj, downproj	qproj, kproj, vproj, oproj, gateproj, upproj, downproj
loradropout	0	0
perdevicetrainbatchsize	2	4
gradientaccumulationsteps	4	4
numtrainepochs	1	3
learningrate	2e-4	2e-8
optim	adamw8bit	adamw8bit
seed	3407	8000
Tempo de Treinamento	28.43 minutos	63.30 minutos

C. Arquitetura de Avaliação Automatizada

A avaliação dos modelos foi estruturada como uma suíte de testes automatizada, utilizando o pytest como orquestrador. Conforme definido no script test-evaluation.py, a avaliação é parametrizada para iterar exaustivamente sobre duas dimensões: todos os 1.034 exemplos do split de desenvolvimento (dev.json) do dataset Spider e cada um dos modelos a serem avaliados (neste caso, "Modelo 1" e "Modelo 2"). Esta abordagem garante que cada pergunta seja testada contra cada variante do modelo, criando uma matriz de teste completa de 2.068 execuções.

O carregamento dos modelos fine-tuned é realizado no início do processo. A metodologia implementada na função load-lora-model consiste em carregar primeiramente o modelo base pré-treinado, unsloth/Llama-3.2-3B-Instruct-bnb-4bit, e em seguida aplicar os pesos do adaptador LoRA salvo em disco (por exemplo, de lora-model-version1) utilizando a classe PeftModel da biblioteca peft. Este processo reconstrói o modelo especializado para a inferência. Para garantir o isolamento, este procedimento é repetido para cada um dos modelos fine-tuned, assegurando que não haja contaminação de pesos entre os testes.

A geração da consulta SQL para cada pergunta é encapsulada na função gerar-sql. Esta função formata a entrada utilizando um template de few-shot (prompt-template.txt), aplica o chat-template apropriado para o Llama 3, e move os tensores de entrada para a GPU (cuda). A inferência é realizada com o método model.generate. Uma etapa de pós-processamento é aplicada à saída bruta do modelo para extrair e limpar a consulta SQL, isolando a resposta do "assistant", removendo quaisquer marcações de código (como "'sql) e eliminando pontos-e-vírgulas terminais para garantir a compatibilidade com o sqlite3.

D. Métrica Customizada: ExecutionAccuracyMetric

Para avaliar a correção funcional das consultas geradas, foi desenvolvida uma métrica customizada denominada ExecutionAccuracyMetric, conforme o script execution-metric.py. A classe herda de deepeval.metrics.BaseMetric e foi explicitamente configurada para operar em modo síncrono (self.async-mode = False), garantindo a compatibilidade com a biblioteca padrão sqlite3 para operações de banco de dados.

O núcleo da avaliação reside no método measure, que implementa um protocolo rigoroso para cada LLMTestCase fornecido pelo pytest. A lógica segue os seguintes passos:

a. Conexão com o Banco de Dados: O método extrai o identificador do banco de dados (db-id) do campo context do LLMTestCase. Este db-id é então utilizado para construir dinamicamente o caminho para o arquivo de banco de dados

SQLite correspondente dentro da estrutura de diretórios do dataset Spider.

b. Execução da Consulta Gerada: A consulta SQL proveniente do modelo (test-case.actual-output) é executada no banco de dados. Toda a operação de banco de dados é encapsulada em um bloco try-except para capturar qualquer sqlite3.Error, como erros de sintaxe na SQL gerada. Uma falha na execução resulta imediatamente em um score de 0.0.

c. Execução da Consulta de Referência: Se a consulta do modelo for executada com sucesso, a consulta ground-truth (test-case.expected-output) é então executada no mesmo banco de dados.

d. Comparação de Resultados: Os resultados de ambas as consultas são obtidos através do método fetchall(). Para garantir uma comparação que seja insensível à ordem das linhas retornadas, as listas de tuplas de ambos os resultados são convertidas para o tipo set.

e. Atribuição de Score: A métrica atribui um score binário: 1.0 se os dois conjuntos de resultados forem idênticos, e 0.0 em qualquer outro caso (resultados diferentes ou falha na execução da consulta gerada). Adicionalmente, um atributo reason é populado com uma mensagem descritiva do resultado, facilitando a análise de erros posterior.

Finalmente, a função de teste test-spider-execution utiliza um assert para verificar se a métrica foi bem-sucedida, registrando formalmente o sucesso ou a falha do caso de teste no relatório final do pytest.

E. Metodologia de Avaliação de Regressão de Capacidade

Para quantificar o impacto do fine-tuning no conhecimento geral e nas capacidades de raciocínio dos modelos, foi conduzida uma avaliação utilizando o benchmark MMLU (Massive Multitask Language Understanding). Conforme implementado no script mmlu-avaliator.py, este processo foi projetado para medir a performance tanto do modelo base quanto dos modelos especializados, permitindo uma análise de regressão de capacidade precisa e categorizada.

O pipeline de avaliação selecionou três subconjuntos do MMLU para representar domínios distintos do conhecimento: economy.parquet para Ciências Sociais, philosophia.parquet (sic) para Humanidades e computer.parquet para STEM. De cada um destes arquivos, foram extraídos os primeiros 50 exemplos para compor o conjunto de teste, garantindo um número balanceado de questões por categoria. A função load-mmlu-data foi responsável por ler os arquivos no formato Parquet, processar as questões de múltipla escolha e estruturar os dados para a avaliação.

A metodologia de inferência seguiu o padrão 4-shot, onde um prompt fixo contendo quatro exemplos de pergunta-resposta foi prependido a cada questão de teste para contextualizar o modelo. Uma técnica de avaliação baseada em log-likelihood, implementada na função get-log-likelihood, foi empregada para determinar a resposta do modelo. Em vez de uma geração de texto aberta, este método consiste em criar quatro versões completas do prompt para cada questão, cada uma terminando com uma das possíveis opções (A, B, C ou D). O script então calcula a log-likelihood que o modelo atribui a cada uma dessas quatro sequências. A opção que resulta na maior pontuação de log-likelihood (ou seja, a mais provável segundo o modelo) é considerada a resposta escolhida.

```
--- AVALIAÇÃO DO BASELINE CONCLUÍDA ---
Total de Perguntas Avaliadas: 1034
Sucessos: 36
Falhas: 998
Taxa de Sucesso (Execution Accuracy): 3.48%
```

Fig. 1. Enter Caption

O processo de avaliação, orquestrado pela função evaluate-model-on-mmlu, foi executado de forma isolada para cada modelo (o base e as duas variantes fine-tuned). Para evitar o esgotamento de recursos da GPU, o script foi estruturado para carregar um modelo, executar a avaliação completa em todas as três categorias do MMLU, e então descarregar o modelo da memória (del model e torch.cuda.empty-cache()) antes de prosseguir para o próximo. A acurácia final para cada modelo, tanto de forma agregada quanto por categoria, foi calculada e apresentada pela função analyze-and-report-results, que também computou a variação percentual de desempenho em relação ao baseline para quantificar a regressão de capacidade.

II. RESULTADOS

A. Resultados Baseline

Esta seção apresenta os resultados quantitativos e uma análise qualitativa do desempenho do modelo de linguagem unsloth/Llama-3.2-3B-Instruct-bnb-4bit em sua forma base, sem fine-tuning, na tarefa de Text-to-SQL. A avaliação foi conduzida no conjunto de desenvolvimento do dataset Spider, composto por 1.034 exemplos, utilizando a métrica de Acurácia de Execução (Execution Accuracy).

1) Resultados Quantitativos do Baseline: A avaliação do modelo base em todos os 1.034 exemplos do split de desenvolvimento do Spider revelou um desempenho significativamente baixo, que serve como uma linha de referência crucial para este trabalho. Conforme sumarizado na Figura 1, o modelo alcançou uma taxa de sucesso de apenas 3.48%. De um total de 1.034 consultas SQL geradas, somente 36 foram executadas com sucesso e retornaram um resultado idêntico ao da consulta de referência (ground-truth). As 998 falhas restantes foram majoritariamente causadas por erros de sintaxe ou de lógica nas consultas geradas.

A análise dos logs de execução permitiu a identificação de padrões de erro recorrentes que justificam a baixa taxa de acurácia. Estes erros podem ser categorizados em três tipos principais:

- Alucinação de Esquema (Schema Hallucination): O erro mais comum foi a incapacidade do modelo de aderir ao esquema exato do banco de dados em questão. Em

múltiplos casos, o modelo gerou consultas sintaticamente plausíveis, mas que referenciavam tabelas ou colunas inexistentes. Por exemplo, para a pergunta "How many poker players are there?" no banco de dados poker-player, o modelo gerou SELECT COUNT(*) FROM player WHERE game = 'poker', tentando acessar uma tabela genérica player em vez da tabela correta poker-player. Este padrão se repetiu consistentemente, como ao tentar acessar a coluna title na tabela tv-series (item 617), que na verdade se chama Episode.

- Geração de Respostas Conversacionais: Em vários casos, em vez de gerar uma consulta SQL, o modelo respondeu à pergunta de forma conversacional. Para a pergunta "When did the episode 'A Love of a Lifetime' air?" (item 621), o modelo retornou um parágrafo explicativo: "To translate the natural language question...". O framework de avaliação tentou executar este texto como SQL, resultando em um erro de sintaxe imediato (near "To": syntax error). Isso demonstra que, sem um fine-tuning rigoroso, o modelo pode não distinguir claramente entre uma instrução para traduzir e uma instrução para responder.
- Lógica de Consulta Incorreta: Em raras ocasiões onde o modelo acertou os nomes das tabelas e colunas, ele ainda falhou na lógica da consulta. No item 807, para a pergunta "how many countries are in Asia?", o modelo gerou SELECT COUNT(*) FROM country WHERE region = 'Asia'. A consulta correta seria ... WHERE continent = 'Asia'. Embora a consulta gerada seja executável, ela retorna um resultado incorreto, pois o modelo confundiu os conceitos de "região" e "continente", levando a uma falha na métrica de comparação de resultados.

Esta análise qualitativa reforça a conclusão quantitativa de que modelos de linguagem pré-treinados, mesmo os mais avançados, requerem uma especialização profunda para se tornarem ferramentas confiáveis para a tarefa de Text-to-SQL em domínios específicos.

B. Resultados Pytest

Esta seção apresenta os resultados quantitativos e uma análise qualitativa do desempenho dos modelos de linguagem fine-tuned na tarefa de Text-to-SQL. A avaliação foi conduzida no conjunto de desenvolvimento do dataset Spider, composto por 1.034 exemplos, e utilizou a métrica de Acurácia de Execução (ExecutionAccuracyMetric) detalhada na seção de Metodologia.

1) Resultados Quantitativos: O processo de fine-tuning demonstrou um ganho de desempenho substancial em relação ao modelo base. De um total de 2.068 testes executados (1.034 para cada um dos dois modelos), foram registrados 137 sucessos e 1.931 falhas. A análise do relatório de avaliação (relatorio-de-avaliacao.html) permitiu a desagregação desses resultados por modelo.

O "Modelo 1" (treinado por 1 época com maior taxa de aprendizado) alcançou 75 execuções corretas, resultando em uma acurácia de 7.25%. O "Modelo 2" (treinado por 3 épocas com menor taxa de aprendizado) obteve 62 sucessos, correspondendo a uma acurácia de 6.00%. Ambos os modelos superaram drasticamente o baseline de 3.48%, com o Modelo 1 exibindo uma melhora de mais de 100% sobre o modelo base. A Tabela 2 sumariza a comparação de desempenho.

Modelo	Nº de Testes	Sucessos	Falhas	Acurácia de Execução (%)
Modelo Base	1034	36	998	3.48%
Fine-Tuned 1	1034	75	959	7.25%
Fine-Tuned 2	1034	62	972	6.00%

TABLE II
RESULTADOS DOS TESTES DE EXECUÇÃO DOS MODELOS

Os resultados indicam que, para esta tarefa, a estratégia de treinamento com uma taxa de aprendizado mais alta e menos épocas (Modelo 1) foi marginalmente mais eficaz do que um treinamento mais longo e com uma taxa de aprendizado mais conservadora (Modelo 2).

2) Análise Qualitativa de Erros: Apesar da melhora significativa, os modelos fine-tuned ainda apresentaram falhas, embora de natureza distinta das observadas no baseline. A análise do relatório de erros revelou que as falhas passaram de simples alucinações de esquema para erros de raciocínio lógico mais complexos. A seguir, são apresentados dois exemplos representativos extraídos da avaliação do "Modelo 1".

Exemplo de Falha 1: Erro de Lógica em Agregação Complexa:

1) Pergunta: "Qual é a nacionalidade mais comum entre as pessoas?"
2) SQL Correta (Gabarito): SELECT Nationality FROM people GROUP BY Nationality ORDER BY COUNT(*) DESC LIMIT 1
3) SQL Gerada (Modelo 1): SELECT nationality, COUNT(*) AS num-people FROM people GROUP BY nationality ORDER BY num-people DESC
4) Análise do Erro: Neste caso, a consulta gerada pelo modelo é sintaticamente perfeita e logicamente quase correta. O modelo conseguiu identificar a tabela people, a coluna Nationality, e a necessidade de agrupar (GROUP BY) e contar (COUNT(*)). No entanto, ele falhou no passo final de interpretação: a pergunta pede "a nacionalidade mais comum" (um único resultado), mas a SQL gerada retorna a lista de todas as nacionalidades e suas contagens. O modelo não aplicou o LIMIT 1 necessário para responder precisamente à pergunta. Isso representa uma falha de raciocínio, não de conhecimento do esquema.

Exemplo de Falha 2: Erro de Compreensão de Cláusulas de Exceção:

1) Pergunta: "Quais profissionais não realizaram nenhum tratamento em cães?"
2) SQL Correta (Gabarito): SELECT professional-id , role-code , email-address FROM Professionals EXCEPT SELECT T1.professional-id , T1.role-code , T1.email-

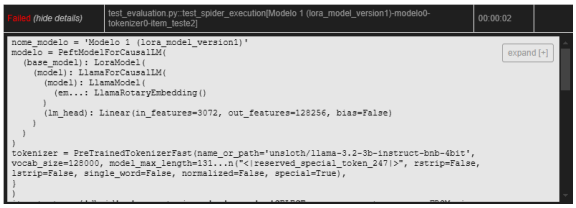

Fig. 2. Erro Captado pelo Pytest.

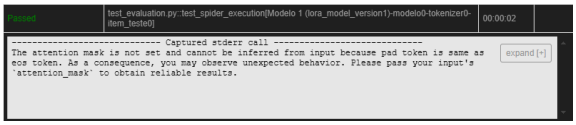

Fig. 3. Acerto Captado pelo Pytest

address FROM Professionals AS T1 JOIN Treatments AS T2 ON T1.professional-id = T2.professional-id

3) SQL Gerada (Modelo 1): SELECT T1.id, T1.role, T1.email FROM professional AS T1 LEFT JOIN treatment AS T2 ON T1.id = T2.professional-id WHERE T2.animal = 'dog' AND T2.treatment = 'none'
4) Análise do Erro: O modelo tentou resolver um problema de exclusão usando LEFT JOIN com uma condição WHERE, uma abordagem que pode ser válida mas que, neste caso, foi implementada de forma incorreta e com alucinação de colunas (T2.animal, T2.treatment). Ele não conseguiu abstrair a necessidade de usar uma operação de conjunto como EXCEPT ou uma subconsulta com NOT IN, que seria a forma canônica de resolver esta questão. Isso indica uma dificuldade em traduzir negativas e exceções complexas da linguagem natural para a lógica SQL correspondente.

Esses exemplos demonstram que, embora o fine-tuning tenha sido eficaz em ensinar o esquema do banco de dados e a sintaxe básica de SQL ao modelo, persistem desafios na interpretação de intenções lógicas complexas e na formulação de consultas não triviais.

C. Resultados MMLU

Esta seção apresenta os resultados quantitativos da avaliação de desempenho dos modelos, focando tanto na performance na tarefa-alvo de Text-to-SQL quanto na análise de regressão de capacidade geral medida pelo benchmark MMLU.

1) Análise de Regressão de Capacidade (MMLU): Para quantificar o impacto do fine-tuning sobre o conhecimento geral e as capacidades de raciocínio dos modelos, o modelo base e as duas variantes fine-tuned foram avaliados em um conjunto de questões do benchmark MMLU, abrangendo três domínios: Ciências Sociais (Economia), Humanidades (Filosofia) e STEM (Ciência da Computação). A metodologia de avaliação, baseada em 4-shot e cálculo de log-likelihood, foi aplicada a cada modelo.

Os resultados, sumarizados na Tabela 3, revelam um impacto heterogêneo do processo de fine-tuning nos diferentes domínios de conhecimento.

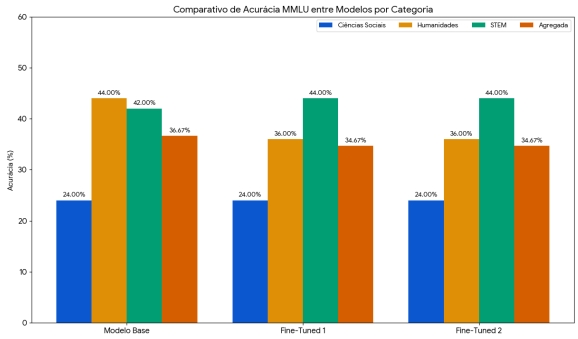

Fig. 4. COMPARATIVO DE ACURÁCIA DOS MODELOS NO BENCHMARK MMLU

A análise dos dados permite extrair três observações principais:

1) Regressão de Capacidade em Humanidades: O impacto mais significativo do fine-tuning foi uma acentuada queda de desempenho na categoria de Humanidades. Ambos os modelos especializados sofreram uma regressão de 8.00 pontos percentuais em comparação com o modelo base. Este resultado sugere que a especialização na tarefa lógica e estruturada de Text-to-SQL pode ter degradado a capacidade do modelo de lidar com o raciocínio mais abstrato e nuanceado, característico de domínios como a filosofia.
2) Melhora de Desempenho em STEM: Em contraste, observou-se uma melhora de 2.00 pontos percentuais na categoria STEM. Este ganho indica uma possível sinergia positiva, onde o treinamento em uma tarefa lógica como a geração de SQL pode ter reforçado as habilidades de raciocínio técnico e matemático do modelo, resultando em um desempenho superior em ciência da computação.
3) Desempenho Idêntico dos Modelos Especializados: Um achado notável é que os modelos "Fine-Tuned 1" e "Fine-Tuned 2", apesar de terem sido treinados com hiperparâmetros distintos (taxa de aprendizado e número de épocas), apresentaram um desempenho absolutamente idêntico em todas as categorias do MMLU. Isso sugere que, para a tarefa de conhecimento geral, as diferentes estratégias de fine-tuning convergiram para um estado funcionalmente equivalente, ou que o impacto do fine-tuning na base de conhecimento foi similar em ambos os casos.

De forma agregada, a perda substancial em Humanidades não foi totalmente compensada pelo ganho em STEM, resultando em uma regressão de capacidade geral de 2.00 pontos percentuais para ambos os modelos fine-tuned.

III. DISCUSSÃO

A análise dos resultados obtidos revela uma dinâmica complexa e multifacetada sobre o processo de especialização de modelos de linguagem. O fine-tuning, embora comprovadamente eficaz para a tarefa-alvo de Text-to-SQL, induziu

uma mensurável, ainda que modesta, regressão na capacidade de conhecimento geral do modelo. Esta seção aprofunda a análise desse trade-off, examina os fatores que podem tê-lo influenciado e discute as implicações práticas destes achados.

A. Análise do Trade-off: Custo vs. Benefício da Especialização

A questão central de qualquer processo de fine-tuning é se o ganho de desempenho na tarefa de especialização justifica a potencial perda de capacidades gerais. Os resultados quantitativos sugerem que, neste caso, o balanço é amplamente favorável. O desempenho na tarefa de Acurácia de Execução de SQL saltou de um baseline de 3.48% para 7.25% com o Modelo 1, representando um aumento relativo de mais de 108%. Este ganho transformou um modelo praticamente inútil para a tarefa em uma ferramenta com uma capacidade funcional inicial.

Em contrapartida, o "custo" dessa especialização foi uma queda agregada de apenas 2.00 pontos percentuais na acurácia do benchmark MMLU. A magnitude do benefício na tarefa-alvo, que é o objetivo primário do desenvolvimento, parece justificar amplamente essa perda marginal em domínios de conhecimento geral. Para uma aplicação comercial onde o modelo seria empregado como um chatbot de banco de dados, a duplicação da sua eficácia na tarefa principal supera em muito uma ligeira redução na sua capacidade de, por exemplo, discorrer sobre filosofia. Portanto, o trade-off observado é não apenas aceitável, mas desejável do ponto de vista prático.

B. Fatores de Influência no Trade-off

A análise dos resultados permite inferir sobre os fatores que influenciam esta dinâmica de ganho e perda.

Primeiramente, a arquitetura de fine-tuning (LoRA) desempenhou um papel crucial em mitigar o esquecimento catastrófico. Ao congelar a vasta maioria dos pesos do modelo base e treinar apenas os adaptadores de baixo ranque, a técnica de PEFT demonstrou ser altamente eficaz em preservar o conhecimento pré-treinado. Uma regressão de apenas 2 pontos percentuais, em vez de um colapso completo das capacidades gerais, é um testemunho da eficiência desta abordagem.

Em segundo lugar, a análise dos hiperparâmetros revela insights interessantes. Apesar das estratégias de treinamento distintas — o Modelo 1 com 1 época e alta taxa de aprendizado (2e-4) e o Modelo 2 com 3 épocas e baixa taxa de aprendizado (2e-8) — ambos convergiram para um estado de conhecimento geral idêntico nos testes do MMLU. No entanto, a estratégia mais "agressiva" do Modelo 1 resultou em um desempenho ligeiramente superior na tarefa de SQL (7.25% vs. 6.00%). Isso sugere que, para a complexidade deste dataset, o fator dominante na regressão de capacidade pode ser o próprio ato de se especializar em um domínio estreito, e não tanto a duração ou a sutileza do treinamento. Além disso, indica que um treinamento mais curto e com maior taxa de aprendizado pode ser mais eficiente para alcançar o desempenho máximo na tarefa-alvo sem incorrer em danos adicionais ao conhecimento geral.

Por fim, a natureza da tarefa e dos dados parece ser um fator determinante. A melhora de 2 pontos em STEM sugere uma sinergia positiva: treinar o modelo para entender a lógica estruturada de SQL pode ter reforçado suas vias neurais associadas a outras tarefas lógicas e técnicas. Em contrapartida, a queda de 8 pontos em Humanidades sugere uma "competição por recursos", onde a otimização para a lógica formal pode ter atrofiado a capacidade de lidar com a ambiguidade e a abstração características das humanidades.

C. Implicações Práticas para LLMs Comerciais

Os achados deste estudo têm implicações diretas para o desenvolvimento e a implantação de modelos de linguagem especializados em contextos comerciais.

A principal implicação é a necessidade de uma avaliação holística e contínua. Não é suficiente avaliar um modelo apenas na sua tarefa de especialização. As organizações devem implementar pipelines de avaliação que monitorem tanto o desempenho na tarefa-alvo (ganho) quanto a performance em benchmarks de conhecimento geral (custo). Isso permite uma tomada de decisão informada sobre o trade-off e a definição de limiares aceitáveis de regressão de capacidade, dependendo da aplicação final do modelo.

Além disso, a descoberta de sinergia entre domínios (SQL e STEM) abre portas para estratégias de treinamento mais inteligentes. Uma empresa poderia, por exemplo, realizar um fine-tuning intermediário em um corpus de dados lógicos ou técnicos para "pré-condicionar" um modelo antes de especializá-lo em uma tarefa final de alta complexidade, como análise de código ou interpretação de dados científicos.

Finalmente, os resultados sugerem que a otimização de hiperparâmetros para modelos especializados deve considerar não apenas a maximização da acurácia na tarefa-alvo, mas também a eficiência computacional. O fato de um treinamento mais curto (Modelo 1) ter produzido resultados superiores indica que treinar por mais épocas não é universalmente melhor e pode levar a custos computacionais desnecessários para ganhos marginais ou até mesmo negativos. Isso é particularmente relevante para empresas que buscam otimizar o ROI (Retorno sobre o Investimento) em seus ciclos de treinamento de IA.

D. Principal Erro no treinamento

Como os dados estão bem estruturados no database, uma forma de melhorar muito todos os resultados é especificar quais bancos de dados estão sendo usados para responder a consulta do usuário. Da mesma forma, ao ser aplicado o pytest, também deverão ser passados os nomes dos bancos de dados. Isso funcionará muito bem para este cenário específico.