Processamento de Alto Desempenho em Python: Uma Análise Comparativa de Bibliotecas de Aprendizado de Máquina

Bernardo Nogueira Rocha, Thiago Bortoluzzi Morais, Nicolas Flores Feijó, Andressa Assae Perri Tamara Rodrigues, Claudio Schepke

[1]Campus Alegrete – Universidade Federal do Pampa (Unipampa)
Av. Tiaraju, 810 - Ibirapuitã, Alegrete - RS, 97546-550

{bernardorocha,thiagomorais,nicolasflores,andressarodrigues}

.aluno@unipampa.edu.br, claudioschepke@unipampa.edu.br

Resumo. *Este trabalho compara os frameworks Pytorch, TensorFlow e JAX para a implementação em Python de aplicações de aprendizado de máquina focados em desempenho. Uma análise teórica das características e otimizações realizadas aponta que JAX e TensorFlow 2.x tenderão a um desempenho de throughput superior em cargas de trabalho como Transformers, enquanto o PyTorch 2.x se destaca no equilíbrio entre usabilidade e velocidade.*

1. Introdução

A Inteligência Artificial (IA) é um dos campos mais discutidos no meio acadêmico atual. Muito se deve aos avanços recentes da área, como os Transformers, que possibilitam a geração de textos, imagens e vídeos semelhantes aos produzidos por humanos. Contudo, essa tecnologia necessita de cálculos extensivos e complexos, devido às redes neurais empregadas para tal fim. Em 2020, GPT-3, um marco da arquitetura Transformers, possuía 175 bilhões de parâmetros; essa quantidade massiva de parâmetros passa por cálculos de multiplicação de matrizes e, se não for otimizada, não pode ser utilizada em larga escala com eficiência [Deep Learning Book 2026].

Para solucionar esse problema, a área de *High Performance Computing* (HPC), que estuda otimizações possíveis para tornar a solução de um problema mais eficiente em termos de tempo ou consumo de recursos (como uso de CPU/GPU ou memória), aplicou soluções como a utilização massiva de GPU's para cálculos de multiplicação de matrizes, com a adoção de interfaces de programação como CUDA, que possibilita essa transferência do cálculo para a placa gráfica.

Para aliar a praticidade de uma linguagem de alto nível ao desempenho exigido pelos grandes modelos, este trabalho foca na avaliação de bibliotecas de *High Performance Machine Learning* para a linguagem Python. Avaliamos a eficiência e a arquitetura das otimizações internas das bibliotecas, que utilizam back-ends compilados e o mapeamento de operações para o hardware acelerador (GPU) por meio de tecnologias como CUDA. Os principais *frameworks* analisados, conhecidos por suas referências em otimizações, são: Pytorch 1.X [A. Paszke et al. 2019], Pytorch 2.0 [10. 2024], TensorFlow [M. Abadi et al. 2016] e JAX (com ênfase em seu compilador JIT, o XLA) [Frostig et al. 2018]. A comparação dessas diferentes implementações e otimizações fornece uma análise crítica sobre o melhor equilíbrio entre flexibilidade e desempenho no contexto de treinamento e inferência de modelos de redes neurais.

Este trabalho está organizado da seguinte maneira: A Seção 2 apresenta as metodologias utilizadas; A Seção 3 traz uma analise das bibliotecas selecionadas. A Seção 4 apresenta uma breve discussão a respeito da verossimilhança do quadro com a realidade; Por fim, a Seção 5 realiza uma recapitulação dos resultados e achados do trabalho.

2. Metodologia

Esta seção apresenta a metodologia utilizada neste estudo. A fim de encontrar as ferramentas mais relevantes do estado da prática no contexto de *High Performance Machine Learning* em Python, uma pesquisa simples foi realizada, utilizando strings de busca como: *"Most popular frameworks for High Performance Machine Learning in Python"*. Também foram analisadas métricas como a quantidade de downloads de cada biblioteca, a partir do site PyPi stats [1] e PyPi [2], que compilam informações sobre diversos pacotes e bibliotecas disponíveis para a linguagem python. Dessa forma, chegamos a três principais candidatos para analise: TensorFlow, Pytorch e JAX.

A Tabela 1 foi elaborada com base na revisão bibliográfica, destacando as diferenças fundamentais de *design* que influenciam a desempenho e a usabilidade em Python. A tabela resultante serve como a base conceitual da comparação elaborada na pesquisa das características dos *frameworks* analisados. Os critérios chave para a comparação foram: (a) **Mecanismo de Execução Base**: Distinguir o modelo *Eager* (dinâmico) do modelo de *Grafo Estático* (compilado), que define a natureza do *overhead* e o potencial de otimização; (b) **Compilador JIT/Backend**: Identificar o motor de otimização de baixo nível (XLA, TorchInductor), pois este é o principal responsável pelo desempenho final na GPU; (c) **Filosofia de Design e Principal Trade-off**: Analisar a prioridade de cada *framework* (Ex: flexibilidade vs. desempenho de pico), fundamental para a discussão do equilíbrio entre usabilidade e HPC.

3. Revisão bibliografia

Esta seção realiza uma analise das bibliotecas Pytorch, TensorFlow e JAX, apresentando um breve contexto sobre cada ferramenta e aprofundando a discussão sobre seu design e estratégias de otimização.

TensorFlow possibilita a criação de topologias para as mais diversas variedades de redes neurais, incluindo funções especializadas para classificação de imagens, otimizadores de hiper-parâmetros, entre outros. O *design* do *framework* foi otimizado por meio de *Dataflow graph* para mapeamento dos subconjuntos e planejamento de execuções tais como concorrente, parcial e distribuída. A execução parcial executa o subconjunto do grafo necessário, otimizando o consumo de recursos; já a execução concorrente aumenta o uso de CPU/GPU por meio de execuções independentes entre si. É importante ressaltar o uso do compilador OpenXLA para acelerar operações de álgebra linear, utilizadas na computação de inferência e treinamento dos modelos criados. A respeito das otimizações feitas, é usada uma camada na linguagem de programação C que faz a comunicação entre a biblioteca em Python e a API que realiza as chamadas para implementações de Kernels para compilação em GPUs e o executor do Dataflow Graph.

[1] https://pypistats.org
[2] https://pypi.org

Tabela 1. Comparação dos Frameworks de HPC em Python

Característica	PyTorch 1.x	PyTorch 2.x	TensorFlow 2.x	JAX
Mecanismo de Execução Base	Eager (Dinâmico)	Eager + JIT	Eager + grafo estático	Funcional / funções puras
Compilador JIT / Backend	TorchScript	TorchDynamo / TorchInductor	XLA	XLA
Mecanismo de Otimização	Kernels C++/CUDA otimizados	Captura dinâmica de grafo e fusão de kernels	Grafo estático via `@tf.function`	Compilação monolítica
Filosofia de Design	Usabilidade e facilidade de debug	Desempenho com usabilidade	Escala e robustez em produção	Desempenho de pico
Principal Trade-off	Alto *overhead* de Python	Overhead inicial de compilação	Complexidade do modelo de execução	Exige programação funcional pura
Aplicações Fortes	Pesquisa rápida e prototipagem	HPC e modelos de grande escala	Produção em larga escala	GPUs e TPUs

Pytorch é uma biblioteca de alto nível (em Python) para a criação de pipelines de aprendizado de máquina e *deep learning*, com notável simplicidade na criação de topologias de redes neurais (como *Multi-layer perceptron* para realizar uma regressão complexa ou uma Rede neural Convolutiva para classificar imagens).

JAX adere ao modelo de programação funcional, que exige que as funções sejam **puras** (sem efeitos colaterais). Esse princípio é a base de seu desempenho e o que permite a otimização através do seu compilador JIT, o **OpenXLA**. Outra característica chave do JAX é a **Diferenciação Automática** (`jax.grad`), que permite a computação eficiente dos gradientes. O *framework* também provê transformações que simplificam o código de HPC, como a **Vetorização Automática** (`jax.vmap`) e o **Paralelismo de Múltiplos Dispositivos** (`jax.pmap`). O *framework* foi expandido para bibliotecas como **Flax** (para arquitetura de modelos) e **Optax** (para otimizadores otimizados) que permitem a elaboração de modelos de aprendizado de máquina de forma mais intuitiva.

4. Resultados e Discussão

Com base na análise arquitetural e na comparação teórica apresentada na Tabela 1, é possível estabelecer uma discussão preliminar e prever o comportamento dos *frameworks* nas cargas de trabalho intensivas (Transformer, CNNs).

O Desafio do Overhead do Python: O principal *trade-off* reside no gerenciamento do *overhead* do Python. O PyTorch 1.x, devido ao seu modelo *Eager*, apresenta o maior risco de latência em modelos menores, onde o tempo de computação é menor do que o tempo gasto pelo interpretador. Tanto o PyTorch 2.x (via TorchDynamo) quanto o TensorFlow 2.x (via `@tf.function`) e o JAX (*Funções Puras*) buscam resolver esse problema, convertendo o código Python em um grafo estático compilável.

Desempenho: XLA vs. TorchInductor: Espera-se que o **JAX** e o **Tensor-**

Flow 2.x apresentem desempenho superior ou similar nas cargas de trabalho mais exigentes (como as do Transformer), dada a sua base comum no compilador **XLA**. A natureza estritamente funcional (PSC) do JAX deve permitir a mais agressiva **fusão de *kernels***, conferindo-lhe uma vantagem em cenários de *throughput* máximo e otimização de VRAM. O PyTorch 2.x deve apresentar ganhos significativos em relação ao 1.x, sendo o seu desempenho dependente da eficácia do TorchInductor em fusão de *kernels* e da estabilidade do TorchDynamo na captura do grafo.

Flexibilidade e Gerenciamento de Memória: A flexibilidade do **PyTorch 2.x** em misturar código *Eager* com código compilado deve oferecer o melhor equilíbrio entre usabilidade e desempenho. Por outro lado, o **JAX** deve demonstrar a maior eficiência no uso de **VRAM** (memória da GPU) devido à otimização estática e ao foco na alocação de memória para vetores.

5. Conclusão

Este trabalho apresentou uma análise detalhada das arquiteturas de alto desempenho em Python, com foco nos *frameworks* PyTorch, TensorFlow e JAX, estabelecendo uma base teórica sólida para a compreensão das diferenças de eficiência no contexto de HPC. A transição do modelo *Eager* (1.x) de **PyTorch**, focado em usabilidade e flexibilidade, para o modelo híbrido (2.x), via **TorchDynamo** e **TorchInductor**, demonstrou o esforço em mitigar o *overhead* do Python para atingir o desempenho de HPC, preservando a facilidade de *debugging*. O *framework* **TensorFlow** mantém seu foco em **escala e produção** através do modelo de *Grafo de Fluxo de Dados* e do compilador XLA , sendo projetado para **execução distribuída** em *clusters* de máquinas. O **JAX** representa o paradigma da **desempenho de pico**, exigindo que o código siga o princípio de **Funções Puras** (PSC) para permitir ao XLA criar um *kernel* monolítico altamente otimizado.

A comparação sintetizou esses *trade-offs* arquiteturais e permitiu a previsão de que o JAX e o TensorFlow 2.x, por serem mais intrinsecamente ligados ao XLA, tenderão a um desempenho de *throughput* superior em cargas de trabalho como *Transformers*, enquanto o PyTorch 2.x deve se destacar no **equilíbrio entre usabilidade e velocidade**. A principal contribuição deste trabalho é a análise comparativa teórica e metodológica, servindo como guia fundamentado para a escolha de *frameworks* em projetos de HPC.

Referências

[10. 2024] (2024). *PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation*, ASPLOS '24, New York, NY, USA. Association for Computing Machinery.

[A. Paszke et al. 2019] A. Paszke et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.

[Deep Learning Book 2026] Deep Learning Book (2026). Capítulo 79 – Conhecendo o Modelo GPT-3. https://www.deeplearningbook.com.br/conhecendo-o-modelo-gpt-3-generative-pre-trained-transformer/.

[Frostig et al. 2018] Frostig, R., Johnson, M., and Leary, C. (2018). Compiling machine learning programs via high-level tracing.

[M. Abadi et al. 2016] M. Abadi et al. (2016). TensorFlow: A system for large-scale machine learning. https://arxiv.org/abs/1605.08695.