# **Análise Comparativa de Frameworks para Modelos BERT e Grandes Modelos de Linguagem**

**I. Introdução**

A introdução de arquiteturas Transformer, exemplificadas por modelos como o BERT (Bidirectional Encoder Representations from Transformers) 1, marcou uma mudança de paradigma no processamento de linguagem natural (PLN) e em outras áreas da inteligência artificial. Estes modelos demonstraram capacidades sem precedentes na compreensão e geração de linguagem, impulsionando avanços em tarefas que vão desde a classificação de texto e resposta a perguntas até à tradução automática e sumarização.1 No entanto, este progresso veio acompanhado por um crescimento exponencial no tamanho e complexidade dos modelos. Modelos com centenas de bilhões ou mesmo trilhões de parâmetros tornaram-se cada vez mais comuns, como evidenciado pelo desenvolvimento de sistemas como MT-NLG e BLOOM, frequentemente treinados com o auxílio de bibliotecas de otimização.4

Este aumento de escala apresenta desafios computacionais e de memória significativos.5 O treinamento de tais modelos massivos exige vastos conjuntos de dados, recursos substanciais de GPU (Graphics Processing Unit) ou TPU (Tensor Processing Unit) e técnicas sofisticadas para gerenciar eficientemente a memória e paralelizar os cálculos. A inferência, o processo de usar um modelo treinado para fazer previsões, também requer otimização cuidadosa para atingir baixa latência e alta taxa de transferência (throughput), especialmente em aplicações em tempo real.4 Superar estes desafios tornou-se um foco central na pesquisa e engenharia de aprendizado de máquina (Machine Learning - ML).

O propósito deste relatório é fornecer uma análise técnica aprofundada e uma comparação dos proeminentes frameworks de software e bibliotecas que permitem e otimizam o desenvolvimento, treinamento e implantação de BERT e outros grandes modelos de linguagem (Large Language Models - LLMs). A análise abrange frameworks fundamentais que fornecem os blocos de construção essenciais, bibliotecas de otimização especializadas projetadas para lidar com a escala extrema e camadas de abstração de alto nível que visam simplificar os fluxos de trabalho e promover a interoperabilidade. O escopo inclui TensorFlow, PyTorch, JAX/Flax, DeepSpeed, Megatron-LM, Hugging Face Transformers e Keras, focando em suas capacidades técnicas, padrões de uso com BERT/LLMs e modelos chave associados, com base no material de pesquisa disponível.

**II. Frameworks Fundamentais de Deep Learning**

Esta categoria engloba os motores principais que fornecem as operações de tensor, diferenciação automática e blocos de construção básicos de redes neurais, sobre os quais outras ferramentas frequentemente se baseiam. A escolha entre estes frameworks muitas vezes define a base para todo o fluxo de trabalho de desenvolvimento e implantação de ML.

* **A. TensorFlow**
  + **Definição e Propósito:** TensorFlow é uma biblioteca de software de código aberto desenvolvida pela equipe do Google Brain 8 para computação numérica usando grafos de fluxo de dados.9 Seu propósito principal é o treinamento e a inferência de redes neurais 8, sendo um dos frameworks de deep learning mais populares.8 Lançado inicialmente em 2015, o Google lançou uma versão atualizada, TensorFlow 2.0, em setembro de 2019.8
  + **Recursos Principais:** O TensorFlow oferece computação de tensores com forte aceleração por GPU.9 Embora possua capacidades de diferenciação automática, estas são talvez menos enfatizadas em sua filosofia de design recente em comparação com PyTorch ou JAX. Historicamente, o TensorFlow utilizava grafos de fluxo de dados estáticos, onde o grafo computacional era definido primeiro e depois executado numa sessão 9; no entanto, o TensorFlow 2.x adotou a execução eager por padrão, tornando-o mais interativo, embora a compilação de grafos ainda seja usada para otimização. O framework suporta múltiplas linguagens de programação, incluindo Python, C++, Java e JavaScript (através do TensorFlow.js).8 Seu ecossistema robusto inclui o TensorFlow Lite para implantação em dispositivos móveis e de borda 8, TensorFlow Serving para implantação em servidores 8, TensorFlow Hub para compartilhamento de modelos pré-treinados 11 e TensorBoard para visualização de treinamento. Uma característica definidora do TensorFlow 2.x é sua integração profunda com a API Keras, que serve como sua interface de alto nível para construção de modelos 8, simplificando significativamente o desenvolvimento. Além disso, o TensorFlow Text fornece utilitários essenciais para pré-processamento de texto.11
  + **Uso para BERT/LLMs:** TensorFlow é amplamente utilizado tanto para pré-treinamento 16 quanto para fine-tuning 12 de modelos BERT. Exemplos e tutoriais oficiais frequentemente demonstram o uso de tf.data para criar pipelines de entrada eficientes 16, a API Keras para definir a arquitetura do modelo 12 e otimizadores específicos como AdamW.12 A NVIDIA também fornece implementações otimizadas de BERT para TensorFlow, visando alto desempenho em suas GPUs.17 Para implantação, o TensorFlow Serving é uma solução comum e robusta, capaz de servir modelos BERT baseados em Keras em produção.10 Modelos BERT pré-treinados são facilmente acessíveis através do TensorFlow Hub 11 e do TensorFlow Model Garden 18, facilitando o início rápido com transfer learning.
  + **Modelos Associados:** Além de várias versões do BERT disponíveis no TF Hub 11 e Model Garden 20, o TensorFlow é associado a muitos outros modelos desenvolvidos pelo Google, como XLNet, T5, PaLM e Gemini 8, embora os detalhes de implementação possam variar. Presets do KerasHub frequentemente oferecem backends TensorFlow.24
  + A evolução do TensorFlow, particularmente com a versão 2.0, reflete uma mudança estratégica em direção a uma maior usabilidade. A integração profunda do Keras como a API de alto nível padrão 8 e a promoção do TensorFlow Hub 12 tornaram o framework mais acessível, especialmente para usuários que podem ter achado a abordagem baseada em grafos estáticos do TensorFlow 1.x mais complexa. Esta mudança parece ser uma resposta à crescente popularidade de frameworks mais "Pythonicos" e dinâmicos, como o PyTorch 27, buscando oferecer uma experiência de desenvolvimento mais suave sem sacrificar o poder subjacente. A disponibilidade de tutoriais detalhados focados em fine-tuning de BERT usando Keras e TF Hub 12 sublinha este foco na facilidade de uso para tarefas comuns de PLN.
  + Apesar da mudança em direção à usabilidade, a força histórica do TensorFlow reside em seu ecossistema maduro para produção. Ferramentas como TensorFlow Serving 10 e TensorFlow Lite 8 fornecem caminhos bem estabelecidos e robustos para implantar modelos em uma variedade de ambientes, desde servidores de alta capacidade até dispositivos móveis e de borda com recursos limitados. Este foco duplo em pesquisa e produção, desde seus primórdios no Google 8, levou ao desenvolvimento precoce dessas ferramentas de implantação, conferindo ao TensorFlow uma reputação de "pronto para produção" que historicamente o diferenciou de frameworks mais focados em pesquisa.
* **B. PyTorch**
  + **Definição e Propósito:** PyTorch é um framework de ML de código aberto, baseado na biblioteca Torch e desenvolvido principalmente pelo Meta AI.28 É conhecido por sua flexibilidade, interface Pythonica e forte aceleração por GPU.27 Ele fornece computação de tensores (semelhante ao NumPy) e redes neurais profundas construídas sobre um sistema de diferenciação automática baseado em fita (tape-based autograd).27
  + **Recursos Principais:** Uma característica chave do PyTorch é o uso de grafos computacionais dinâmicos, que são definidos durante a execução ("define-by-run").27 Isso contrasta com os grafos estáticos do TensorFlow 1.x e contribui para seu estilo de programação imperativo e Pythonico, que muitos pesquisadores acham intuitivo.27 O framework oferece excelente aceleração por GPU.27 Seus componentes principais incluem torch.nn para construir redes neurais 27, torch.autograd para diferenciação automática 27 e torch.optim para algoritmos de otimização.29 Para escalar o treinamento, o torch.distributed fornece funcionalidades robustas para treinamento distribuído.28 O ecossistema PyTorch inclui TorchScript para compilar modelos para um modo de grafo (para otimização e implantação) 28, TorchServe para servir modelos em produção 28, Captum para interpretabilidade de modelos 28 e PyTorch Geometric para aprendizado profundo em grafos.28 Ele também se integra bem com bibliotecas externas populares, notavelmente a Hugging Face Transformers.30
  + **Uso para BERT/LLMs:** PyTorch é extremamente popular para o treinamento (pré-treinamento e fine-tuning) de BERT e seus variantes.31 Sua flexibilidade o torna uma escolha preferida em ambientes de pesquisa.28 Existem muitos tutoriais e exemplos que demonstram como construir componentes do BERT (embeddings, atenção, encoder) do zero 32 ou como usar módulos pré-construídos, especialmente através da biblioteca Hugging Face Transformers.30 O treinamento distribuído é uma capacidade central frequentemente utilizada para LLMs.28 A biblioteca Hugging Face Transformers, que se tornou um padrão de fato para modelos de PLN, utiliza PyTorch como seu backend principal 36, fornecendo acesso fácil a inúmeros modelos BERT pré-treinados.1 O PyTorch Hub também hospeda modelos.30
  + **Modelos Associados:** BERT (inúmeras implementações, padrão no Hugging Face), RoBERTa, variantes GPT (frequentemente implementadas primeiro em PyTorch pela comunidade de pesquisa), modelos do Meta AI.28 Muitos modelos treinados usando ferramentas de otimização como DeepSpeed 38 ou Megatron-LM 40 são construídos sobre PyTorch.
  + A flexibilidade e a natureza Pythonica do PyTorch 27 contribuíram enormemente para sua adoção generalizada na comunidade de pesquisa.28 A capacidade de depurar código usando ferramentas Python padrão e a natureza dinâmica dos grafos computacionais facilitam a prototipagem rápida e a experimentação com arquiteturas de modelos complexas. Esta popularidade foi significativamente amplificada pelo ecossistema Hugging Face, que construiu sua biblioteca Transformers principalmente sobre PyTorch.30 Isso criou um ciclo virtuoso: pesquisadores desenvolviam novos modelos em PyTorch, os disponibilizavam através do Hugging Face, que por sua vez tornava PyTorch ainda mais atraente para a comunidade de PLN.
  + Embora inicialmente percebido como mais forte em pesquisa, PyTorch investiu ativamente no desenvolvimento de caminhos para a produção. A introdução do TorchScript permitiu que modelos definidos dinamicamente fossem capturados em um formato de grafo estático, possibilitando otimizações e implantação em ambientes onde a flexibilidade dinâmica não é necessária ou desejada.28 Complementarmente, o TorchServe foi desenvolvido para fornecer uma solução robusta e escalável para servir modelos PyTorch em produção.28 Estes desenvolvimentos representam um esforço concertado para tornar o PyTorch um framework completo de ponta a ponta, abordando as preocupações iniciais sobre sua prontidão para produção em comparação com o ecossistema mais maduro do TensorFlow naquela época.
* **C. JAX e Flax**
  + **Definição e Propósito:**
    - **JAX:** É uma biblioteca Python da Google Research 8 projetada para computação numérica de alto desempenho e pesquisa em aprendizado de máquina.41 Sua singularidade reside na combinação de uma API familiar no estilo NumPy com transformações de função composáveis (grad, jit, vmap, pmap) que operam sobre código Python/NumPy padrão.41 Essas transformações permitem diferenciação automática, compilação just-in-time (JIT) para aceleradores (CPU, GPU, TPU) e paralelização automática. JAX efetivamente une uma versão modificada do Autograd com o compilador XLA (Accelerated Linear Algebra) do TensorFlow.8
    - **Flax:** É uma biblioteca de rede neural de código aberto para JAX, também da Google Research 45, projetada com foco em flexibilidade e clareza.45 Enquanto JAX fornece as operações de array de baixo nível e as transformações, Flax oferece uma abstração de nível superior para construir redes neurais. Isso inclui a definição de camadas (como módulos), gerenciamento de parâmetros e estado, e utilitários para treinamento.41 Flax funciona como uma camada estrutural sobre o motor de computação de alto desempenho do JAX.
  + **Recursos Principais:**
    - **JAX:** Os pilares do JAX são suas transformações de função: jit compila funções Python para código XLA otimizado para execução em aceleradores 41; grad calcula gradientes de funções Python 41; vmap realiza vetorização automática (mapeamento de uma função sobre eixos de lote) 41; e pmap paraleliza automaticamente a execução de uma função em múltiplos dispositivos (como GPUs ou núcleos TPU).41 JAX incentiva um paradigma de programação funcional, onde as transformações operam sobre funções puras (sem efeitos colaterais).42 É particularmente conhecido por seu desempenho em TPUs.44
    - **Flax:** Introduz o conceito de Módulos (análogos a camadas em outros frameworks) como blocos de construção composáveis.45 Ele gerencia explicitamente os parâmetros do modelo (pesos treináveis) e o estado (variáveis não treináveis, como médias móveis em batch normalization).45 Integra-se perfeitamente com as transformações JAX e colabora com outras bibliotecas do ecossistema JAX, como Optax para otimizadores.41
  + **Uso para BERT/Transformers:** A combinação JAX/Flax é poderosa para o treinamento de alto desempenho de grandes modelos Transformer. As capacidades de compilação e paralelização do JAX são cruciais para a eficiência, especialmente em clusters de TPU.44 Flax simplifica a definição da arquitetura complexa do Transformer, incluindo mecanismos de atenção e blocos de encoder/decoder.45 A biblioteca Hugging Face Transformers oferece implementações em Flax para muitos modelos populares, incluindo exemplos para modelagem de linguagem causal (GPT2), modelagem de linguagem mascarada (RoBERTa) e classificação de texto (BERT).44 Projetos de pesquisa do Google frequentemente lançam código em JAX/Flax.47 Além disso, o Keras 3 pode usar JAX como backend computacional.14
  + **Modelos Associados:** BERT, RoBERTa, GPT2 (versões Flax no Hugging Face) 44, T5X 47, MaxText 41, Vision Transformer (ViT) 47, e outros modelos proeminentes originados da pesquisa do Google/DeepMind. Modelos que utilizam Keras com o backend JAX.15
  + A filosofia central de design do JAX, focada em transformações de função e compilação XLA 8, o torna excepcionalmente adequado para tarefas de ML em larga escala e críticas em termos de desempenho. Sua origem no Google o otimizou particularmente para o hardware TPU da empresa.44 Flax complementa JAX fornecendo as abstrações necessárias para tornar o desenvolvimento de redes neurais viável dentro deste paradigma focado no desempenho.45 Esta combinação atrai pesquisadores e engenheiros que trabalham na vanguarda de modelos grandes, onde a eficiência computacional é primordial, levando ao seu uso em projetos ambiciosos como T5X 47 e MaxText.41
  + No entanto, o paradigma de programação funcional, com sua exigência de funções puras para que transformações como jit e pmap funcionem corretamente 42, pode apresentar uma curva de aprendizado mais acentuada em comparação com os estilos mais orientados a objetos do TensorFlow (com Keras) e PyTorch. Os desenvolvedores precisam pensar de forma diferente sobre o gerenciamento de estado, passando explicitamente parâmetros e estado para as funções.45 Embora isso possa ser desafiador inicialmente, essa explicitude e pureza permitem otimizações de compilador mais agressivas via XLA 8 e facilitam a paralelização (pmap) 44, resultando em ganhos de desempenho significativos, especialmente em grande escala.

**III. Frameworks para Otimização de Modelos em Larga Escala**

Esta categoria abrange bibliotecas especializadas, frequentemente construídas sobre frameworks fundamentais como PyTorch, projetadas especificamente para enfrentar os gargalos de memória e computação encontrados ao treinar modelos extremamente grandes, com bilhões ou trilhões de parâmetros.

* **A. DeepSpeed**
  + **Definição e Propósito:** DeepSpeed é uma biblioteca de otimização de deep learning da Microsoft Research 4 criada para tornar o treinamento distribuído e a inferência de modelos massivos fáceis, eficientes e eficazes.4 Seu objetivo é melhorar a velocidade, a escala, a relação custo-benefício e a usabilidade no treinamento e implantação de modelos que ultrapassam os limites dos sistemas de GPU tradicionais.4
  + **Recursos Principais:**
    - **ZeRO (Zero Redundancy Optimizer):** A inovação central do DeepSpeed para otimização de memória. ZeRO elimina redundâncias de memória no treinamento distribuído com paralelismo de dados, particionando os estados do modelo (parâmetros, gradientes e estados do otimizador) entre os processos de dados paralelos. São oferecidos diferentes estágios (ZeRO-1, ZeRO-2, ZeRO-3) que progressivamente particionam mais estados, reduzindo drasticamente a memória necessária por GPU e permitindo o treinamento de modelos muito maiores.4
    - **ZeRO-Infinity:** Uma extensão do ZeRO-3 que permite descarregar (offload) os estados do modelo particionados para a memória da CPU e até mesmo para armazenamento NVMe (Non-Volatile Memory Express) rápido. Isso quebra efetivamente a barreira da memória da GPU, possibilitando o treinamento de modelos com trilhões de parâmetros em hardware mais acessível.4 Requer o uso do ZeRO-3.50
    - **Paralelismo 3D:** DeepSpeed combina estrategicamente diferentes dimensões de paralelismo – Paralelismo de Dados (distribuindo dados), Paralelismo de Pipeline (dividindo camadas sequencialmente) e Paralelismo de Tensor (dividindo operações dentro das camadas) – para obter escalabilidade e eficiência holísticas.4 O DeepSpeed-AutoTP pode automatizar a aplicação do paralelismo de tensor para modelos Hugging Face.4
    - **Otimizações de Comunicação:** Inclui técnicas para minimizar a sobrecarga de comunicação entre GPUs durante o treinamento distribuído, como o ZeRO++ (que introduz otimizações como o particionamento hierárquico - hpZ) 4 e a sobreposição de comunicação com computação.49
    - **Otimização de Inferência (DeepSpeed-Inference):** Um conjunto de ferramentas e técnicas para otimizar a inferência de modelos grandes, utilizando paralelismo, kernels CUDA personalizados e gerenciamento de memória heterogênea para alcançar baixa latência e alta taxa de transferência.7 Inclui otimizações como descarregamento eficiente de cache KV (Key-Value).4
    - **Compressão (DeepSpeed-Compression):** Oferece técnicas como ZeroQuant para compressão de modelos, reduzindo o tamanho do modelo e acelerando a inferência com baixo custo.7
    - **Usabilidade:** Fornece uma API simplificada (métodos como initialize, backward, step) 38 e se integra facilmente com frameworks populares como PyTorch e bibliotecas como Hugging Face Transformers 38, PyTorch Lightning 39 e Accelerate.39 A configuração é gerenciada através de um arquivo JSON.38
  + **Uso para Escalar BERT/LLMs:** DeepSpeed é explicitamente projetado e usado para treinar modelos grandes como BERT 38 e LLMs muito maiores (por exemplo, MT-NLG 530B, BLOOM 176B).4 Suas técnicas de otimização de memória e paralelismo permitem treinar modelos de bilhões ou trilhões de parâmetros no hardware disponível.4 É aplicado tanto no pré-treinamento 54 quanto no fine-tuning 53 e também na otimização da inferência.52
  + **Modelos Associados:** BERT 38, MT-NLG 530B 4, BLOOM 176B 4, Llama/Llama-2 (via DeepSpeed-Chat) 4, Mixtral, Phi-2, Falcon (via DeepSpeed-FastGen) 4, GPT-NeoX, AlexaTM, Turing NLG.4 Muitos modelos treinados usando o Hugging Face Trainer com uma configuração DeepSpeed.50
  + A principal inovação do DeepSpeed, o ZeRO 49, aborda diretamente o gargalo mais crítico no treinamento de modelos grandes: as limitações de memória da GPU. O paralelismo de dados tradicional replica todo o modelo, estados do otimizador e gradientes em cada GPU 6, tornando-se inviável para modelos com bilhões de parâmetros. ZeRO reconheceu que grande parte desse estado é redundante durante a etapa de atualização do otimizador. Ao particionar esses estados entre as GPUs (Estágio 1: otimizador; Estágio 2: + gradientes; Estágio 3: + parâmetros) 50, DeepSpeed reduz drasticamente a pegada de memória por GPU, permitindo escalas de modelo anteriormente impossíveis.4 ZeRO-Infinity 4 leva isso adiante, tratando a RAM do sistema e o armazenamento NVMe como camadas de memória adicionais, expandindo ainda mais a capacidade.
  + DeepSpeed se destaca por sua abordagem sistêmica e holística para a otimização. Não se trata apenas de uma única técnica; ele combina múltiplas dimensões de paralelismo (Dados, Pipeline, Tensor 4), otimização de memória (ZeRO 49), redução de comunicação 49 e até otimização de E/S (DeepNVMe 4) em uma suíte abrangente.4 O treinamento eficiente de modelos massivos requer a otimização de todo o pipeline, desde o carregamento de dados até a atualização dos pesos. Ao integrar soluções para múltiplos gargalos (memória, computação, comunicação, E/S) em uma única biblioteca, DeepSpeed oferece uma solução mais completa do que frameworks que se concentram isoladamente em paralelismo ou memória.
* **B. Megatron-LM**
  + **Definição e Propósito:** Megatron-LM é uma biblioteca desenvolvida pela NVIDIA 40 focada no treinamento de modelos Transformer extremamente grandes (LLMs).40 Sua principal contribuição é a implementação de técnicas eficientes de paralelismo de tensor e de pipeline, otimizadas especificamente para a arquitetura das GPUs NVIDIA e suas interconexões.40 Megatron-Core é uma evolução mais recente, projetada para ser mais modular e composável.40
  + **Recursos Principais:**
    - **Paralelismo de Modelo Tensorial (Tensor Model Parallelism):** Esta técnica divide a computação de camadas individuais (por exemplo, multiplicações de matrizes em camadas lineares ou de atenção) entre múltiplas GPUs, geralmente dentro de um mesmo nó de computação conectado por links de alta velocidade como NVLink.6 Isso reduz a memória necessária por GPU para armazenar pesos e ativações intermediárias, permitindo que camadas maiores sejam processadas.
    - **Paralelismo de Modelo em Pipeline (Pipeline Model Parallelism):** Divide as camadas do modelo sequencialmente em múltiplos estágios, onde cada estágio é atribuído a uma ou mais GPUs (potencialmente distribuídas entre nós).6 O lote de treinamento é dividido em micro-lotes que fluem através desses estágios de forma pipelinada, permitindo que diferentes estágios processem diferentes micro-lotes concorrentemente. Isso permite escalar a profundidade do modelo além dos limites de um único nó.
    - **Paralelismo de Sequência (Sequence Parallelism):** Embora mencionado principalmente no contexto de trabalhos inspirados no Megatron 6, esta técnica, que paraleliza cálculos ao longo da dimensão da sequência, é frequentemente integrada para lidar eficientemente com sequências de entrada muito longas, dividindo o trabalho entre GPUs de paralelismo tensorial.
    - **Otimizador Distribuído:** Coordena as etapas de otimização entre os diferentes ranks paralelos (tensor, pipeline, dados).
    - **Carregamento de Dados:** Pipelines otimizados para carregar e pré-processar eficientemente grandes conjuntos de dados.56
    - **Integração:** Megatron-LM pode ser combinado com outras bibliotecas como DeepSpeed (por exemplo, usando ZeRO para otimização de memória em conjunto com o paralelismo de Megatron).40 O Megatron-Core visa facilitar essa composição através de APIs mais modulares.55
  + **Uso para Escalar BERT/LLMs:** Megatron-LM é usado primariamente para o pré-treinamento de modelos Transformer massivos, como variantes de GPT e BERT.40 Sua força reside na capacidade de escalar para centenas de bilhões ou trilhões de parâmetros, combinando eficazmente o paralelismo tensorial e de pipeline.40 É frequentemente a escolha em grandes laboratórios industriais ou de pesquisa que possuem acesso a grandes clusters de GPUs NVIDIA e buscam o desempenho máximo.57
  + **Modelos Associados:** Megatron-Turing NLG 530B (treinado em conjunto com DeepSpeed) 40, modelos BERT/GPT estudados em pesquisas da NVIDIA 40, BioMegatron 40, e outros modelos de grande escala citados em publicações que utilizaram Megatron.40 Modelos treinados usando frameworks construídos sobre o Megatron-Core.55
  + Megatron-LM desempenhou um papel pioneiro no desenvolvimento e popularização de técnicas de paralelismo de modelo, especificamente paralelismo tensorial e de pipeline 6, como soluções viáveis para escalar modelos Transformer além dos limites do paralelismo de dados tradicional. À medida que os modelos cresciam, tornou-se impossível encaixar até mesmo os pesos ou ativações de uma única camada em uma GPU. O paralelismo de modelo, ao dividir o próprio modelo 57 – seja horizontalmente (tensor) 6 ou verticalmente (pipeline) 6 –, permitiu distribuir esses modelos massivos por muitas GPUs de forma eficaz, possibilitando avanços como o MT-NLG.40
  + Desenvolvido pela NVIDIA 40, Megatron-LM é intrinsecamente otimizado para GPUs NVIDIA e tecnologias associadas como NVLink. Esta afinidade com o hardware, combinada com seu foco em técnicas de paralelismo complexas 6, posiciona-o frequentemente como um framework de pesquisa ou uma base para construir sistemas de treinamento em larga escala, em vez de uma ferramenta pronta para uso por usuários gerais. Sua complexidade e foco em pesquisa podem torná-lo menos acessível do que bibliotecas como DeepSpeed ou Hugging Face Transformers, que priorizam uma usabilidade mais ampla e integração.3 A modularidade introduzida com o Megatron-Core 55 pode ser vista como um esforço para ampliar sua aplicabilidade e facilitar sua integração em sistemas maiores.

**IV. Frameworks de Alto Nível e Interoperabilidade**

Esta seção aborda frameworks que fornecem abstrações de nível superior sobre os frameworks fundamentais (TensorFlow, PyTorch, JAX). O objetivo principal dessas ferramentas é simplificar fluxos de trabalho comuns de ML, promover a reutilização de código e facilitar o acesso a modelos pré-treinados, tornando a tecnologia de ponta mais acessível.

* **A. Hugging Face Transformers**
  + **Definição e Propósito:** Hugging Face Transformers é uma biblioteca de código aberto extremamente popular que fornece APIs e ferramentas para baixar, treinar e usar facilmente modelos pré-treinados de última geração em diversas modalidades, incluindo PLN, Visão Computacional, Áudio e Multimodal.3 Sua missão declarada é democratizar o acesso a modelos Transformer e relacionados, reduzindo custos computacionais, pegada de carbono e o tempo necessário para treinar modelos do zero.3
  + **Recursos Principais:**
    - **Model Hub:** Um repositório centralizado que hospeda milhares de modelos pré-treinados, conjuntos de dados e demonstrações interativas (Spaces), alimentado pela comunidade.36
    - **API Padronizada:** Oferece uma interface consistente e fácil de usar (from\_pretrained) para carregar diferentes componentes do modelo (Configuração, Tokenizer, Modelo) para uma vasta gama de arquiteturas.37
    - **Pipelines:** Uma abstração de alto nível que simplifica drasticamente a execução de inferência para tarefas comuns (classificação de texto, reconhecimento de entidade nomeada, resposta a perguntas, geração de texto, etc.) com poucas linhas de código.3
    - **AutoClasses:** Classes como AutoModel, AutoTokenizer, AutoConfig que inferem automaticamente a arquitetura correta do modelo a partir do nome ou caminho fornecido, simplificando o carregamento.37
    - **Trainer/TFTrainer:** Classes de treinamento ricas em recursos e otimizadas para PyTorch 36 e TensorFlow, respectivamente. Elas abstraem o loop de treinamento, facilitando o fine-tuning e o pré-treinamento. Suportam treinamento distribuído, precisão mista, integração com DeepSpeed e Accelerate.36
    - **Tokenizers Library:** Uma biblioteca separada, mas integrada, que fornece implementações rápidas e versáteis de vários algoritmos de tokenização (WordPiece, BPE, SentencePiece) usados por modelos Transformer.
    - **Interoperabilidade de Framework:** Suporta nativamente backends PyTorch, TensorFlow e JAX. Isso permite, por exemplo, treinar um modelo em PyTorch e carregá-lo para inferência em TensorFlow.3
  + **Uso para BERT/LLMs:** A biblioteca Transformers é o principal meio pelo qual muitos desenvolvedores e pesquisadores acessam e utilizam modelos BERT pré-treinados (como bert-base-uncased 2, multilíngue 26, etc.) e suas variantes (RoBERTa 44, ALBERT 25, etc.). Ela simplifica enormemente o processo de fine-tuning do BERT para tarefas downstream, como classificação de sequência 1, classificação de token 1, resposta a perguntas 1 e modelagem de linguagem mascarada 1, seja usando a classe Trainer ou loops de treinamento nativos do framework. Fornece todos os componentes necessários, como BertTokenizer 19 e classes de modelo específicas da tarefa (BertForSequenceClassification, BertForQuestionAnswering, etc.).1
  + **Modelos Associados:** A força da biblioteca reside na vasta coleção de modelos disponíveis no Hub.44 Isso inclui inúmeras variantes de BERT 2, RoBERTa 44, GPT-2 44, T5 59, BART 59, DistilBERT 59, ALBERT 25, ELECTRA 59, Llama, Mistral 59, BLOOM 25 e milhares de outros modelos contribuídos pela comunidade e por organizações de pesquisa.
  + O sucesso retumbante da Hugging Face Transformers pode ser atribuído à sua capacidade de abstrair as complexidades inerentes ao uso de diferentes modelos Transformer. Antes de sua popularização, usar um novo modelo frequentemente exigia aprender sobre seus requisitos específicos de pré-processamento, mecanismos de carregamento de peso e base de código idiossincrática. A HF padronizou isso através de AutoClasses, Tokenizer e Pipeline.37 Combinado com o Model Hub 44, que fornece um local centralizado para descoberta e compartilhamento, a biblioteca reduziu drasticamente a barreira de entrada para utilizar modelos de última geração, acelerando a pesquisa e o desenvolvimento de aplicações.
  + Impulsionado pela facilidade de uso da biblioteca Transformers, o Hugging Face Hub 44 emergiu como a plataforma padrão de fato para compartilhar e acessar modelos pré-treinados, não apenas para PLN, mas cada vez mais para outras modalidades. Isso criou um poderoso efeito de rede: a facilidade de acesso atrai usuários, que por sua vez são incentivados a compartilhar seus próprios modelos no Hub, enriquecendo ainda mais a plataforma e solidificando seu papel central no ecossistema de ML. A forma como a pesquisa é disseminada e como os modelos são aplicados na prática foi profundamente influenciada por esta plataforma.
* **B. Keras**
  + **Definição e Propósito:** Keras é uma API de deep learning de alto nível projetada com foco na experiência do desenvolvedor ("para seres humanos, não máquinas").15 Prioriza a facilidade de uso, modularidade e a capacidade de permitir experimentação rápida.15 Keras teve uma história evolutiva: começou como uma biblioteca independente que suportava múltiplos backends, depois tornou-se a API de alto nível primária e integrada ao TensorFlow.60 Com o Keras 3, ele retornou às suas raízes multi-backend, oferecendo uma interface unificada que pode rodar sobre TensorFlow, JAX ou PyTorch.14
  + **Recursos Principais:**
    - **API Amigável:** Oferece uma interface simples e consistente para definir modelos (usando a API Sequencial para pilhas lineares 61 ou a API Funcional para grafos mais complexos 15), adicionar camadas, compilar o modelo (especificando perda, otimizador e métricas) 61 e treinar com o método fit.61
    - **Modularidade:** Modelos são construídos a partir de blocos composáveis e reutilizáveis, como camadas, funções de ativação, otimizadores e regularizadores.61
    - **Suporte Multi-Backend (Keras 3):** Permite que o mesmo código de modelo Keras seja executado sobre TensorFlow, JAX ou PyTorch, simplesmente configurando uma variável de ambiente.15
    - **KerasCV e KerasNLP:** Bibliotecas específicas de domínio que estendem Keras com componentes pré-construídos para Visão Computacional e PLN, respectivamente. Incluem modelos de backbone pré-treinados, camadas específicas de domínio, tokenizers, métricas e funções de pré-processamento.48
    - **KerasHub:** Uma biblioteca (e parte do ecossistema Keras) que fornece implementações Keras 3 de arquiteturas de modelos populares (incluindo variantes BERT) e pesos pré-treinados, frequentemente hospedados no Kaggle Models.15
    - **Extensibilidade:** É relativamente fácil criar camadas, modelos ou outras componentes personalizadas, herdando das classes base do Keras.61
  + **Uso para BERT/LLMs:** Keras, especialmente através do KerasNLP e KerasHub, simplifica o uso de modelos BERT. Oferece APIs de alto nível como BertClassifier, BertPreprocessor e BertBackbone que encapsulam a complexidade do fine-tuning de BERT em tarefas downstream.24 Também é possível usar camadas BERT do TensorFlow Hub diretamente dentro de um modelo Keras usando hub.KerasLayer.12 Keras suporta fluxos de trabalho de transfer learning padrão, como congelar camadas do modelo base e adicionar novas camadas treináveis no topo.62 Com Keras 3, modelos BERT definidos com a API Keras podem ser executados nos backends TF, JAX ou PyTorch.15 KerasNLP inclui tokenizers (como WordPiece 22) e camadas de pré-processamento compatíveis com BERT.22
  + **Modelos Associados:** BERT (vários presets no KerasHub/KerasNLP como bert\_base\_en\_uncased, bert\_tiny\_en\_uncased) 24, ALBERT.25 Outros modelos proeminentes disponíveis via KerasHub incluem Gemma, Llama, Stable Diffusion e Mistral.15 Modelos que utilizam camadas do TF Hub dentro do Keras.12
  + A reescrita principal do Keras para a versão 3, com o objetivo explícito de suportar múltiplos backends (TensorFlow, JAX, PyTorch) 15, marca uma direção estratégica significativa. Keras visa se posicionar como uma API de alto nível unificadora, permitindo aos usuários aproveitar os pontos fortes de diferentes motores de computação subjacentes (por exemplo, desempenho JAX/XLA 48, ecossistema de implantação TensorFlow 48, ecossistema de pesquisa PyTorch 48) sem a necessidade de reescrever o código do modelo. Isso aborda o desejo do usuário por flexibilidade e potencialmente mitiga o aprisionamento (lock-in) a um único framework.
  + Uma filosofia central do Keras sempre foi a priorização da experiência do desenvolvedor e da facilidade de uso.15 O desenvolvimento de bibliotecas de domínio como KerasCV e KerasNLP 48, juntamente com o KerasHub 15, leva essa filosofia adiante. Essas extensões abstraem ainda mais tarefas comuns, como o uso de backbones pré-treinados, tokenização específica do modelo e fine-tuning para tipos de tarefas específicas (por exemplo, BertClassifier 63). Ao fornecer modelos de Task pré-configurados que agrupam pré-processamento, backbone e cabeça da tarefa, KerasNLP/KerasHub reduzem drasticamente o código boilerplate necessário para casos de uso comuns, como análise de sentimento 63, alinhando-se com o objetivo principal do Keras de permitir experimentação rápida.61

**V. Resumo Comparativo dos Frameworks**

A tabela a seguir sintetiza as informações chave sobre cada framework discutido, fornecendo uma referência rápida para suas características principais e modelos associados.

| **Nome do Framework** | **Resumo Curto** | **Principais Modelos que Utilizam** |
| --- | --- | --- |
| **TensorFlow** | Framework fundamental (Google). Forte ecossistema de produção (Serving, Lite). Usa API Keras. Bom para implantação. | BERT (TF Hub/Garden) 11, T5, PaLM, Gemini 8, modelos Google. Backend para Keras.15 |
| **PyTorch** | Framework fundamental (Meta). Flexível e Pythonico. Dominante em pesquisa. Forte integração com Hugging Face. Ferramentas de produção em crescimento (TorchServe).28 | BERT (padrão HF) 30, RoBERTa, variantes GPT, modelos Meta.28 Base para DeepSpeed/Megatron.38 |
| **JAX** | Biblioteca fundamental (Google) para computação numérica de alto desempenho via transformações de função (JIT, pmap).41 API NumPy. | Usado como backend para Flax/Keras.15 Transformers de alto desempenho. |
| **Flax** | Biblioteca de rede neural para JAX (Google). Estilo funcional, gerenciamento explícito de parâmetros.45 Boa para pesquisa crítica em desempenho. | BERT/RoBERTa/GPT2 (HF Flax) 44, T5X 47, MaxText 41, ViT.47 |
| **DeepSpeed** | Biblioteca de otimização (Microsoft) para treinamento/inferência de modelos grandes. Otimização de memória ZeRO 49, paralelismo 3D.4 Integra com PyTorch/HF. | BLOOM 4, MT-NLG 4, Llama-2 4, Mixtral 4, modelos treinados com HF+DeepSpeed.50 Otimiza BERT.38 |
| **Megatron-LM** | Biblioteca de otimização (NVIDIA) para treinamento de modelos massivos. Paralelismo tensorial/pipeline.6 Otimizado para GPUs NVIDIA. | Megatron-Turing NLG 40, modelos de pesquisa que exigem escala extrema (BERT/GPT variants).40 |
| **Hugging Face Transformers** | Biblioteca/plataforma de alto nível. Vasto Model Hub.44 API padronizada (AutoClasses, Pipeline, Trainer).37 Suporte multi-backend (PyTorch, TF, JAX).3 | Milhares, incluindo BERT 2, GPT 44, T5 59, BART 59, Llama, Mistral.59 |
| **Keras** | API de alto nível. Foco na experiência do usuário. Multi-backend (TF, JAX, PyTorch via Keras 3).15 KerasHub/KerasNLP para presets.15 | BERT (KerasHub) 25, Gemma 67, Llama 15, modelos usando API Keras em vários backends. |

**VI. Síntese e Conclusão**

A análise dos frameworks e bibliotecas para o desenvolvimento, treinamento e implantação de modelos BERT e outros LLMs revela um ecossistema complexo e em camadas. Podemos categorizar essas ferramentas com base em suas funções principais:

* **Frameworks Fundamentais (TensorFlow, PyTorch, JAX):** Estes fornecem os blocos de construção essenciais, como operações de tensor, diferenciação automática e primitivas básicas de redes neurais. A escolha entre eles muitas vezes reflete um equilíbrio entre necessidades de pesquisa versus produção, flexibilidade versus foco em desempenho e a maturidade de seus respectivos ecossistemas. TensorFlow historicamente se destacou pela robustez em produção 8, PyTorch pela flexibilidade e adoção em pesquisa 28, e JAX pelo desempenho bruto e escalabilidade, especialmente em hardware especializado como TPUs.41
* **Bibliotecas de Rede Neural (Flax, Keras):** Estas oferecem abstrações de nível superior sobre os frameworks fundamentais para facilitar a construção de redes. Flax opera sobre JAX, mantendo um foco em desempenho e um estilo funcional 45, enquanto Keras prioriza a facilidade de uso e, com Keras 3, a compatibilidade multi-backend (TF, PyTorch, JAX).15
* **Bibliotecas de Otimização em Larga Escala (DeepSpeed, Megatron-LM):** Estas ferramentas especializadas abordam os desafios específicos impostos pela escala extrema dos LLMs modernos, como limitações de memória e necessidade de paralelismo avançado. DeepSpeed 4 introduziu otimizações de memória inovadoras (ZeRO) e paralelismo 3D, enquanto Megatron-LM 40 foi pioneiro em paralelismo de tensor e pipeline. Ambas geralmente aumentam as capacidades de um framework fundamental, mais comumente PyTorch.
* **Bibliotecas de Abstração/Ecossistema (Hugging Face Transformers, KerasHub/NLP):** Estas focam em simplificar os fluxos de trabalho do usuário, fornecendo acesso fácil a modelos pré-treinados e padronizando tarefas comuns. Hugging Face Transformers 3 tornou-se um padrão de fato para compartilhamento e uso de modelos de PLN, enquanto KerasHub e KerasNLP 15 visam simplificar o uso de modelos dentro do ecossistema Keras multi-backend.

Observam-se tendências e interdependências claras neste cenário:

* **Tendência à Interoperabilidade:** A capacidade de mover modelos entre diferentes ambientes computacionais é cada vez mais valorizada. A abordagem multi-backend do Keras 3 48 e o suporte inerente a múltiplos frameworks na biblioteca Hugging Face Transformers 3 são exemplos claros dessa tendência, oferecendo aos usuários maior flexibilidade e reduzindo o risco de dependência de um único fornecedor.
* **Ascensão das Bibliotecas de Otimização:** A própria existência e popularidade de DeepSpeed 4 e Megatron-LM 40 demonstram que os frameworks fundamentais, por si só, muitas vezes não são suficientes para treinar eficientemente os LLMs de última geração. Isso criou um ecossistema em camadas, onde bibliotecas de otimização são essenciais para alcançar a escala necessária.
* **Importância das Abstrações de Alto Nível:** O sucesso da Hugging Face 3 e a direção estratégica do Keras com KerasNLP/Hub 15 indicam uma forte demanda por ferramentas que simplifiquem fluxos de trabalho complexos e democratizem o acesso a modelos poderosos, tornando-os utilizáveis por um público mais amplo.
* **Interdependência:** O ecossistema é altamente interconectado. Bibliotecas de otimização como DeepSpeed e Megatron geralmente são construídas sobre frameworks fundamentais como PyTorch.38 Bibliotecas de alto nível como Hugging Face Transformers e Keras não apenas envolvem os frameworks fundamentais, mas também integram ferramentas de otimização (por exemplo, o Hugging Face Trainer pode usar DeepSpeed 50).

A escolha do(s) framework(s) apropriado(s) depende crucialmente dos requisitos específicos do projeto:

* **Pesquisa de Ponta e Flexibilidade:** PyTorch 28, frequentemente combinado com Hugging Face Transformers 36, continua sendo uma escolha popular devido à sua natureza dinâmica e ampla adoção na comunidade de pesquisa.
* **Desempenho Máximo e Escala (especialmente com TPUs):** JAX/Flax 41 oferece vantagens significativas, particularmente para equipes confortáveis com programação funcional e que necessitam de otimização extrema.
* **Implantação em Produção e Ecossistema Maduro:** TensorFlow 8 mantém pontos fortes com TF Serving/Lite 8, embora PyTorch esteja diminuindo a diferença.28 Keras (sobre o backend TF) oferece um caminho amigável para este fim.8
* **Treinamento de Modelos Massivos:** Invariavelmente requer a combinação de um framework fundamental (provavelmente PyTorch 38) com bibliotecas de otimização como DeepSpeed 4 ou Megatron-LM 40, muitas vezes gerenciadas através de abstrações como Hugging Face Trainer ou Accelerate.39
* **Facilidade de Uso, Prototipagem Rápida e Multi-Backend:** Keras 3 15 e Hugging Face Transformers 3 fornecem as experiências mais simplificadas para tarefas comuns, como o fine-tuning de modelos BERT pré-treinados.

Em conclusão, o cenário de frameworks para LLMs é dinâmico e caracterizado por uma crescente especialização e interconexão. Frameworks estão em constante evolução (por exemplo, Keras 3, Megatron-Core) e integração (por exemplo, DeepSpeed no HF Trainer). A decisão sobre qual ferramenta ou combinação de ferramentas usar deve ser baseada em uma avaliação cuidadosa da tarefa específica, da escala necessária, do hardware disponível, da experiência da equipe e dos trade-offs desejados entre flexibilidade, desempenho, facilidade de uso e prontidão para produção. A natureza em camadas do ecossistema permite que os usuários combinem diferentes ferramentas para construir soluções que atendam às suas necessidades únicas no campo em rápida evolução dos grandes modelos de linguagem.

#### Referências citadas

1. BERT - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/v4.32.1/model_doc/bert>
2. google-bert/bert-base-uncased - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/google-bert/bert-base-uncased>
3. Transformers - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/v4.39.0/index>
4. DeepSpeed is a deep learning optimization library that makes distributed training and inference easy, efficient, and effective. - GitHub, acessado em abril 14, 2025, <https://github.com/deepspeedai/DeepSpeed>
5. What is DeepSpeed? Features & Getting Started, acessado em abril 14, 2025, <https://www.deepchecks.com/llm-tools/deepspeed/>
6. Model Parallelism - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/v4.13.0/parallelism>
7. DeepSpeed - Microsoft Research, acessado em abril 14, 2025, <https://www.microsoft.com/en-us/research/project/deepspeed/>
8. TensorFlow - Wikipedia, acessado em abril 14, 2025, <https://en.wikipedia.org/wiki/TensorFlow>
9. TensorFlow Overview - NVIDIA Docs Hub, acessado em abril 14, 2025, <https://docs.nvidia.com/deeplearning/frameworks/tensorflow-release-notes/overview.html>
10. tensorflow2.0 - Tensorflow serving keras bert model issue - Stack Overflow, acessado em abril 14, 2025, <https://stackoverflow.com/questions/79154761/tensorflow-serving-keras-bert-model-issue>
11. TensorFlow | bert - Kaggle, acessado em abril 14, 2025, <https://www.kaggle.com/models/tensorflow/bert>
12. Classify text with BERT - TensorFlow, acessado em abril 14, 2025, <https://www.tensorflow.org/text/tutorials/classify_text_with_bert>
13. BERT - The perfect tutorial to easily use it ! - Inside Machine Learning, acessado em abril 14, 2025, <https://inside-machinelearning.com/en/bert-the-perfect-tutorial-to-easily-use-it/>
14. ML Engineer comparison of Pytorch, TensorFlow, JAX, and Flax - SoftwareMill, acessado em abril 14, 2025, <https://softwaremill.com/ml-engineer-comparison-of-pytorch-tensorflow-jax-and-flax/>
15. Keras: Deep Learning for humans, acessado em abril 14, 2025, <https://keras.io/>
16. BERT Preprocessing with TF Text - TensorFlow, acessado em abril 14, 2025, <https://www.tensorflow.org/text/guide/bert_preprocessing_guide>
17. DeepLearningExamples/TensorFlow/LanguageModeling/BERT/README.md at master - GitHub, acessado em abril 14, 2025, <https://github.com/NVIDIA/DeepLearningExamples/blob/master/TensorFlow/LanguageModeling/BERT/README.md>
18. BERT pre-training from scratch with tensorflow version 2.x - Stack Overflow, acessado em abril 14, 2025, <https://stackoverflow.com/questions/73107793/bert-pre-training-from-scratch-with-tensorflow-version-2-x>
19. Text classification with transformers in Tensorflow 2: BERT, XLNet - Atheros Learning, acessado em abril 14, 2025, <https://learning.atheros.ai/blog/text-classification-with-transformers-in-tensorflow-2>
20. Fine-tuning a BERT model | Text - TensorFlow, acessado em abril 14, 2025, <https://www.tensorflow.org/tfmodels/nlp/fine_tune_bert>
21. Fine-tuning a BERT Model - Colab, acessado em abril 14, 2025, <https://colab.research.google.com/github/dlmacedo/starter-academic/blob/master/content/courses/deeplearning/notebooks/tensorflow/fine_tuning_bert.ipynb>
22. Pretraining a Transformer from scratch with KerasHub, acessado em abril 14, 2025, <https://keras.io/keras_hub/guides/transformer_pretraining/>
23. How to increase the rank (ndim) of input of BERT keras hub layer for learning-to-rank, acessado em abril 14, 2025, <https://stackoverflow.com/questions/67557800/how-to-increase-the-rank-ndim-of-input-of-bert-keras-hub-layer-for-learning-to>
24. keras-team/keras-hub: Pretrained model hub for Keras 3. - GitHub, acessado em abril 14, 2025, <https://github.com/keras-team/keras-hub>
25. KerasHub pretrained models, acessado em abril 14, 2025, <https://keras.io/keras_hub/presets/>
26. keras/bert\_base\_multi - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/keras/bert_base_multi>
27. Tensors and Dynamic neural networks in Python with strong GPU acceleration - GitHub, acessado em abril 14, 2025, <https://github.com/pytorch/pytorch>
28. PyTorch, acessado em abril 14, 2025, <https://pytorch.org/>
29. Learning PyTorch with Examples, acessado em abril 14, 2025, <https://pytorch.org/tutorials/beginner/pytorch_with_examples.html>
30. PyTorch-Transformers, acessado em abril 14, 2025, <https://pytorch.org/hub/huggingface_pytorch-transformers/>
31. maknotavailable/pytorch-pretrained-BERT: A PyTorch implementation of Google AI's BERT model provided with Google's pre-trained models, examples and utilities. - GitHub, acessado em abril 14, 2025, <https://github.com/maknotavailable/pytorch-pretrained-BERT>
32. How to Code BERT Using PyTorch - Tutorial With Examples - Neptune.ai, acessado em abril 14, 2025, <https://neptune.ai/blog/how-to-code-bert-using-pytorch-tutorial>
33. BERT Fine-Tuning Tutorial with PyTorch - Chris McCormick, acessado em abril 14, 2025, <https://mccormickml.com/2019/07/22/BERT-fine-tuning/>
34. Google AI 2018 BERT pytorch implementation - GitHub, acessado em abril 14, 2025, <https://github.com/codertimo/BERT-pytorch>
35. Implement BERT From Scratch - PyTorch - YouTube, acessado em abril 14, 2025, <https://www.youtube.com/watch?v=v5cyVwAXR1I>
36. Transformers - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/index>
37. BERT - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/model_doc/bert>
38. DeepSpeed Model Training - Tutorialspoint, acessado em abril 14, 2025, <https://www.tutorialspoint.com/deepspeed/deepspeed-model-training.htm>
39. Get Started with DeepSpeed - Ray Docs, acessado em abril 14, 2025, <https://docs.ray.io/en/latest/train/deepspeed.html>
40. NVIDIA/Megatron-LM: Ongoing research training ... - GitHub, acessado em abril 14, 2025, <https://github.com/NVIDIA/Megatron-LM>
41. JAX: High performance array computing — JAX documentation, acessado em abril 14, 2025, <https://docs.jax.dev/>
42. jax-ml/jax: Composable transformations of Python+NumPy programs: differentiate, vectorize, JIT to GPU/TPU, and more - GitHub, acessado em abril 14, 2025, <https://github.com/jax-ml/jax>
43. JAX compatibility - ROCm Documentation - AMD, acessado em abril 14, 2025, <https://rocm.docs.amd.com/en/docs-6.3.1/compatibility/ml-compatibility/jax-compatibility.html>
44. transformers/examples/flax/README.md at main · huggingface ..., acessado em abril 14, 2025, <https://github.com/huggingface/transformers/blob/main/examples/flax/README.md>
45. flax/docs/README.md at main · google/flax · GitHub, acessado em abril 14, 2025, <https://github.com/google/flax/blob/master/docs/README.md>
46. Source code for transformers.models.bert.modeling\_flax\_bert - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/transformers/v4.9.0/_modules/transformers/models/bert/modeling_flax_bert.html>
47. Google Research examples - Flax - Read the Docs, acessado em abril 14, 2025, <https://flax.readthedocs.io/en/v0.8.3/examples/google_research_examples.html>
48. KerasCV and KerasNLP: Multi-framework Models - Journal of Machine Learning Research, acessado em abril 14, 2025, <https://www.jmlr.org/papers/volume25/24-0404/24-0404.pdf>
49. Training Overview and Features - DeepSpeed, acessado em abril 14, 2025, <https://www.deepspeed.ai/training/>
50. DeepSpeed - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/deepspeed>
51. DeepSpeed User Guide for Training — Gaudi Documentation 1.20.0 documentation, acessado em abril 14, 2025, <https://docs.habana.ai/en/latest/PyTorch/DeepSpeed/DeepSpeed_User_Guide.html>
52. transformers-deepspeed/inference/bert\_optimzation.ipynb at main - GitHub, acessado em abril 14, 2025, <https://github.com/philschmid/transformers-deepspeed/blob/main/inference/bert_optimzation.ipynb>
53. BingBertSQuAD Fine-tuning - DeepSpeed, acessado em abril 14, 2025, <https://www.deepspeed.ai/tutorials/bert-finetuning/>
54. BERT Pre-training - DeepSpeed, acessado em abril 14, 2025, <https://www.deepspeed.ai/tutorials/bert-pretraining/>
55. NVIDIA Megatron-Core, acessado em abril 14, 2025, <https://docs.nvidia.com/megatron-core/index.html>
56. User Guide - NVIDIA Docs Hub, acessado em abril 14, 2025, <https://docs.nvidia.com/megatron-core/developer-guide/latest/user-guide/index.html>
57. How does NVIDIA's Megatron-LM training example utilize model parallelism and data parallelism? - Massed Compute, acessado em abril 14, 2025, [https://massedcompute.com/faq-answers/?question=How%20does%20NVIDIA's%20Megatron-LM%20training%20example%20utilize%20model%20parallelism%20and%20data%20parallelism?](https://massedcompute.com/faq-answers/?question=How+does+NVIDIA's+Megatron-LM+training+example+utilize+model+parallelism+and+data+parallelism?)
58. Quickstart - Hugging Face, acessado em abril 14, 2025, <https://huggingface.co/docs/transformers/quicktour>
59. codota/transformers-test: Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX. - GitHub, acessado em abril 14, 2025, <https://github.com/codota/transformers-test>
60. Keras - Wikipedia, acessado em abril 14, 2025, <https://en.wikipedia.org/wiki/Keras>
61. Home - Keras Documentation, acessado em abril 14, 2025, <https://faroit.com/keras-docs/1.2.0/>
62. Transfer learning & fine-tuning - Keras, acessado em abril 14, 2025, <https://keras.io/guides/transfer_learning/>
63. KerasNLP\_ Quick start - Kaggle, acessado em abril 14, 2025, <https://www.kaggle.com/code/ahlemammar2/kerasnlp-quick-start>
64. LazurasLong/keras-nlp: Modular Natural Language Processing workflows with Keras - GitHub, acessado em abril 14, 2025, <https://github.com/LazurasLong/keras-nlp>
65. KerasCV and KerasNLP: Vision and Language Power-Ups - arXiv, acessado em abril 14, 2025, <https://arxiv.org/html/2405.20247v1>
66. Introducing Keras Hub: Your one-stop shop for pretrained models - Google Developers Blog, acessado em abril 14, 2025, <https://developers.googleblog.com/en/introducing-keras-hub-for-pretrained-models/>
67. Fine-Tune and Run Inference on Google's Gemma Model with TPU | DataCamp, acessado em abril 14, 2025, <https://www.datacamp.com/tutorial/combine-google-gemma-with-tpus-fine-tune-and-run-inference-with-enhanced-performance-and-speed>