Capítulo 3

Treinamento de Grandes Modelos de Linguagem na prática

João Vitor Mariano Correia
João Renato Ribeiro Manesco
Danilo Samuel Jodas
Douglas Rodrigues
Gabriel Lino Garcia
Pedro Henrique Paiola
João Paulo Papa

Publicado em: 16/04/2026

3.1 Introdução

Grandes Modelos de Linguagem (LLMs) representam uma mudança de paradigma no Processamento de Linguagem Natural, impulsionada pela arquitetura de *Transformers* (Vaswani et al., 2017b), que permitiu um avanço significativo na capacidade dos modelos de ponderar contexto. Trabalhos pioneiros como o BERT (Devlin et al., 2019), com sua abordagem de pré-treinamento bidirecional, validaram a eficácia de treinar modelos massivos em vastos conjuntos textuais, dando um passo fundamental para a compreensão de linguagem via redes neurais. O resultado é uma série de modelos com uma capacidade sem precedentes de processar e gerar texto de forma fluente e coerente.

Essa fundação de entendimento de linguagem é adquirida durante a fase de pré-treinamento, em que o modelo absorve conhecimento de fontes como a internet e grandes acervos de livros, permitindo que diversas aplicações de ampla relevância possam surgir, tais como sistemas já discutidos anteriormente nesta edição, como modelos de linguagem introduzidos no Capítulo Modelos de linguagem, tradução automática (Capítulo Tradução Automática), sumarização automática (Capítulo Sumarização Automática), abrangendo diferentes contextos de uso (Minaee et al., 2024).

Apesar da eficácia dos LLM, sua aplicabilidade em cenários do mundo real depende criticamente de sua especialização, em especial tratando-se de modelos menores. A base generalista, embora ampla, é insuficiente para tarefas que exigem mais precisão ou conhecimento de nicho, tornando o processo de adaptação uma necessidade para determinados contextos (Anisuzzaman et al., 2025). Essa necessidade de adaptação pode ser analisada a partir de duas dimensões principais e complementares: a especialização para domínios de conhecimento e o alinhamento para contextos linguísticos e culturais específicos.

A primeira dimensão busca fazer com que o corpo de conhecimento dos modelos de linguagem seja composto por termos linguísticos e conhecimentos específicos de um campo nichado, facilitando sua aplicação em tarefas direcionadas, sendo crucial para aplicações de

alto impacto. No campo jurídico, por exemplo, um modelo refinado é capaz de interagir mais precisamente com o vernáculo próprio de documentos legais para realizar sumarizações ou revisões (Lai et al., 2024). Na medicina, a especialização permite que um modelo interprete literatura científica e registros de saúde para que seja capaz de apoiar decisões clínicas (Zhou et al., 2023b). Da mesma forma, no setor financeiro, um modelo ajustado pode identificar nuances em relatórios de mercado que seriam invisíveis a um sistema generalista (Lee et al., 2024). Nesses contextos, o refinamento é o que distingue uma ferramenta funcional de um sistema verdadeiramente confiável e eficaz.

A segunda dimensão busca realizar a adaptação linguística e cultural dos modelos, um desafio fundamental na construção de modelos equitativos. A grande maioria dos dados utilizados no pré-treinamento de modelos publicados abertamente está em inglês, o que cria um desequilíbrio cultural e representativo a nível de linguagem nesses modelos (AlKhamissi et al., 2024). Desta forma, modelos pouco expostos à língua portuguesa sofrem dificuldade em entender certas nuances linguísticas, bem como podem estar expostos a certos vieses culturais dos dados, como é o caso de modelos médicos que ignoram a possibilidade de diagnóstico de certas doenças endêmicas como a dengue pelos dados de treinamento serem de locais não expostos a este problema (Paiola et al., 2024).

Para que a tecnologia seja efetivamente democrática e útil em contextos não anglófonos, como o brasileiro, a tradução da resposta nem sempre é a solução. Exigindo um processo de ajuste fino ou continuação de pré-treino com dados no idioma local. Garantindo que o modelo compreenda as particularidades culturais, expressões idiomáticas e o contexto social do idioma refinado.

Atingir um alto nível de robustez com LLMs, no entanto, requer um processo que vai além do treinamento em escala. É necessária uma estratégia de adaptação cuidadosa, que considere tanto a profundidade técnica de um domínio quanto a riqueza de um contexto linguístico. Para abordar essa necessidade, este capítulo irá explorar o espectro de técnicas de especialização utilizadas no treinamento de modelos de linguagem. Iniciaremos discutindo a continuação de pré-treino, seguiremos para o ajuste fino supervisionado em suas diversas formas, incluindo técnicas de parametrização eficiente e aprendizado *few-shot*, e concluiremos com métodos de alinhamento, que refinam o comportamento do modelo com base em preferências humanas. Ao final do capítulo, apresentamos uma implementação prática do refinamento de modelos.

3.2 Continuação de pré-treino

A Continuação de Pré-treino (CPT) consiste em uma técnica de adaptação realizada antes do ajuste fino supervisionado, visando ampliar o conhecimento de um LLM em um domínio específico. Essa etapa intermediária entre o pré-treinamento generalista e o *fine-tuning* retoma o treinamento original do modelo, de forma autossupervisionada, através de técnicas como a Modelagem de Linguagem Mascarada (MLM) ou Causal (CLM), aplicadas em um *corpus* especializado de grande escala. Esta estratégia resulta na incorporação de vocabulário, conceitos e fatos do domínio desejado. Neste capítulo serão apresentados os fundamentos técnicos, cenários de aplicação e distinções relevantes dessa abordagem.

3.2.1 Mecanismos de treinamento autossupervisionado

Para modelos autorregressivos, como os das famílias GPT e Qwen, a técnica predominante é a de Modelagem de Linguagem Causal (CLM) (Bai et al., 2023; Radford et al., 2019b),

cujo objetivo é treinar o modelo para prever o próximo *token* em uma sequência com base nos *tokens* anteriores. De forma mais precisa, a probabilidade de uma sequência de *tokens* é decomposta em uma cadeia de probabilidades condicionais, onde cada *token* é previsto condicionando-se nos *tokens* que o precedem.

O objetivo da CLM é maximizar essa probabilidade nos textos observados, o que equivale a minimizar a soma dos erros associados a cada previsão individual, ou seja, a entropia cruzada entre as distribuições preditas e os *tokens* reais. A natureza estritamente unidirecional, da esquerda para a direita, do CLM flui naturalmente com a arquitetura do tipo *decoder* em arquiteturas de *transformers*, que empregam máscaras nos mecanismos de atenção. Essas máscaras garantem que, ao prever um *token* em uma determinada posição, o modelo utilize apenas informações dos *tokens* que o antecedem na sequência, e não dos posteriores.

Em contrapartida, arquiteturas bidirecionais como o BERT utilizam a Modelagem de Linguagem Mascarada (MLM) (Devlin et al., 2019), cujo funcionamento, ao invés de prever o próximo *token* de uma sequência, trabalha corrompendo uma fração dos *tokens* no texto de entrada, criando uma espécie da "máscara" no texto, desta forma, o modelo deve prever quais os *tokens* originais que deveriam compor a sequência. Nesse processo, para uma sequência de entrada T, um subconjunto de posições M é selecionado para mascaramento. Para cada posição $i \in M$, um *token* t_i pode ser substituído pelo símbolo de máscara, por um *token* aleatório ou mantido inalterado, de modo a incentivar o modelo a aprender representações contextuais mais robustas. A função de perda é calculada apenas sobre os *tokens* mascarados, já que o modelo se concentra em predizer esses elementos, e como a predição é condicionada à sequência inteira corrompida, o modelo aprende representações contextuais, considerando tanto o contexto à esquerda quanto à direita.

Outra estratégia, chamada de Previsão da Próxima Sentença (NSP), introduzida no pré-treinamento do BERT, consiste em classificar se uma sentença sucede outra no texto. Diferente da Modelagem CLM, que realiza predição *token* a *token* de forma unidirecional, o NSP busca classificar sentenças completas. Essa técnica, no entanto, caiu em desuso no estado da arte, devido a estudos indicarem benefício limitado, e foi descartada em modelos recentes (Liu et al., 2019b).

3.2.2 Contextos e objetivos da continuação de pré-treino

A aplicação da continuação de pré-treino é uma decisão estratégica que visa aprimorar o modelo de maneiras que o ajuste fino supervisionado por si só não consegue alcançar. O principal desses cenários é a adaptação para aplicações específicas, onde um LLM generalista é infundido com o conhecimento de um campo técnico (Shi et al., 2024). Um exemplo prático seria treinar um LLM com uma biblioteca de artigos jurídicos, de tal forma que ele se torne fluente no jargão e nos conceitos do direito, ou até mesmo em conhecimento médico, como será trabalhado ao longo deste capítulo. Outro uso importante é a adaptação temporal, que atualiza o conhecimento do modelo para além de sua data de corte, utilizando um conjunto de textos e notícias recentes. Adicionalmente, a CPT é útil para adaptações de idioma, melhorando a proficiência de um modelo em uma língua menos representada nos dados de treinamento originais, como o português, aprimorando sua fluência e compreensão de nuances culturais.

É importante reforçar a distinção entre a CPT e o Ajuste Fino Supervisionado, já que a CPT tem o objetivo de expandir a base de conhecimento geral de um modelo, através da utilização de dados não estruturados. Já o ajuste fino busca impor um comportamento

específico ao modelo, utilizando dados rotulados preparados, para uma ou mais tarefas específicas. A CPT opera sobre conjuntos massivos de texto bruto, o resultado é um modelo com vocabulário e conhecimento de domínio aprofundados, mas não necessariamente com os detalhes específicos das tarefas em que deve atuar.

3.3 Ajuste fino de modelos

Conforme explorado na seção anterior, na fase de pré-treinamento os Grandes Modelos de Linguagem adquirem um vasto conhecimento de mundo e uma profunda compreensão de padrões linguísticos, adquiridos a partir da exposição a *corpora* textuais massivos. Essa base de conhecimento confere aos modelos a capacidade de realizar uma gama de tarefas de forma emergente, sem treinamento explícito. Contudo, para alcançar excelência e confiabilidade em aplicações especializadas, é necessário um processo de adaptação direcionado. Este processo é conhecido como ajuste fino (do inglês, *fine-tuning*).

O ajuste fino representa uma importante mudança de paradigma na história do Processamento de Linguagem Natural. Ele pode ser definido como a especialização de um modelo generalista, transformando-o em uma ferramenta especialista capaz de executar tarefas com alta precisão e adaptar-se a domínios de conhecimento específicos (Wu et al., 2025). A intuição por trás desta técnica pode ser ilustrada por uma analogia com a formação médica: o pré-treinamento é análogo aos anos de faculdade, onde se adquire um conhecimento abrangente. O ajuste fino, por sua vez, equivale à residência médica, na qual o profissional aprofunda seu conhecimento em uma área específica, como a cardiologia, para diagnosticar e tratar com uma proficiência que um clínico geral não possui. De forma similar, o ajuste fino não ensina o LLM a gerar linguagem do zero; ele refina suas competências preexistentes para uma nova aplicação.

Formalmente, a técnica se fundamenta no princípio da aprendizagem por transferência (do inglês, *transfer learning*), no qual um modelo desenvolvido para uma tarefa inicial é reutilizado como ponto de partida para treinar outro modelo em uma segunda tarefa, relacionada à primeira. O conhecimento capturado durante o pré-treinamento em um domínio de origem e tarefa de origem é transferido para melhorar o desempenho e acelerar a aprendizagem em um domínio de destino e uma tarefa de destino. Essa abordagem mostrou-se fundamental para superar um dos maiores desafios históricos da área: a necessidade de vastos conjuntos de dados rotulados e o custo computacional proibitivo do treinamento de modelos de ponta a partir do zero (Howard; Ruder, 2018b).

O principal método para essa especialização é o Ajuste Fino Supervisionado (do inglês, *Supervised Fine-Tuning* ou SFT) (Devlin et al., 2019). Neste processo, o modelo pré-treinado tem seu treinamento estendido, mas desta vez em um conjunto de dados menor e de alta qualidade, composto por exemplos rotulados que demonstram o comportamento desejado — geralmente em um formato de pares de entrada e saída (por exemplo, no caso da tarefa de *Question Answering* (QA), o par `pergunta -> resposta esperada`). O mecanismo é uma continuação direta do treinamento de redes neurais:

1. **Seleção do modelo base**: A escolha de um modelo pré-treinado apropriado é o primeiro passo. A decisão deve levar em conta a arquitetura, o tamanho e a afinidade do domínio de pré-treinamento com a tarefa final.

2. **Preparação do *dataset***: A qualidade do ajuste é diretamente proporcional à qualidade do conjunto de dados. Esta etapa envolve a coleta, limpeza e formatação

de dados rotulados e representativos da tarefa, geralmente estruturados em pares de *prompt*-resposta.

3. **Ajuste de hiperparâmetros**: A seleção de hiperparâmetros como taxa de aprendizado, tamanho do lote (do inglês, *batch size*) e número de épocas é crucial. Um ajuste inadequado pode levar a um treinamento instável ou a um sobreajuste.

4. **Treinamento e validação**: O modelo é treinado em lotes de dados, e seu desempenho é continuamente monitorado em um conjunto de validação separado. O monitoramento da perda de validação é essencial para detectar sobreajuste e aplicar estratégias como a parada antecipada, que interrompe o treinamento quando a generalização para dados não vistos começa a piorar.

Este processo iterativo refina gradualmente o conhecimento do modelo, especializando-o para as nuances da tarefa alvo. Neste ponto, é crucial diferenciar o ajuste fino de outras técnicas de adaptação:

- **Continuação de pré-treino**: Enquanto a continuação de pré-treino visa expandir a base de conhecimento geral do modelo em um novo domínio (ex: alimentar um LLM com uma biblioteca de artigos jurídicos), utilizando dados não estruturados, o ajuste fino visa ensinar uma habilidade ou comportamento específico (ex: ensinar o modelo a sumarizar processos judiciais), utilizando dados rotulados.

- **Engenharia de *prompts***: Esta técnica não modifica nenhum dos parâmetros do modelo. Em vez disso, ela guia o comportamento do modelo no momento da inferência, formulando cuidadosamente a entrada (*prompt*) para induzir a resposta desejada. Métodos como *one-shot* ou *few-shot learning*, onde um ou alguns exemplos da tarefa são incluídos no *prompt*, são formas de engenharia de *prompts*. É uma forma de adaptação sem treinamento, que depende inteiramente das capacidades de aprendizado no contexto do modelo base.

Em suma, o ajuste fino representa um caminho poderoso para a especialização de LLMs, capacitando-os a ir além da fluência geral para entregar desempenho de ponta em aplicações de nicho. Contudo, a estratégia pela qual essa adaptação de parâmetros é realizada varia significativamente, levando a um espectro de métodos com diferentes *trade-offs* entre desempenho, custo computacional e complexidade. Nas subseções a seguir, exploraremos as principais abordagens práticas: desde o ajuste fino completo, que modifica todos os parâmetros do modelo, passando pelas eficientes técnicas de PEFT (do inglês, *Parameter-Efficient Fine-Tuning*), que ajustam apenas um pequeno subconjunto de parâmetros, até as estratégias de *Few-Shot Learning*, que operam na fronteira entre o ajuste e a engenharia de *prompts*. As seções seguintes exploram técnicas avançadas como *Reinforcement Learning from Human Feedback* (RLHF) e *Direct Preference Optimization* (DPO), que estendem o SFT para alinhamento com preferências humanas.

3.3.1 Ajuste fino completo

O ajuste fino completo (do inglês, *Full Fine-Tuning* ou FFT) representa a forma mais direta e conceitualmente mais simples de especialização de um modelo de linguagem. Nesta abordagem, a totalidade dos parâmetros do modelo pré-treinado é descongelada e se torna treinável. Isso concede ao modelo máxima flexibilidade para que todo o seu conhecimento

generalista, encapsulado em bilhões de parâmetros, seja reconfigurado para se adaptar profundamente aos padrões do novo conjunto de dados (Lv et al., 2024; Raffel et al., 2020b).

A principal vantagem desta abordagem é seu potencial para alcançar o máximo desempenho possível na tarefa alvo. Ao permitir que todas as camadas da rede neural sejam atualizadas, desde as representações iniciais de *embeddings* até os mecanismos de atenção e as camadas *feed-forward*, o modelo pode aprender nuances extremamente complexas e específicas do domínio, algo que abordagens mais restritivas poderiam não capturar. No entanto, este potencial de desempenho vem acompanhado de custos e desafios significativos, que devem ser cuidadosamente ponderados:

- **Alto custo computacional e de memória**: O FFT é extremamente exigente em termos de recursos. Treinar todos os parâmetros de um modelo com bilhões de pesos requer acesso a múltiplas GPUs de ponta com alta capacidade de VRAM, não apenas para armazenar os pesos do modelo, mas também os gradientes, os estados do otimizador e as ativações intermediárias durante o treinamento. Isso torna o processo financeiramente proibitivo para muitas organizações.
- **Custo de armazenamento e implantação**: Como o FFT modifica todos os pesos, cada nova tarefa especializada resulta em uma cópia completa e independente do modelo, que pode ocupar dezenas ou centenas de gigabytes. Gerenciar, armazenar e servir múltiplas instâncias de modelos tão grandes em um ambiente de produção é logisticamente complexo e caro.
- **Esquecimento catastrófico**: Este é um dos riscos mais proeminentes. Ao se especializar intensamente na nova tarefa, o modelo pode sobrescrever e perder drasticamente as capacidades gerais adquiridas durante o pré-treinamento (Howard; Ruder, 2018b). Por exemplo, um modelo ajustado para gerar código em Python pode esquecer como manter um diálogo coerente sobre história.
- **Risco de sobreajuste**: Especialmente quando o conjunto de dados de ajuste fino é pequeno ou pouco diverso, o modelo pode memorizar os exemplos de treinamento em vez de aprender a generalizar o padrão subjacente. Isso resulta em um excelente desempenho nos dados de validação, mas um baixo desempenho em dados novos e não vistos.
- **Amplificação de viés**: Se o conjunto de dados utilizado para o ajuste fino contiver vieses sociais ou demográficos, o processo de FFT pode não apenas replicá-los, mas também reforçá-los, resultando em um modelo final ainda mais enviesado do que sua versão pré-treinada. Por exemplo, um modelo pode ser induzido a associar automaticamente certas profissões a um gênero, reforçando estereótipos sociais.

Apesar de seus desafios, o FFT permanece uma ferramenta relevante e poderosa para cenários específicos. A decisão de utilizá-lo deve ser pragmática, considerando as seguintes condições:

1. **Quando o desempenho máximo é crítico**: Em aplicações de alto risco, como diagnóstico médico ou análise financeira, onde cada ponto percentual de acurácia é crucial e o orçamento permite o investimento.
2. **Quando há uma mudança substancial de domínio**: Se a tarefa alvo for radicalmente diferente dos dados de pré-treinamento, uma adaptação profunda pode ser necessária.

3. **Quando recursos e dados são abundantes**: O FFT só deve ser considerado quando se tem acesso a recursos computacionais massivos e, fundamentalmente, a um conjunto de dados de ajuste fino que seja grande, diverso e de altíssima qualidade para mitigar os riscos.

Em conclusão, o ajuste fino completo é uma faca de dois gumes. Ele oferece o caminho para o mais alto nível de especialização, mas a um custo extremo e com riscos significativos de degradar propriedades valiosas do modelo original. Para a grande maioria dos casos de uso práticos, o balanço entre desempenho, custo e risco pende favoravelmente para abordagens mais eficientes. Essas desvantagens impulsionaram a pesquisa e o desenvolvimento de uma nova família de técnicas, conhecidas como ajuste eficiente de parâmetros (do inglês, *Parameter-Efficient Fine-Tuning* ou PEFT).

3.3.2 PEFT

O desenvolvimento de metodologias voltadas à redução do custo computacional associado a LLMs tornou-se um requisito crítico para sua adaptação a tarefas especializadas. Essa demanda torna-se relevante no contexto do *fine-tuning* sobre conjuntos de dados de domínio específico, onde o custo computacional impacta diretamente a viabilidade prática da aplicação. Nesse cenário, busca-se empregar estratégias de treinamento que priorizem a atualização seletiva de camadas ou módulos do modelo, ao mesmo tempo minimizando o número de parâmetros ajustados. Tal abordagem tem o objetivo de não apenas reduzir o consumo de recursos e o tempo de processamento, mas também preservar, ou até aprimorar, a acurácia de predição visando garantir que a especialização do modelo não comprometa seu desempenho geral.

Neste contexto, surge uma abordagem denominada *Parameter-Efficient Fine-Tuning* (PEFT) (Ding et al., 2023), cujo objetivo é oferecer um conjunto de técnicas para adaptar grandes modelos de linguagem previamente pré-treinados, como o LLaMA (Gajulamandyam et al., 2025) ou BERT (Li, 2024), para tarefas mais específicas. Em vez de re-treinar integralmente o modelo, que pode conter bilhões de parâmetros e exigir recursos computacionais significativos, o PEFT se concentra em ajustar apenas uma pequena fração desses parâmetros. O objetivo primário consiste em manter o conhecimento geral que o modelo adquiriu durante seu treinamento inicial. O PEFT, então, age como uma "camada extra", adicionando um pequeno número de novos parâmetros treináveis. Esses novos parâmetros são ajustados para a tarefa específica, enquanto a maioria do modelo original permanece "congelada".

A motivação fundamental por trás do PEFT é a observação de que, durante o *fine-tuning*, nem todos os parâmetros de um modelo precisam ser atualizados para alcançar a melhor adaptação. Modelos pré-treinados são compostos por representações que generalizam o conhecimento linguístico. Neste sentido, apenas pequenos ajustes direcionais são necessários para especializar o modelo em domínios ou tarefas específicas. Este conceito levou ao desenvolvimento de técnicas que preservam o conhecimento original do modelo enquanto introduzem capacidades adaptativas através de estruturas paramétricas eficientes.

Técnicas de PEFT têm demonstrado melhorias significativas no que tange à eficácia dos novos modelos desenvolvidos, tanto do ponto de análise quantitativa quanto da avaliação qualitativa dos resultados. Estes avanços indicam a consolidação de tais técnicas no contexto de adaptação de modelos em diversos cenários, permitindo uma redução que abrange parcelas minúsculas dos parâmetros do modelo original, enquanto mantém performance similar com o *fine-tuning* completo. Li (Li, 2024), por exemplo, apresentou um refinamento

seletivo de 0,2% dos parâmetros de um modelo BERT usando uma abordagem baseada na atualização dos parâmetros de *bias* e normalização, demonstrando resultados competitivos com técnicas consolidadas na literatura.

3.3.2.1 Low-Ranking Adaptation

No contexto do PEFT, a técnica de *Low-Rank Adaptation* (LoRA) (Hu et al., 2021) emergiu como uma das mais conhecidas e amplamente utilizadas. Seu funcionamento consiste em aplicar uma decomposição matemática para atualizar os pesos durante o processo de *fine-tuning*. Quando o *fine-tuning* tradicional é aplicado, cada matriz de peso do modelo original, W_0, é atualizada para $W_0 + \Delta W$. No entanto, o LoRA reformula essa atualização ΔW como o produto de duas matrizes menores:

$$\Delta W = BA, \tag{3.1}$$

onde B e A são matrizes de baixo *rank*, onde B possui dimensões $d \times r$ e A possui dimensões $r \times k$, sendo d a dimensão da matriz original de pesos. Esta abordagem é fundamentada no princípio de que as mudanças necessárias para adaptar um modelo a uma nova tarefa podem ser representadas de forma eficiente em um espaço de menor dimensão.

Durante o processo de treinamento com LoRA, as matrizes de peso originais, W_0, são mantidas completamente congeladas para preservar o conhecimento adquirido durante o pré-treinamento. As novas matrizes A e B são inicializadas de forma que seu produto BA seja inicialmente zero. Isso assegura que o modelo, no início do treinamento, seja idêntico ao modelo pré-treinado original. Conforme o treinamento avança, as matrizes A e B são as únicas que aprendem as adaptações necessárias para a tarefa específica. A saída de cada camada é então calculada como $h = W_0x + BAx$, onde o primeiro termo mantém o comportamento original do modelo e o segundo adiciona a especialização aprendida.

O parâmetro de *rank*, r, é um hiperparâmetro essencial no LoRA, que governa o equilíbrio entre eficiência e capacidade expressiva do modelo. *Ranks* menores resultam em maior eficiência computacional e menor uso de memória, mas podem limitar a capacidade de adaptação do modelo. Por outro lado, *ranks* maiores aumentam a flexibilidade, mas também elevam o custo computacional.

3.3.2.2 Quantized Low-Rank Adaptation

A combinação de quantização e LoRA é uma técnica avançada e extremamente eficiente para otimizar o ajuste fino de LLMs. A quantização é o processo de converter os pesos e ativações de uma rede neural de um formato de alta precisão (como ponto flutuante de 32 bits - **float32**) para um formato de baixa precisão (como inteiros de 4 ou 8 bits - **int8** ou **int32**). A principal motivação para isso é a economia de memória, já que um **float32** ocupa 4 bytes, enquanto um **int8** ocupa apenas 1 byte. Apesar de a diferença de memória na quantização de um único parâmetro ser pequena, ela se torna muito significativa quando aplicada a modelos de bilhões de parâmetros. Além de economizar memória, a quantização também acelera os cálculos, pois processadores e GPUs podem realizar operações com inteiros de forma muito mais rápida e eficiente. Contudo, o principal desafio dessa abordagem é a possível perda de precisão, que pode degradar o desempenho do modelo se não for tratada com cautela.

A técnica de *Quantization* LoRA (Dettmers et al., 2023), denominada QLoRA, fornece uma solução para o desafio da perda de precisão ao aplicar a quantização seletivamente.

O processo é dividido basicamente em duas etapas. Primeiro, o modelo pré-treinado original é quantizado para um formato de baixa precisão, usando uma técnica denominada **NormalFloat** de 4 bits. Todos os parâmetros são convertidos e armazenados em menos bits, reduzindo o modelo de dezenas de *gigabytes* para apenas alguns.

Em seguida, o processo de *fine-tuning* é realizado apenas nas matrizes de baixo *rank* do LoRA, que são mantidas em um formato de alta precisão, ou seja, em f*loats* de 16 ou 32 bits. Esses poucos parâmetros em alta precisão representam o ponto chave para evitar a degradação de desempenho, pois o otimizador precisa de precisão para calcular os gradientes e atualizar os pesos de forma eficaz. Se as matrizes LoRA fossem quantizadas, o processo de aprendizado seria instável e ineficaz.

Mais especificamente, o QLoRA é baseado nas seguintes técnicas:

1. **NormalFloat 4 (NF4)**: A maioria dos pesos em uma rede neural profunda segue uma distribuição próxima de uma gaussiana ou normal, onde a maioria dos valores se concentra perto de zero. O NF4 define 16 pontos de quantização (já que $2^4 = 16$) de forma não uniforme, alocando mais pontos de precisão para os valores próximos de zero. Isso significa que os valores mais comuns e críticos do modelo são representados com maior precisão, enquanto os valores extremos, que são menos frequentes, são representados com menor precisão. O resultado é uma perda de informação muito menor em comparação com a quantização uniforme.

2. ***Double Quantization***: Executa a quantização das constantes de quantização NF4 que auxiliam na reconstrução dos valores originais. Essa segunda etapa de quantização visa aplicar uma segunda redução do consumo de memória da rede, aplicando uma quantificação de ponto flutuante de 32 bits dessas constantes para 8 bits.

3. ***Paged Optimizers***: é uma técnica de gerenciamento de memória cuja função é evitar os picos de uso de VRAM (memória da GPU) que ocorrem durante o treinamento, especialmente para armazenar parâmetros como gradiente, o momento do gradiente e a variância. Para solucionar esse problema, os *Paged Optimizers* alocam o estado do otimizador na memória da CPU. Quando a GPU precisa de espaço específico desses dados para o cálculo de gradientes, ela o aloca na VRAM. Após isso, esse espaço é realocado de volta para a CPU. Essa troca dinâmica entre a memória da CPU e da GPU garante que o treinamento continue de forma estável, sem ultrapassar o limite de VRAM.

Na prática, durante o treinamento, o fluxo de dados em uma camada com LoRA quantizado funciona da seguinte maneira: a entrada de um *token*, $\boldsymbol{x}$, é multiplicada pela matriz de peso quantizada do modelo base, W_0^{int8}. Em paralelo, essa mesma entrada é multiplicada pelas matrizes de alta precisão do LoRA, A e B. O resultado do cálculo com LoRA, isto é, BAx, é então somado ao resultado do cálculo com o modelo base (W_0^{int8}), e tudo é feito em alta precisão antes de ser passado para a próxima camada.

A principal vantagem reside no fato de que a computação de maior custo, que envolve a maior parte dos parâmetros (W_0^{int8}), é feito de forma rápida e com pouca memória. O cálculo com os parâmetros do LoRA, que é a parte que realmente "aprende', é mantido em alta precisão para garantir a qualidade do *fine-tuning* e evitar perda de precisão no cálculo dos gradientes. Como resultado, a combinação de *quantization* e LoRA preserva o desempenho do modelo, mas de maneira mais eficiente no que concerne ao uso de memória.

3.3.2.3 Módulos adapters

Outra técnica que se destaca no contexto de PEFT são os módulos *Adapters*, propostos por Houlsby et al. (Houlsby et al., 2019), cuja estrutura é formada por pequenos módulos de rede neural que são incorporados nas camadas de um modelo *Transformer* pré-treinado. O princípio consiste em manter fixos os parâmetros do modelo base e treinar apenas os parâmetros desses novos módulos adicionados. Essa abordagem permite que o modelo original preserve o conhecimento já adquirido durante o pré-treinamento, enquanto os *Adapters* aprendem as adaptações específicas para a nova tarefa durante o ajuste fino do modelo.

A arquitetura de um módulo *Adapter*, em sua forma original, segue o padrão de um gargalo (*bottleneck*). Essa estrutura consiste em:

1. ***Down-projection*:** A entrada de uma camada do modelo original (com dimensão d) é projetada para uma dimensão menor, m, usando uma matriz de peso W_{down}. Isso cria um "gargalo" na arquitetura para comprimir a representação do dado.

2. ***Up-projection*:** A representação comprimida (h_{down}) é então projetada de volta para a dimensão original d usando uma segunda matriz de peso W_{up}, dando origem à uma representação de dimensão original h_{up}.

O valor da dimensão m é um hiperparâmetro, e geralmente é menor que d. Isso faz com que o número de parâmetros nos *Adapters* seja menor que o número de parâmetros do modelo original.

Para garantir a estabilidade do treinamento, é comum que a saída do módulo *Adapter* seja somada à entrada da camada, seguindo uma conexão residual (*skip-connection*). A saída final da camada com o *Adapter* é então representada por:

$$h_{final} = \mathbf{x} + h_{up}, \tag{3.2}$$

onde $\mathbf{x}$ é a entrada original do *Adapter*. A matriz W_{up} é geralmente inicializada com valores próximos de zero, para que o módulo *Adapter* comece com uma função de identidade. Isso garante que, no início do treinamento, a saída do *Adapter* seja zero e o comportamento do modelo seja idêntico ao original.

Em suma, as técnicas de PEFT consolidaram-se no contexto de ajuste fino de grandes modelos de linguagens ao oferecerem uma alternativa com a introdução de módulos com pouco impacto em suas arquiteturas originais. Entretanto, tais técnicas possuem suas próprias limitações, sendo a principal delas a de não poder replicar completamente o desempenho do *fine-tuning* tradicional em todos os cenários. Em tarefas que exigem uma adaptação mais complexa e mudanças significativas no modelo, o ajuste completo de todos os parâmetros ainda pode ser a abordagem mais eficaz.

3.3.3 *Few-shot learning*

Complementar às estratégias de ajuste fino completo e às técnicas de PEFT, as abordagens de aprendizagem com poucas amostras (do inglês, *few-shot learning*) buscam maximizar ainda mais a eficiência computacional, reduzindo a quantidade de exemplos necessários para a especialização de um modelo. No campo do Processamento de Linguagem Natural, este termo abrange duas metodologias distintas para adaptar modelos a novas tarefas, nomeadamente *In-Context Learning* e Ajuste Fino com Poucas Amostras. Embora ambas

partam da premissa de utilizar uma quantidade mínima de exemplos, elas diferem fundamentalmente no mecanismo de aprendizado, onde uma ocorre sem a atualização dos pesos do modelo, enquanto a outra envolve um ajuste fino deliberado.

3.3.3.1 Aprendizagem no contexto (*In-Context Learning*)

A primeira abordagem, e a que mais popularizou o termo no contexto de PLN. Este método não envolve uma fase de treinamento com atualização de gradientes. Em vez disso, o modelo é condicionado a realizar uma nova tarefa no momento da inferência. O aprendizado é transitório e guiado pelo *prompt* de entrada, que contém uma descrição da tarefa e um pequeno númeo de exemplos ilustrativos (os *shots*).

O trabalho de demonstrou que modelos de grande escala possuem capacidades emergentes de generalização a partir de poucos exemplos, viabilizando a aprendizagem no contexto de forma eficaz. Os autores avaliam três configurações:

- ***Zero-shot learning*:** O modelo recebe apenas a descrição em linguagem natural da tarefa, sem nenhum exemplo de como executá-la.
- ***One-shot learning*:** O modelo recebe a descrição da tarefa acompanhada de um único exemplo.
- ***Few-shot learning*:** O modelo recebe a descrição da tarefa e múltiplos exemplos (tipicamente entre 2 e 32).

A eficácia desta técnica está intrinsecamente ligada à qualidade e ao formato do *prompt*. A engenharia de *prompts* torna-se uma disciplina crucial, pois a maneira como os exemplos são apresentados pode influenciar drasticamente a precisão e a coerência da saída do modelo. A seleção de exemplos diversificados e representativos da tarefa em questão é fundamental para guiar o modelo de forma adequada.

A principal vantagem desta abordagem reside na sua eficiência computacional e na economia de recursos. Como não há treinamento, os custos associados à esta etapa são completamente eliminados. No entanto, a performance embora muitas vezes competitiva, pode não alcançar a mesma robustez de um modelo que passou pelo processo de atualização de parâmetros, especialmente em tarefas altamente especializadas ou que requerem conhecimento de um domínio muito específico. Além disso, a janela de contexto do modelo impõe um limite físico à quantidade de exemplos que podem ser fornecidos, e o custo da inferência pode aumentar.

3.3.3.2 Ajuste fino com poucas amostras

A segunda abordagem alinha-se mais com a definição clássica de *few-shot learning* em outras áreas de IA. Ela consiste em realizar o ajuste fino supervisionado do modelo utilizando um conjunto de dados deliberadamente pequeno, mas de alta qualidade. A hipótese central é que, para modelos que já possuem um vasto conhecimento de mundo adquirido durante o pré-treinamento, não é a quantidade de dados de ajuste fino que importa, mas sim a sua qualidade.

Um exemplo proeminente desta técnica é apresentado no trabalho de (Ye et al., 2025). Os autores exploram a premissa de que o raciocínio complexo exige volumes massivos de dados de treinamento. Eles propõem a hipótese LIMO (do inglês, *Less-Is-More Reasoning Hypothesis*) que postula que, em modelos onde o conhecimento de um domínio já foi

extensivamente codificado o raciocínio sofisticado pode emergir a partir de demonstrações mínimas, mas estrategicamente desenhadas, de processos cognitivos. Segundo a hipótese, existem dois fatores principais que determinam o ponto de ativação para o raciocínio complexo: (1) a presença de conhecimento prévio dentro dos parâmetros do modelo, adquirido durante o pré-treinamento e (2) a efetividade de poucos exemplos em demonstrar processos de solução de problemas que estimulem uma reflexão mais extensa.

Para validar essa hipótese, os autores desenvolvem um rigoroso processo de curadoria de dados. Partindo de um conjunto com dezenas de milhões de problemas matemáticos, eles aplicam um sistema de filtragem em múltiplos estágios para selecionar apenas 800 amostras de treinamento de alta qualidade. Os critérios de seleção priorizaram problemas desafiadores e cadeias de raciocínio que exibiam características como elaboração detalhada, autoverificação e clareza lógica. Utilizando apenas 800 amostras para realizar um ajuste fino supervisionado simples o modelo alcançou resultados notáveis e amplamente superiores a outros modelos, tanto dentro quanto fora do domínio, confirmando que, para certas tarefas, a curadoria meticulosa de um pequeno conjunto de dados é mais crucial do que a quantidade massiva, permitindo o desenvolvimento de capacidades de raciocínio robustas de forma muito mais eficiente

Em suma, as duas vertentes do *few-shot learning* oferecem um contraponto claro em termos de mecanismo e objetivo. A aprendizagem no contexto é uma técnica de inferência: ela não altera os pesos do modelo e depende da qualidade do *prompt* para guiar o comportamento do modelo de forma temporária. Por outro lado, o ajuste fino com poucas amostras é uma técnica de treinamento que modifica permanentemente o modelo. Porém, seu sucesso não está no volume, mas na qualidade e na densidade informacional dos dados. O investimento é deslocado da coleta massiva para a curadoria meticulosa, com o objetivo de alcançar um desempenho robusto e de ponta em domínios específicos.

3.4 *AI alignment*

O alinhamento de sistemas de IA aos valores humanos (do inglês, *AI Alignment*) refere-se ao desafio de garantir que agentes artificiais otimizem objetivos que estejam em conformidade com as intenções humanas, mesmo em contextos complexos, ambíguos ou não totalmente especificados. Esse problema é frequentemente enquadrado no contexto mais amplo do *controle de sistemas inteligentes*, conforme discutido por Russell (Russel, 2019) e Bostrom (Bostrom, 2014), e está no cerne das preocupações de segurança da IA contemporânea (Hendrycks et al., 2021; Ngo et al., 2023).

À medida que LLMs, agentes de decisão e sistemas autônomos adquirem maior expressividade e poder de generalização, torna-se cada vez mais crítico mitigar riscos decorrentes de comportamentos inesperados, otimizações mal direcionadas ou divergências entre metas explícitas e consequências práticas. Trabalhos como Gabriel (Gabriel, 2020) destacam que o alinhamento não se restringe a uma questão técnica, mas envolve também aspectos normativos e éticos sobre quais valores humanos devem orientar a otimização algorítmica.

Ao contrário de tarefas com métricas bem definidas, como acurácia em classificação, muitos problemas reais envolvem critérios subjetivos, valores éticos e decisões com múltiplas dimensões. Nesses casos, especificar manualmente uma função de recompensa ou um conjunto de regras explicitamente alinhadas com os valores humanos é frequentemente impraticável. Assim, cresce o interesse por abordagens que incorporam diretamente o julgamento humano no processo de treinamento, não apenas como fonte de dados supervisionados, mas como sinal de alinhamento.

É útil esclarecer o que se entende por agentes artificiais. São sistemas de IA capazes de interagir com um ambiente e tomar decisões de forma autônoma, com base em objetivos explícitos ou implícitos. Eles operam em um ciclo de percepção, decisão e ação, recebendo observações do ambiente (por exemplo, entradas textuais ou sensores), executando ações (como gerar respostas, mover-se fisicamente ou chamar APIs) e otimizando comportamentos de acordo com critérios de desempenho. Exemplos incluem assistentes conversacionais interativos, robôs autônomos, sistemas de recomendação adaptativos ou LLMs integrados a agentes de planejamento e execução como o AutoGPT. À medida que o grau de autonomia e poder de ação desses agentes aumenta, cresce também o potencial de desalinhamento, tornando o *AI Alignment* uma prioridade crítica.

O tema ganhou relevância crescente à medida que sistemas de IA passaram a atuar de forma autônoma em cenários críticos, como decisões financeiras, diagnósticos médicos e controle de veículos autônomos, onde pequenos desalinhamentos entre a função de recompensa e os valores humanos podem gerar comportamentos perigosos ou não intencionais (Amodei et al., 2016; Carlsmith, 2022). Além disso, fenômenos como *goal misgeneralization* e *specification gaming* ilustram a dificuldade de definir objetivos de forma robusta (Krakovna et al., 2020; Shah et al., 2022).

O primeiro ocorre quando o modelo aprende corretamente o comportamento desejado durante o treinamento, mas generaliza a meta subjacente de forma incorreta em novos contextos, perseguindo intenções aparentes em vez de objetivos reais. Por exemplo, um agente treinado para coletar moedas em um jogo pode, em cenários novos, priorizar qualquer objeto circular, interpretando erroneamente o conceito de "moeda". Já o *specification gaming* refere-se à exploração de falhas na função de recompensa ou nas restrições impostas pelo projetista, levando o agente a atingir pontuações elevadas sem realizar a tarefa de maneira adequada. Um exemplo clássico é o de agentes que maximizam métricas intermediárias, como distância percorrida, sem cumprir o objetivo final de forma útil. Esses fenômenos reforçam a necessidade de métodos de alinhamento que incorporem o julgamento humano de forma mais direta e adaptativa.

Duas das abordagens mais representativas desse paradigma são o *Reinforcement Learning from Human Feedback* (RLHF) (Christiano et al., 2017; Leike et al., 2018; Ouyang et al., 2022; Stiennon et al., 2020b) e o *Direct Preference Optimization* (DPO) (Rafailov et al., 2023), descritos a seguir com maior detalhamento.

3.4.1 Fundamentos de aprendizado por reforço

Antes de descrever RLHF e DPO, é importante revisar brevemente conceitos fundamentais do aprendizado por reforço (RL). O RL modela a interação entre um agente e um ambiente como um processo de decisão de Markov (MDP), definido por um conjunto de estados $\mathcal{S}$, ações $\mathcal{A}$, uma função de transição $P(s' \mid s, a)$ e uma função de recompensa $r(s, a)$.

Uma política $\pi_\phi(a \mid s)$ descreve a probabilidade de o agente tomar a ação a no estado s, parametrizada por ϕ. O objetivo do RL é maximizar a recompensa esperada acumulada ao longo do tempo:

$$J(\pi_\phi) = \mathbb{E}_{\pi_\phi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \tag{3.3}$$

onde $\gamma \in [0, 1)$ é um fator de desconto temporal. Tradicionalmente, assume-se que a função de recompensa $r(s, a)$ é totalmente especificada pelo projetista do sistema. No

entanto, em tarefas abertas e de alto nível, como diálogo natural, decisões éticas ou planejamento estratégico, definir manualmente $r(s, a)$ de forma alinhada aos valores humanos é virtualmente impossível. Essa limitação motiva métodos que estimam recompensas indiretamente a partir de preferências humanas ou substituem sua modelagem explícita por técnicas baseadas em sinais comportamentais observados.

3.4.2 *Reinforcement Learning from Human Feedback* (RLHF)

O RLHF (Christiano et al., 2017; Leike et al., 2018; Ouyang et al., 2022) é um procedimento de alinhamento que modela o problema como uma tarefa de controle sequencial, na qual um agente otimiza uma política $\pi_\phi(y \mid x)$, a probabilidade de gerar uma resposta y dado um *prompt* x, com o objetivo de maximizar uma função de recompensa derivada de preferências humanas.

3.4.2.1 Modelo de recompensa

Dado um conjunto de comparações humanas entre respostas y^+ (preferida) e y^- (rejeitada) para um mesmo x, o objetivo é aprender uma função de recompensa $r_\theta(x, y)$ tal que:

$$P(y^+ \succ y^-) = \frac{\exp(r_\theta(x, y^+))}{\exp(r_\theta(x, y^+)) + \exp(r_\theta(x, y^-))} \tag{3.4}$$

A função de perda associada é a entropia cruzada negativa entre as preferências humanas e a distribuição induzida por r_θ:

$$\mathcal{L}_{\text{RM}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)} \left[\log \frac{\exp(r_\theta(x, y^+))}{\exp(r_\theta(x, y^+)) + \exp(r_\theta(x, y^-))} \right] \tag{3.5}$$

3.4.2.2 Aprendizado por reforço

Com r_θ fixado, ajusta-se a política π_ϕ para maximizar a recompensa esperada:

$$\max_\phi \quad \mathbb{E}_{x\sim\mathcal{D}} \left[\mathbb{E}_{y\sim\pi_\phi(\cdot|x)} \left[r_\theta(x, y) \right] \right] \tag{3.6}$$

Para garantir estabilidade e evitar *drift*, impõe-se uma penalidade de divergência Kullback-Leibler (KL) em relação à política original π_0:

$$\mathcal{L}_{\text{RLHF}}(\phi) = -\mathbb{E}_{x\sim\mathcal{D}} \left[\mathbb{E}_{y\sim\pi_\phi(y|x)} \left[r_\theta(x, y) - \beta \cdot \log \frac{\pi_\phi(y \mid x)}{\pi_0(y \mid x)} \right] \right] \tag{3.7}$$

A função da Equação 3.6 pode ser reinterpretada como a maximização de uma vantagem regularizada:

$$\mathcal{L}_{\text{RLHF}}(\phi) = -\mathbb{E}_{\pi_\phi} \left[A^{r_\theta}(x, y) - \beta D_{\text{KL}}(\pi_\phi \parallel \pi_0) \right]$$

onde $A^{r_\theta}(x, y)$ representa a vantagem induzida pelo modelo de recompensa. Essa formulação evidencia que o treinamento busca não apenas maximizar a recompensa, mas também restringir a divergência da política em relação ao comportamento original do modelo. Essa função é comumente otimizada via *Proximal Policy Optimization* (PPO) (Schulman et al., 2017), um algoritmo de aprendizado por reforço que atualiza a política por gradientes com restrições de divergência KL, garantindo que mudanças bruscas não degradem o desempenho.

3.4.2.3 Discussão sobre instabilidades

A separação entre a estimativa de preferências (r_θ) e a otimização da política (π_ϕ) pode introduzir instabilidades devido ao fenômeno conhecido como *reward hacking*. Quando o modelo de recompensa é imperfeito, a política pode explorar regiões do espaço de respostas onde r_θ é superestimado, produzindo comportamentos desalinhados com as intenções humanas. Além disso, a escolha do coeficiente β na Eq. (6) afeta a regularização: valores baixos permitem derivações excessivas da política, enquanto valores altos podem restringir a exploração, levando à subotimização.

3.4.3 *Direct Preference Optimization* (DPO)

O DPO (Rafailov et al., 2023) propõe um modelo mais direto e principiado para treinamento de políticas alinhadas, evitando a estimativa explícita de uma função de recompensa. A ideia central é modelar as preferências humanas como uma distribuição *a posteriori* sobre políticas, baseada na inferência bayesiana.

3.4.3.1 Formulação bayesiana

Assume-se que os dados de preferência $y^+ \succ y^-$ são amostrados de uma distribuição preferencial induzida por uma política ótima π^*, tal que:

$$P(y^+ \succ y^- \mid x) = \frac{\exp(\beta \cdot \log \pi^*(y^+ \mid x))}{\exp(\beta \cdot \log \pi^*(y^+ \mid x)) + \exp(\beta \cdot \log \pi^*(y^- \mid x))} \tag{3.8}$$

onde $\beta > 0$ controla a rigidez da preferência. Ao aproximar $\pi^* \approx \pi_\phi$, pode-se obter uma função de perda supervisionada para o aprendizado direto da política:

$$\mathcal{L}_{\text{DPO}}(\phi) = -\mathbb{E}_{(x,y^+,y^-)} \left[\log \frac{\exp(\beta \cdot \Delta \log \pi_\phi)}{\exp(\beta \cdot \Delta \log \pi_\phi) + 1} \right] \tag{3.9}$$

com $\Delta \log \pi_\phi = \log \pi_\phi(y^+ \mid x) - \log \pi_\phi(y^- \mid x)$.

A função de perda da Eq. (8) é análoga à perda logística em tarefas de classificação binária, onde o modelo aprende a distinguir respostas preferidas de rejeitadas. Sob essa perspectiva, o DPO pode ser interpretado como uma forma de inferência bayesiana aproximada, na qual a política ótima é tratada como distribuição posterior sobre ações condicionada às preferências humanas observadas.

3.4.3.2 Relação com aprendizado por reforço

A perda DPO pode ser interpretada como uma aproximação supervisionada do gradiente de política que surge da maximização da preferência empírica. Em vez de utilizar amostragem ou atualização por episódios, o DPO permite otimização por *backpropagation* puro, pois todas as quantidades relevantes são diretamente diferenciáveis.

3.4.3.3 Vantagens e limitações

Dentre as principais vantagens do DPO, temos:

- **Eficiência computacional:** não requer RL nem estimativa de r_θ;
- **Estabilidade:** treinamento supervisionado direto evita instabilidades de PPO;

- **Simplicidade:** menos componentes, menos fontes de erro, maior interpretabilidade.

Apesar de seus méritos, o DPO depende fortemente da fidelidade das distribuições de preferência observadas. Essa dependência significa que a política tende a reproduzir fielmente os padrões nos dados disponíveis, mas pode falhar em capturar estruturas de longo prazo que só emergem após várias etapas de decisão. Um modelo de recompensa explícito, ao inferir padrões latentes e recompensas cumulativas, pode capturar dependências temporais e objetivos hierárquicos que o DPO, por sua natureza supervisionada, tende a ignorar.

3.4.4 Discussão comparativa: RLHF vs. DPO

Ambas as abordagens compartilham o objetivo de incorporar preferências humanas ao treinamento de políticas, mas diferem em aspectos fundamentais. O RLHF separa o aprendizado da função de recompensa do processo de otimização, o que oferece maior flexibilidade e capacidade de generalização, porém introduz complexidade e risco de instabilidades devido ao *reward hacking*. O DPO, por outro lado, adota uma formulação mais direta e eficiente, reduzindo o custo computacional e a instabilidade, ao preço de depender fortemente da qualidade dos dados de preferência. Assim, o RLHF tende a ser mais adequado para cenários com dados escassos, mas onde a generalização é essencial, enquanto o DPO se mostra vantajoso quando há abundância de *feedback* humano confiável e rápido treinamento é desejável. Revisões abrangentes recentes (Hendrycks et al., 2021; Ngo et al., 2023) destacam que abordagens híbridas e multiestágio, combinando aprendizado por preferências, interpretabilidade e segurança formal, representam um caminho promissor para sistemas de IA alinhados e robustos. O Quadro 3.1 resume comparativamente as principais diferenças entre o RLHF e o DPO, destacando seus fundamentos teóricos, características de treinamento e limitações práticas.

Quadro 3.1: Comparação entre *Reinforcement Learning from Human Feedback* (RLHF) e *Direct Preference Optimization* (DPO)

Aspecto	**RLHF**	**DPO**
Formulação	Baseado em aprendizado por reforço com modelagem explícita de recompensa e atualização da política via PPO (Christiano et al., 2017; Ouyang et al., 2022; Schulman et al., 2017).	Baseado em otimização supervisionada direta de preferências humanas, sem estimar explicitamente a função de recompensa (Rafailov et al., 2023).
Objetivo de treinamento	Maximizar a recompensa aprendida $r_\theta(x, y)$ regularizada pela divergência KL entre políticas.	Maximizar diretamente a probabilidade de respostas preferidas y^+ sobre rejeitadas y^- em uma formulação logística.
Tipo de atualização	Aprendizado por reforço com amostragem de trajetórias e gradientes de política (*policy gradients*).	Treinamento supervisionado via *backpropagation* padrão, sem interação com o ambiente.

Custo computacional	Alto, devido à necessidade de amostragem, atualização iterativa da política e estimativa de recompensas.	Baixo, pois elimina o ciclo de RL e opera inteiramente em dados estáticos de preferências.
Estabilidade de treinamento	Sensível ao ajuste do coeficiente de regularização β e à qualidade do modelo de recompensa, sujeito a *reward hacking*.	Mais estável, pois não requer modelagem de recompensas nem otimização sequencial via PPO.
Interpretação teórica	Aproximação regularizada do gradiente de vantagem em um MDP com recompensa aprendida.	Inferência bayesiana aproximada sobre políticas condicionadas a preferências humanas.
Generalização	Pode extrapolar preferências para novos contextos, desde que o modelo de recompensa capture relações latentes.	Limitado à distribuição observada de preferências, podendo falhar em extrapolações de longo prazo.
Aplicações típicas	Treinamento de LLMs (por exemplo, InstructGPT) e agentes de decisão com *feedback* humano em tempo real.	Ajuste fino supervisionado de LLMs com grandes conjuntos de preferências curadas (*offline alignment*).
Principais limitações	Complexidade computacional, dependência de modelo de recompensa robusto e risco de instabilidade.	Dependência da qualidade das preferências e dificuldade em capturar dependências temporais complexas.

3.5 Exemplo prático

Nesta seção, materializamos os conceitos teóricos discutidos ao longo do capítulo em uma aplicação prática de ponta a ponta. O objetivo é demonstrar o processo de especialização de um grande modelo de linguagem para uma tarefa de nicho no idioma português. Para isso, vamos criar o Dr. Bode, um assistente virtual de saúde, partindo de um modelo generalista e aplicando as técnicas de ajuste fino supervisionado (com PEFT) e otimização por preferência direta para adaptá-lo a um domínio altamente especializado, como a medicina.

3.5.1 Preparando o ambiente

A reprodutibilidade é um pilar da pesquisa e desenvolvimento em IA. Para garantir a consistência dos resultados, nosso ambiente de trabalho deve ser configurado corretamente. Este guia foi desenvolvido e testado com `Python 3.9`. As bibliotecas e suas respectivas versões são listadas abaixo. Destacamos as principais: `transformers` e `datasets` para o ecossistema *Hugging Face*; `peft` para a aplicação de *Parameter-Efficient Fine-Tuning*; `trl` para os treinadores de alto nível (SFT e DPO); e `bitsandbytes` para a quantização do modelo.

```
transformers==4.53.1
torch==2.7.1
datasets==3.6.0
tqdm==4.67.1
```

```
peft==0.16.0
accelerate==1.8.1
trl==0.9.6
bitsandbytes==0.41.1
```

Ainda, definimos as seguintes variáveis globais, utilizadas durante a execução das células a seguir

```
NOME_MODELO = "Qwen/Qwen3-1.7B"
DATASET_AVALIACAO = "recogna-nlp/drbodebench"
DATASET_SFT = "recogna-nlp/drbode240_dataset"
DATASET_DPO = "recogna-nlp/dpo-mmlu-ptbr"
```

3.5.2 O ponto de partida

Todo projeto de ajuste fino começa com a escolha de um modelo de propósito geral. Para este estudo de caso, nosso ponto de partida será o `Qwen/Qwen3-1.7B`, um modelo com 1.7 bilhão de parâmetros. Este modelo foi escolhido por seu sólido desempenho em tarefas de raciocínio e seguimento de instruções, aliado a um tamanho que permite o treinamento em hardware acessível. Contudo, ele carece de conhecimento especializado em saúde, e nosso objetivo é justamente preencher essa lacuna. O treinamento será realizado com o *dataset* `recogna-nlp/drbode240_dataset`[1].

Antes de qualquer modificação, é crucial estabelecer uma linha de base (*baseline*) para quantificar nosso progresso. Avaliaremos o desempenho do modelo original no *benchmark* `recogna-nlp/drbodebench`[2], um conjunto composto por questões de múltipla escolha extraídas de provas de residência médica no Brasil. O resultado nos mostrará a capacidade inicial do modelo de responder a questões médicas complexas, servindo como ponto de partida para a jornada de especialização. O script a seguir realiza três etapas fundamentais:

- **Formatação do *prompt*:** Estrutura a questão no formato esperado pelo modelo.
- **Geração e extração da resposta:** Envia o *prompt* ao modelo e extrai a alternativa prevista.
- **Cálculo de acurácia:** Compara a resposta do modelo com o gabarito e calcula a acurácia total.

```
import re
import json
import time
import torch
from datasets import load_dataset
from transformers import AutoModelForCausalLM, AutoTokenizer
from tqdm import tqdm


def formatar_prompt_medico(questao: dict) -> str:
    """
    Cria um prompt formatado a partir de um item do dataset.
```

[1] https://huggingface.co/datasets/recogna-nlp/drbode240_dataset
[2] https://huggingface.co/datasets/recogna-nlp/drbodebench

```
    Args:
        questao (dict): Um dicionário contendo 'enunciado', 'alternativas'
                        e opcionalmente 'img_description'.

    Returns:
        str: O prompt completo e formatado, pronto para ser enviado ao modelo.
    """

    prompt = f"""Você é um especialista médico. Responda APENAS com a letra da
    alternativa correta (A, B, C, D ou E). Não escreva nada mais.

Questão:
{questao['enunciado']}

"""
    # Trata de forma segura a existência de imagens na questão.
    if questao.get("contains_img", False):
        prompt += f"Descrição da imagem: {questao.get('img_description', '')}\n\n"

    for letra, texto in questao["alternativas"].items():
        prompt += f"{letra}) {texto}\n"

    prompt += "\nResposta (apenas a letra):"
    return prompt.strip()


def obter_resposta_modelo(prompt: str, modelo: AutoModelForCausalLM,
    tokenizer: AutoTokenizer) -> str:
    """
    Gera uma resposta do modelo para um determinado prompt.

    Args:
        prompt (str): O prompt formatado.
        modelo (AutoModelForCausalLM): O modelo de linguagem carregado.
        tokenizer (AutoTokenizer): O tokenizer correspondente.

    Returns:
        str: A resposta textual gerada pelo modelo.
    """
    # O template de chat é específico de cada família de modelos.
    # Esta etapa garante que a entrada seja formatada como o modelo espera.
    messages = [{"role": "user", "content": prompt}]
    input_ids = tokenizer.apply_chat_template(
        messages, add_generation_prompt=True, return_tensors="pt",
        enable_thinking=False
    ).to(modelo.device)

    # Gera a resposta. `max_new_tokens` é baixo pois esperamos apenas uma letra.
    outputs = modelo.generate(
        input_ids,
        max_new_tokens=10,
        pad_token_id=tokenizer.eos_token_id
    )

    # Decodifica os tokens gerados, ignorando o prompt de entrada.
    resposta = tokenizer.decode(
        outputs[0][input_ids.shape[1] :],
        skip_special_tokens=True
```

```
    )

    return resposta.strip()


def extrair_alternativa(texto_resposta: str) -> str:
    """
    Extrai a primeira letra de alternativa (A-E) de um texto.

    Args:
        texto_resposta (str): O texto bruto gerado pelo modelo.

    Returns:
        str: A letra da alternativa em maiúsculo, ou uma string vazia se não for
        encontrada.
    """
    # A regex [A-Ea-e] busca a primeira ocorrência de uma das letras.
    match = re.search(r"[A-Ea-e]", texto_resposta)
    if match:
        return match.group(0).upper()
    return ""


def executar_avaliacao(modelo: AutoModelForCausalLM, tokenizer: AutoTokenizer,
    dataset: load_dataset, arquivo_saida: str):
    """
    Orquestra o processo de avaliação em todo o dataset.
    """

    start_time = time.time()
    resultados = []

    for item in tqdm(dataset, desc="Avaliando Questões"):
        prompt = formatar_prompt_medico(item)
        resposta_bruta = obter_resposta_modelo(prompt, modelo, tokenizer)
        predicao_processada = extrair_alternativa(resposta_bruta)

        resultados.append({
            "origem": item["origem"],
            "prova": item["prova"],
            "numero_questao": item["numero"],
            "enunciado": item["enunciado"],
            "resposta_correta": item["resposta"],
            "predicao_modelo": predicao_processada,
            "resposta_bruta_modelo": resposta_bruta,
        })


    # Cálculo de métricas e salvamento
    end_time = time.time()
    duracao_min = (end_time - start_time) / 60
    print(f"\nAvaliação finalizada em {duracao_min:.2f} minutos.")

    corretos = sum(1 for res in resultados if res["predicao_modelo"] ==
    res["resposta_correta"])
    total = len(resultados)
    acuracia = (corretos / total) * 100
    print(f"\nAcurácia final: {acuracia:.2f}% ({corretos}/{total})")
```

```
    with open(arquivo_saida, "w", encoding="utf-8") as f:
        json.dump(resultados, f, ensure_ascii=False, indent=4)
    print(f"Resultados salvos em: {arquivo_saida}")


# Carregando modelo e tokenizer
tokenizer = AutoTokenizer.from_pretrained(
    NOME_MODELO,
    trust_remote_code=True
)
model = AutoModelForCausalLM.from_pretrained(
    NOME_MODELO,
    trust_remote_code=True,
    torch_dtype=torch.bfloat16,
    device_map="auto"
)

# Carregando dataset de avaliação
eval_dataset = load_dataset(DATASET_AVALIACAO, split="train")

# Executar a avaliação
output_file = "resultado_modelo_base.json"
executar_avaliacao(model, tokenizer, eval_dataset, output_file)
```

Após a avaliação, o modelo base acerta 281 de 1301 questões, resultando em uma acurácia de 21,60%. Este valor, pouco acima de uma escolha aleatória (20% para 5 alternativas), confirma a ausência de conhecimento médico especializado e estabelece nosso ponto de partida.

3.5.3 Ajuste fino supervisionado com LoRA

Com a linha de base estabelecida, o próximo passo é ensinar ao modelo o conhecimento específico do domínio proposto. Utilizaremos a técnica de ajuste fino supervisionado, aplicando QLoRA para otimizar o uso de memória e reduzir os requisitos computacionais. Para o treinamento, seguiremos os passos de (1) carregar o modelo base quantizado, (2) configurar os adaptadores LoRA e (3) executar o treinamento com o `SFTTrainer`.

- **Configuração e carregamento do modelo:** Primeiro, configuramos a quantização e carregamos o modelo base. A quantização de 4 bits, gerenciada pela biblioteca `bitsandbytes`, é o que nos permite carregar um modelo de bilhões de parâmetros em uma GPU com VRAM limitada. A biblioteca `accelerate` auxilia no mapeamento do modelo para o dispositivo correto de forma automática.
- **Configuração do LoRA**: Definimos a configuração do LoRA usando a biblioteca `peft`. Em vez de treinar todos os 1.7 bilhão de parâmetros, vamos inserir adaptadores em camadas específicas do modelo. Apenas esses adaptadores, que representam uma pequena fração do total de parâmetros, serão treinados, resultando em uma economia massiva de recursos.
- **Treinamento com SFTTrainer:** Com o modelo preparado, utilizamos o `SFTTrainer` da biblioteca `trl`. Esta classe é um facilitador de alto nível que abstrai a complexidade do loop de treinamento. Ela lida com a formatação dos dados, o processamento em lotes e a otimização do treinamento de forma eficiente, exigindo apenas a configuração dos hiperparâmetros através da classe `SFTConfig`.

```
import os

import torch
from accelerate import PartialState
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
from datasets import load_dataset
from trl import SFTConfig, SFTTrainer
```

```
def encontrar_modulos_lineares(modelo: AutoModelForCausalLM) -> list:
    """
    Encontra todos os módulos lineares no modelo para aplicar o LoRA.

    Esta função auxiliar percorre todas as camadas do modelo e identifica aquelas
    que são instâncias de `torch.nn.Linear`, que são os alvos ideais para
    a inserção dos adaptadores LoRA. A camada 'lm_head' é geralmente excluída
    porque sua adaptação pode levar a instabilidade no treinamento.

    Args:
        modelo (AutoModelForCausalLM): O modelo a ser inspecionado.

    Returns:
        list: Uma lista de nomes dos módulos lineares alvo.
    """
    cls = torch.nn.Linear
    modulos_lora = set()
    for nome, modulo in modelo.named_modules():
        if isinstance(modulo, cls):
            nomes = nome.split(".")
            modulos_lora.add(nomes[0] if len(nomes) == 1 else nomes[-1])

    if "lm_head" in modulos_lora:
        modulos_lora.remove("lm_head")

    return list(modulos_lora)


def preprocessar_dataset(exemplo):
    """
    Formata uma entrada do dataset para o formato de texto único esperado pelo
    SFTTrainer.
    """
    partes_texto = []
    for entrada in exemplo["data"]:
        partes_texto.append(f"{entrada['role'].capitalize()}: {entrada['content']}")
    return {"data": "\n".join(partes_texto)}


# Determina o dispositivo para treinamento distribuído, se aplicável
string_dispositivo = PartialState().process_index

# Configuração da Quantização (QLoRA)
# QLoRA reduz o uso de memória ao carregar o modelo com pesos de 4 bits.
config_quantizacao = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",  # Tipo de quantização (Normal Float 4)
    bnb_4bit_compute_dtype=torch.bfloat16,  # Tipo de dado para os cálculos
)

# Carregamento do modelo com a configuração de quantização
modelo_base = AutoModelForCausalLM.from_pretrained(
    NOME_MODELO,
    torch_dtype=torch.bfloat16,
    trust_remote_code=True,
    quantization_config=config_quantizacao,
    device_map={"": string_dispositivo},
)
```

```
# Desativa o cache para treinamento
modelo_base.config.use_cache = False
modelo_base = prepare_model_for_kbit_training(modelo_base)

# Carregamento do tokenizer
tokenizer = AutoTokenizer.from_pretrained(
    NOME_MODELO,
    trust_remote_code=True
)
tokenizer.pad_token = tokenizer.eos_token  # Token de padding como fim de sentença.
tokenizer.padding_side = "right" # Garante que o padding seja adicionado à direita.


# Configuracao do PEFT (LoRA)
# LoRA congela os pesos do modelo original e injeta
# matrizes treináveis (adapters) em camadas específicas.
peft_config = LoraConfig(
    r=128,  # O posto (rank) das matrizes de adaptação.
    lora_alpha=256,  # Fator de escala para os pesos do LoRA.
    target_modules=encontrar_modulos_lineares(modelo_base),  # Aplica LoRA em
                                                             #todas as camadas lineares.
    lora_dropout=0.05,  # Dropout para regularização dos adaptadores LoRA.
    bias="none",  # Não treina os vieses das camadas.
    task_type="CAUSAL_LM", # Especifica a tarefa como modelagem de linguagem causal.
)

# Aplica o wrapper PEFT para preparar o modelo para o treinamento LoRA.
modelo_peft = get_peft_model(modelo_base, peft_config)

# Carregamento do dataset
sft_dataset = load_dataset(DATASET_SFT, split="train")
sft_dataset = sft_dataset.map(preprocessar_dataset, remove_columns=["data"])

# Configuração do treinamento
# A classe SFTConfig armazena todos os hiperparâmetros para o SFTTrainer
output_dir_sft = "./modelo_sft"
argumentos_treinamento = SFTConfig(
    per_device_train_batch_size=8,
    gradient_accumulation_steps=4,
    gradient_checkpointing=True,
    max_grad_norm=0.3,
    num_train_epochs=1,
    learning_rate=2e-5,
    bf16=True,
    save_total_limit=3,
    logging_steps=1,
    output_dir=output_dir_sft,
    optim="paged_adamw_32bit",
    lr_scheduler_type="cosine",
    overwrite_output_dir=True,
    warmup_ratio=0.05,
    dataset_text_field="data",
)

# Instanciação e execução do treinador
treinador = SFTTrainer(
    model=modelo_peft,
    train_dataset=sft_dataset,
```

```
    peft_config=peft_config,
    args=argumentos_treinamento,
    processing_class=tokenizer,
)
treinador.train()

# Salva apenas os pesos dos adaptadores LoRA, que é um arquivo muito pequeno.
diretorio_checkpoint_final = os.path.join(output_dir_sft, "final_checkpoint")
treinador.model.save_pretrained(diretorio_checkpoint_final)
tokenizer.save_pretrained(diretorio_checkpoint_final)
```

Após o treinamento por uma única época, realizamos uma nova avaliação no *benchmark*. O modelo ajustado responde corretamente a 325 questões, alcançando uma acurácia de 24,98%, um aumento absoluto de 3,38 pontos percentuais. O ganho, embora positivo, é modesto. Isso ilustra um ponto importante: o SFT é altamente dependente da qualidade, diversidade e volume dos dados de treinamento. Fatores como o tamanho limitado do conjunto e o treinamento por uma única época podem explicar a magnitude do ganho. Para melhorias mais substanciais, seria necessário um conjunto de dados mais extenso, de maior qualidade ou treinando por múltiplas épocas de treinamento.

3.5.4 Refinamento do comportamento com DPO

Nosso modelo agora possui conhecimento especializado, adquirido durante o ajuste fino supervisionado. O próximo passo é refinar seu comportamento para que ele não apenas gere respostas corretas, mas também prefira um estilo de resposta a outro. O objetivo do DPO não é primariamente ensinar novo conhecimento, mas alinhar o comportamento do modelo a um estilo preferível.

O DPO requer um *dataset* de preferências no formato `(prompt, chosen, rejected)`. A criação desses datasets é um dos grandes gargalos em AI Alignment, pois geralmente exige anotação humana especializada e cara. Para fins didáticos, utilizamos um pequeno *dataset* de exemplo. O mecanismo do DPO ajusta o modelo para que a probabilidade da resposta chosen seja maximizada em relação à rejected, aprendendo implicitamente uma função de recompensa. O ponto de partida para o DPO é o nosso modelo já ajustado com SFT. Utilizamos o `DPOTrainer` da biblioteca `trl`, que ajustará os pesos do modelo com base no *dataset* de preferências para alinhar seu comportamento.

```
import torch
from peft import PeftModel
from transformers import AutoModelForCausalLM, AutoTokenizer
from trl import DPOConfig, DPOTrainer


# Carregamento do dataset de preferência
dataset_dpo = load_dataset(DATASET_DPO, split="train")
dataset_dpo = dataset_dpo.train_test_split(test_size=0.1)


# Carregamento do modelo ajustado com SFT
caminho_adaptadores = "./modelo_sft/final_checkpoint"

modelo_base = AutoModelForCausalLM.from_pretrained(
    NOME_MODELO,
    trust_remote_code=True,
    torch_dtype=torch.bfloat16,
    device_map="auto",
)

# Anexa os adaptadores LoRA do SFT e os torna treináveis novamente para a fase de
# DPO.
modelo_dpo = PeftModel.from_pretrained(
    modelo_base,
    caminho_adaptadores,
```

```
    is_trainable=True
)

tokenizer = AutoTokenizer.from_pretrained(caminho_adaptadores,
trust_remote_code=True)

# Configura o token de padding, se necessário.
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# A classe DPOConfig armazena todos os hiperparâmetros para o treinamento.
config_dpo = DPOConfig(
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    gradient_accumulation_steps=8,
    num_train_epochs=1,
    learning_rate=5e-7,
    logging_steps=10,
    eval_strategy="steps",
    eval_steps=100,
    save_strategy="steps",
    save_steps=500,
    save_total_limit=2,
    report_to="none",
    remove_unused_columns=False,
    max_length=512,
    padding_value=None,
    beta=0.5,
    loss_type="sigmoid",
)

# Instanciação e execução do treinador DPO
treinador_dpo = DPOTrainer(
    model=modelo_dpo,
    ref_model=None,
    args=config_dpo,
    processing_class=tokenizer,
    train_dataset=dataset_dpo["train"],
    eval_dataset=dataset_dpo["test"],
)
treinador_dpo.train()

# Salva os adaptadores LoRA refinados pelo DPO.
output_dir_dpo = "./modelo_dpo"
treinador_dpo.save_model(output_dir_dpo)
tokenizer.save_pretrained(output_dir_dpo)
```

Após a etapa de DPO, a acurácia do modelo foi de 25,06% (326 acertos). O ganho de 0,08 pontos percentuais é marginal, o que é esperado. O DPO não foi projetado para melhorar a performance em testes de conhecimento de múltipla escolha, mas sim para moldar o estilo da geração de texto. Uma avaliação qualitativa, analisando a cautela, clareza e segurança das respostas abertas do modelo, seria necessária para medir o verdadeiro impacto desta etapa em cenários reais de avaliação.

3.5.5 Considerações finais

Neste estudo de caso, transformamos o modelo generalista `Qwen/Qwen3-1.7B` em um protótipo de especialista em saúde. Partimos de uma acurácia de 21,60% e, por meio de um processo incremental de duas fases, aprimoramos tanto seu conhecimento quanto seu comportamento.

1. Ajuste fino supervisionado: Com QLoRA, infundimos conhecimento médico específico, elevando a acurácia para 24,98%. Esta etapa foi crucial para ensinar ao modelo o conteúdo factual do domínio.
2. Otimização por preferência direta: Com um *dataset* de preferências, alinhamos o comportamento do modelo, resultando em uma acurácia final de 25,06% e refinando o estilo e a forma da resposta.

A análise comparativa revela os pontos fortes e fracos inerentes a cada técnica. O SFT é a principal fonte de ganho de conhecimento, enquanto o DPO oferece um controle mais refinado sobre o comportamento. Este exemplo prático demonstra que a especialização de LLMs não é um ato único, mas um processo iterativo, onde a combinação de técnicas permite criar modelos mais capazes, seguros e alinhados. O Dr. Bode, desenvolvido neste capítulo, embora ainda um protótipo, ilustra o potencial e os desafios práticos dessa jornada.

Referências

ALKHAMISSI, B. et al. **Investigating Cultural Alignment of Large Language Models**. (L.-W. Ku, A. Martins, V. Srikumar, Eds.)Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). **Anais**...Bangkok, Thailand: Association for Computational Linguistics, ago. 2024. Disponível em: <https://aclanthology.org/2024.acl-long.671/>

AMODEI, D. et al. **Concrete Problems in AI Safety**. arXiv preprint arXiv:1606.06565. **Anais**...2016.

ANISUZZAMAN, D. et al. Fine-tuning large language models for specialized use cases. **Mayo Clinic Proceedings: Digital Health**, v. 3, n. 1, p. 100184, 2025.

BAI, J. et al. Qwen technical report. **arXiv preprint arXiv:2309.16609**, 2023.

BOSTROM, N. **Superintelligence: Paths, Dangers, Strategies**. Oxford, UK: Oxford University Press, 2014.

CARLSMITH, J. **Is Power-Seeking AI an Existential Risk?** [s.l.] Open Philanthropy, 2022.

CHRISTIANO, P. F. et al. **Deep Reinforcement Learning from Human Preferences**. (I. Guyon et al., Eds.)Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. **Anais**...2017. Disponível em: <https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html>

DETTMERS, T. et al. QLoRA: Efficient Finetuning of Quantized LLMs. **arXiv preprint arXiv:2305.14314**, 2023.

DEVLIN, J. et al. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. (J. Burstein, C. Doran, T. Solorio, Eds.)Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019. **Anais**...Minneapolis, MN, USA: Association for Computational Linguistics, 2019. Disponível em: <https://doi.org/10.18653/v1/n19-1423>

DING, N. et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. **Nature machine intelligence**, v. 5, n. 3, p. 220–235, 2023.

GABRIEL, I. Artificial Intelligence, Values and Alignment. **Minds and Machines**, v. 30, n. 3, p. 411–437, 2020.

GAJULAMANDYAM, D. K. et al. **Domain Specific Finetuning of LLMs Using PEFT Techniques**. 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC). **Anais**...2025.

HENDRYCKS, D. et al. Unsolved Problems in ML Safety. **arXiv preprint arXiv:2109.13916**, 2021.

HOULSBY, N. et al. **Parameter-Efficient Transfer Learning for NLP**. (K. Chaudhuri, R. Salakhutdinov, Eds.)Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June

2019, Long Beach, California, USA. **Anais**...: Proceedings of Machine Learning Research.PMLR, 2019. Disponível em: <http://proceedings.mlr.press/v97/houlsby19a.html>

HOWARD, J.; RUDER, S. **Universal Language Model Fine-tuning for Text Classification**. (I. Gurevych, Y. Miyao, Eds.)Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). **Anais**...Melbourne, Australia: Association for Computational Linguistics, jul. 2018. Disponível em: <https://aclanthology.org/P18-1031/>. Acesso em: 25 jul. 2025

HU, E. J. et al. **LoRA: Low-Rank Adaptation of Large Language Models**., 2021. Disponível em: <https://arxiv.org/abs/2106.09685>

KRAKOVNA, V. et al. Specification Gaming: The Flip Side of AI Ingenuity. **DeepMind Safety Research Blog**, 2020.

LAI, J. et al. Large language models in law: A survey. **AI Open**, v. 5, p. 181–196, 2024.

LEE, J. et al. A survey of large language models in finance (finllms). **arXiv preprint arXiv:2402.02315**, 2024.

LEIKE, J. et al. **Scalable Agent Alignment via Reward Modeling: a Research Direction**. arXiv preprint arXiv:1811.07871. **Anais**...2018.

LI, Q. **Parameter Efficient Fine-Tuning on Selective Parameters for Transformer-Based Pre-Trained Models**. 2024 IEEE International Conference on Multimedia and Expo (ICME). **Anais**...2024.

LIU, Y. et al. Roberta: A robustly optimized bert pretraining approach. **arXiv preprint arXiv:1907.11692**, 2019.

LV, K. et al. **Full Parameter Fine-tuning for Large Language Models with Limited Resources**., 2024. Disponível em: <https://arxiv.org/abs/2306.09782>

MINAEE, S. et al. Large language models: A survey. **arXiv preprint arXiv:2402.06196**, 2024.

NGO, R.; CHAN, L.; MINDERMANN, S. The Alignment Problem from a Deep Learning Perspective. **arXiv preprint arXiv:2303.16200**, 2023.

OUYANG, L. et al. **Training language models to follow instructions with human feedback**. (A. H. Oh et al., Eds.)Advances in Neural Information Processing Systems. **Anais**...2022. Disponível em: <https://openreview.net/forum?id=TG8KACxEON>

PAIOLA, P. H. et al. Adapting llms for the medical domain in portuguese: A study on fine-tuning and model evaluation. **arXiv preprint arXiv:2410.00163**, 2024.

RADFORD, A. et al. Language models are unsupervised multitask learners. **OpenAI blog**, v. 1, n. 8, p. 9, 2019.

RAFAILOV, R. et al. **Direct Preference Optimization: Your Language Model is Secretly a Reward Model**. Advances in Neural Information Processing Systems (NeurIPS). **Anais**...2023.

RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. **J. Mach. Learn. Res.**, v. 21, n. 1, p. 140:5485–140:5551, jan. 2020.

RUSSEL, S. **Human Compatible Artificial Intelligence and the Problem of Control**. [s.l.] Penguin Books, 2019.

SCHULMAN, J. et al. **Proximal Policy Optimization Algorithms**. arXiv preprint arXiv:1707.06347. **Anais**...2017.

SHAH, R. et al. Goal Misgeneralization in Deep Reinforcement Learning. **arXiv preprint**

arXiv:2210.01790, 2022.

SHI, H. et al. Continual learning of large language models: A comprehensive survey. **ACM Computing Surveys**, 2024.

STIENNON, N. et al. Learning to Summarize with Human Feedback. **Advances in Neural Information Processing Systems (NeurIPS)**, 2020.

VASWANI, A. et al. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

WU, X.-K. et al. LLM Fine-Tuning: Concepts, Opportunities, and Challenges. **Big Data and Cognitive Computing**, v. 9, n. 4, p. 87, abr. 2025.

YE, Y. et al. **LIMO: Less is More for Reasoning**. arXiv, jul. 2025. Disponível em: <http://arxiv.org/abs/2502.03387>. Acesso em: 7 ago. 2025

ZHOU, H. et al. A survey of large language models in medicine: Progress, application, and challenge. **arXiv preprint arXiv:2311.05112**, 2023.