UM ESTUDO DA TRANSFERÊNCIA DE APRENDIZADO EMPREGANDO MODELOS DE APRENDIZADO PROFUNDO: APLICAÇÕES DE PROCESSAMENTO DE LINGUAGEM NATURAL

Victor Pasquini Ribeiro Campos (IC)[1], Isabela Neves Drummond (PQ)[1]
[1]*Universidade Federal de Itajubá*

Palavras-chave: *Retrieval-Augmented Generation*, Modelos de Linguagem de Grande Escala, Quantização, LLama 2.

Introdução

Nos últimos anos, os Modelos de Linguagem de Grande Escala (LLMs), baseados em arquiteturas de aprendizado profundo (Janiesch, et al., 2021), têm desempenhado um papel fundamental no avanço do processamento de linguagem natural (PLN). O LLama 2 (Touvron, et al., 2023), desenvolvido pela Meta, destaca-se como um exemplo recente, aproveitando a arquitetura *Transformer* (Vaswani et al., 2017), que utiliza mecanismos de autoatenção para capturar relações contextuais em textos, superando as limitações das redes recorrentes.

Técnicas como aprendizado por transferência (AT) e *Retrieval-Augmented Generation* (RAG) têm sido essenciais para o aprimoramento desses modelos. O AT permite reutilizar conhecimentos pré-treinados, acelerando o treinamento para tarefas específicas. Além disso, reduz a necessidade de grandes recursos computacionais. O RAG (Lewis et al., 2020) combina geração de texto com recuperação de informações externas, elevando a precisão e contextualização das respostas geradas.

Este estudo explorou o impacto dessas técnicas, além de estratégias como quantização e adaptação de pesos, para otimizar o desempenho do LLama 2 em cenários com recursos computacionais limitados.

A quantização (Dettmers et al., 2022) reduz o tamanho dos modelos ao compactar suas representações. Enquanto a adaptação de pesos (Houlsby et al., 2019) permite ajustar pequenas partes da rede neural para novas tarefas, evitando a necessidade de recalibrar todo o modelo.

Essa pesquisa visa demonstrar que essas abordagens podem reduzir os requisitos de infraestrutura sem comprometer a performance dos LLMs.

Metodologia

O estudo focou no modelo LLama 2, especialmente na versão de 7 bilhões de parâmetros, e foi dividido em cinco etapas principais: (1) Identificação do ambiente de testes, (2) Configuração dos parâmetros e escolha dos dados, (3) Treinamento e *fine-tuning* e (4) Utilização da técnica RAG.

O primeiro desafio na primeira etapa (1) foi o alto consumo de recursos do modelo, testado inicialmente no Google Colab, uma plataforma Python com suporte a GPU (Bisong, 2019).

No entanto, ao iniciar o processo de treinamento do modelo, visando aplicar o AT, era exigido uma enorme quantidade de processamento gráfico e memória, o que limitava o objeto de estudo.

Para contornar essa limitação, foram usadas técnicas como QLoRA e PEFT discutidas por Dettmers, et al., 2024 e Houlsby et al., 2019, que permitiram a quantização e adaptação dos pesos do modelo.

A linguagem Python, juntamente com as bibliotecas da *Hugging Face* (Wolf, 2019), foi utilizada, destacando-se a classe SFTTrainer, adequada para *datasets* com labels e menor exigência de dados e processamento.

```
bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
    )
```

Código 1 - Configuração do *BitsAndBytes*

Partiu-se então para segunda etapa (2), onde se iniciou o ajuste dos parâmetros. Como é mostrado no Código 1, foi implementada a quantização em 4 bits, que permitiu uma considerável redução do uso de memória, mantendo a qualidade com quantização não linear, ideal para preservar a precisão do modelo.

```
lora_r = 16
lora_alpha = 64
lora_dropout = 0.1
lora_target_modules = [
    "q_proj",
    "up_proj",
    "o_proj",
```

```
    "k_proj",
    "down_proj",
    "gate_proj",
    "v_proj",
]
```

Código 2 - Configuração do LoRA

Além disso, de acordo com o Código 2, a redução do número de parâmetros treinados, indicada pelo parâmetro lora_r, permite que o modelo mantenha sua expressividade. Enquanto, ainda, reduz de maneira significativa o custo computacional, essencial em ambientes com limitações de hardware.
O fator de escalonamento lora_alpha amplifica as alterações nos pesos. Dessa forma, é garantido que as adaptações feitas pelo LoRA impactem o desempenho do modelo de forma significativa, sem perturbar o aprendizado já estabelecido.
Para evitar o *overfitting*, a taxa de *dropout* lora_dropout foi implementada, desativando uma fração dos neurônios durante o ajuste, o que promoveu uma melhor generalização do modelo.
Por fim, os módulos-alvo, listados em lora_target_modules, foram selecionados por sua importância no mecanismo de atenção e nas camadas *feedforward*. Isso maximizou o impacto do ajuste nas partes críticas do processamento interno do modelo.
A partir disso, foram realizados os testes de AT para diferentes funções. Para isso, foi essencial a utilização de *datasets* com intuito de treinar os modelos. *Datasets* estes que foram retirados do *Hub* de *datasets* da *Hugging Face*. Eles são criados pela comunidade e foram escolhidos de acordo com a compatibilidade e o cumprimento das diretrizes da comunidade.
O primeiro utilizado foi "*Abirate/english quotes*", um *dataset* voltado para classificação de texto, em que o modelo foi treinado com frases famosas de alguns renomados cientistas. Assim, a partir de um *prompt*, com apenas um trecho da frase, ele identificava o autor e gerava o restante da frase.
Esse teste anterior foi baseado num exemplo fornecido na documentação do PEFT da *Hugging Face* (Wolf, 2019) e utilizando-o como base, foram realizados outros testes, focados em outros objetivos.
Dentre eles, o de maior destaque, foi com o *dataset* "dair-ai/emotion", que continha sentenças rotuladas com as classes *sadness* (0), *joy* (1), *love* (2), *anger* (3), *fear* (4), *surprise* (5).
Seus ajustes foram feitos com as configurações aplicadas anteriormente, juntamente com os argumentos de treinamento conforme o Código 3.

```
training_arguments = TrainingArguments(
    per_device_train_batch_size=4,
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit",
    logging_steps=1,
    learning_rate=1e-4,
    fp16=True,
    max_grad_norm=0.3,
    num_train_epochs=2,
    evaluation_strategy="steps",
    eval_steps=0.2,
    warmup_ratio=0.05,
    save_strategy="epoch",
    group_by_length=True,
    output_dir=OUTPUT_DIR,
    report_to="tensorboard",
    save_safetensors=True,
    lr_scheduler_type="cosine",
    seed=42,
)
```

Código 3 - Argumentos de treinamento

A configuração dos argumentos de treinamento foi otimizada para equilibrar eficiência e desempenho. O parâmetro *per_device_train_batch_size* permitiu o funcionamento em dispositivos com recursos limitados, enquanto *gradient_accumulation_steps* estabilizou o treinamento ao simular lotes maiores.
A taxa de aprendizado (*learning_rate*) foi ajustada para promover um aprendizado gradual, evitando oscilações nos pesos. O uso de *fp16* melhorou a eficiência sem sacrificar a precisão, tornando o treinamento mais rápido e menos intensivo em memória.
O controle de gradientes com *max_grad_norm* preveniu explosões, assegurando estabilidade, e a *evaluation_strategy* monitorou o desempenho continuamente, permitindo ajustes em tempo real. O *warmup_ratio* facilitou um aumento gradual da taxa de aprendizado, evitando oscilações iniciais, enquanto o *lr_scheduler_type* garantiu um decaimento suave da taxa de aprendizado, favorecendo a convergência do modelo.
Após os resultados iniciais da terceira etapa (3), ainda nela uma nova abordagem foi adotada. Dessa vez, utilizou-se *datasets* externos em formato CSV de outras fontes, como o Repositório de dados Kaggle. Com isso, seria possível manipular os dados de maneira personalizada.
Apesar de parcialmente promissores, os testes com um *dataset* de uma loja de suprimentos com produtos com preços e nomes bem exóticos, revelaram que o modelo não conseguia fixar corretamente essas informações. Um exemplo disso, o nome do produto 1 chamado "*David's Cookies Mile High Peanut Butter Cake"* com o

preço de $56,99.
Foi inferido ao modelo uma pergunta sobre os produtos com o preço referido anteriormente, mas dentre o resultado, não havia o produto 1. Ao invés disso, foram gerados outros produtos como "*Heavenly Layers Chocolate Hazelnut Cake"*, mas que foram condizentes com as características do dataset.
Embora o modelo gerasse novos nomes e valores semelhantes com base nas informações existentes, ficou claro que uma técnica diferente era necessária para respostas estritamente baseadas em dados.
Assim, deu-se início à última etapa (4), em que a pesquisa passou a explorar a técnica *Retrieval-Augmented Generation* (RAG). Esta permitiu usar *datasets* como consultas diretas, garantindo a recuperação das informações para gerar respostas desejadas.
A técnica foi aplicada no mesmo ambiente e linguagem anteriores, usando a biblioteca *transformers*. O experimento envolveu a definição de um prompt e uma lógica para transformar perguntas em embeddings, ou vetores numéricos. Um modelo especializado gerava esses embeddings de forma resumida e estruturada.
O conteúdo de um arquivo PDF era dividido em seções, convertidas em embeddings com índices associados. Assim, ao receber uma pergunta, o sistema acessava os embeddings armazenados em um VectorStoreIndex e gerava a resposta com base no documento.

Resultados e discussão

Dos experimentos iniciais, com treinamento utilizando o ajuste fino, foram obtidos bons resultados comparando-os aos resultados ideais previstos pelo *dataset*. O primeiro, utilizando o *dataset* com frases famosas, houve êxito do modelo ao gerar o restante das frases como seguia-se na base de dados.

```
batch = tokenizer("Two things are infinite: ",
return_tensors='pt')

with torch.cuda.amp.autocast():
  output_tokens = model.generate(**batch,
max_new_tokens=50)

print('\n\n', tokenizer.decode(output_tokens[0],
skip_special_tokens=True))
```

Código 4 – Inferência do modelo

Com a inferência mostrada no código 4, o resultado foi o seguinte: "Two things are infinite: the universe and human stupidity; and I'm not sure about the universe. -Albert Einstein I'm not sure about the universe either", é possível observar a eficácia ao completar a sentença e denominar o autor, exatamente como é descrito no dataset: "Two things are infinite: the universe and human stupidity; and I'm not sure about the universe."
Dessa forma, parte-se para os outros testes com ajuste fino, dessa vez com mais dados e técnicas de avaliação do treinamento.
Foi então utilizado um *dataset* rotulado com sentenças, em que cada *label* (rótulo) classificava uma delas com sentimento de 0 a 5 já apresentados. Após realizar o treinamento, foram obtidas algumas métricas de avaliação utilizadas.
Durante o treinamento e a avaliação, a métrica de perda (*loss*) mostrou que o modelo estava ajustando bem seus parâmetros, com valores de 0,7769 na avaliação e 0,4628 no treinamento, indicando aprendizado eficiente. O tempo de execução foi de 186,9 segundos na avaliação e cerca de 45 minutos no treinamento, refletindo a complexidade do modelo.
A taxa de processamento foi de 5,35 amostras por segundo na avaliação e 1,118 no treinamento, com uma quantidade relativamente baixa de etapas por segundo, o que se deve à grande quantidade de operações envolvidas. A taxa de aprendizado foi reduzida gradualmente, ajudando a estabilizar o modelo, e quase duas épocas de treinamento foram completadas, sugerindo que mais épocas podem melhorar o desempenho, desde que o *overfitting* seja evitado.
Dados os testes com a porção *"test"* do *dataset* escolhido, foram obtidos resultados promissores, mas não totalmente corretos.

Tabela 1 - Resultados dos Modelos Treinado (FT), base (N) e resposta correta (R)

Prompt	FT	N	R
i was feeling really troubled and down over what my dad said	0	0	0
i feel so thrilled to have three such distinguished individuals such as yourselves here	1	0	1

A partir do mesmo *prompt* do treinamento, foram gerados 10 testes, utilizando o modelo ajustado e o modelo sem ajuste fino. O modelo ajustado obteve as 3 primeiras respostas corretas, entretanto, avaliou todas as seguintes como 0 (*sadness*).
Embora não tenha sido tão satisfatório, o mesmo *prompt* aplicado ao modelo não ajustado teve um desempenho pior, ao avaliar todas as sentenças como 0 (*sadness*). Uma porção desse teste pode ser visualizado na Tabela 1, com o exemplo de duas frases. Uma com sentimento 0 (sadness) e outra com 1 (joy), em que o modelo

ajustado obteve êxito, enquanto o modelo base não.
Dessa forma, é possível concluir que o ajuste, apesar de possuir uma quantia baixa de dados, juntamente de adaptadores limitantes do modelo, o ajuste fino atingiu seu objetivo de maneira parcial. Com uma maior capacidade de processamento e quantidade de informações, é provável uma melhora significativa.
Em relação à utilização da técnica de RAG, as atividades tiveram foco em um cronograma da disciplina de Engenharia de Software I da UNIFEI. O método utilizado seria de gerar as respostas com o modelo, enquanto se utiliza de resposta ideal às informações correspondentes no documento.

Tabela 2 - Resultado do RAG

Pergunta	Resposta esperada	RAG
Quais são os temas dos seminários?	Seminários: 1. Gerência de Configuração e Ferramentas de Gerência de Configuração 2. Ferramentas de Issue Tracking 3. Teste de Software e Automação de Teste de Software 4. Gerência de Projetos de Software	Seminar 1: Gerência de Configuração and Ferramentas de Apoio ao Controle de Versão de Software Seminar 2: Garantia da Qualidade e Ferramentas de Issue Tracking
Quanto tempo tenho para apresentar o seminário	De 30 a 40 minutos	Based on the provided context information, you have 30 minutes to 40 minutes to present your seminar.

Foram feitas as inferências, com 10 perguntas diversas sobre metodologia, atividades e assuntos da disciplina, conforme se observa uma porção deles na Tabela 2. Nestes exemplos, o RAG foi parcialmente bem sucedido, respondendo de forma incompleta a primeira e corretamente a segunda.
O modelo teve êxito em recuperar informações mais simples e concisas, como temas de trabalho, tempo de duração de apresentação e atividades a serem entregues.
No entanto, apresentou muitas dificuldades e fracassos ao buscar informações como links, datas, além de não manter uma consistência coesa e precisa nas respostas, o que pode ser fruto da estrutura do pdf, quanto da utilização de um modelo de menor porte.

Conclusões

Este estudo explorou o uso de LLMs, com foco no LLama 2, em ambientes de recursos limitados. Verificou-se que é possível ajustar esses modelos para tarefas de PLN, como classificação de texto, apesar das limitações de infraestrutura.
Técnicas como quantização e adaptação de pesos, junto com ferramentas como Google Colab e Hugging Face, reduziram os requisitos computacionais sem comprometer o desempenho. No entanto, o LLama 2 apresentou eficiência inferior ao esperado, sugerindo a necessidade de infraestrutura mais robusta.
A técnica RAG, ao integrar informações externas, foi eficaz, melhorando a precisão, especialmente em contextos educacionais.

Agradecimentos

Agradeço à UNIFEI pelo apoio financeiro que possibilitou a realização deste estudo. Expresso minha gratidão à Professora Isabela Drummond pela orientação e suporte acadêmico, juntamente ao Flávio Mota e ao Professor Adler Diniz.

Referências

JANIESCH, Christian; ZSCHECH, Patrick; HEINRICH, Kai. Machine learning and deep learning. Electronic Markets, v. 31, n. 3, p. 685-695, 2021.
VASWANI, A. Attention is all you need. Advances in Neural Information Processing Systems, 2017.
TOUVRON, Hugo et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.
DETTMERS, Tim et al. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. Advances in Neural Information Processing Systems, v. 35, p. 30318-30332, 2022.
HOULSBY, Neil et al. Parameter-efficient transfer learning for NLP. In: International conference on machine learning. PMLR, 2019. p. 2790-2799.
LEWIS, Patrick et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, v. 33, p. 9459-9474, 2020.
DETTMERS, Tim et al. Qlora: Efficient finetuning of quantized llms. Advances in Neural Information Processing Systems, v. 36, 2024.
BISONG, Ekaba; BISONG, Ekaba. Google colaboratory. Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners, p. 59-64, 2019.
WOLF, T. Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771, 2019.