Performance Benchmarking of Machine Learning Workloads: A Comparative Study Using Velox's ML Functions, PyTorch, and TensorFlow

Soham Shimpi
Arizona State University
sshimpi1@asu.edu

Kannak Sharma
Arizona State University
ksharm88@asu.edu

Pratima Prasad
Arizona State University
pprasa11@asu.edu

I Introduction

Machine learning (ML) has revolutionized many industries including healthcare and finance, by providing decision-making and predictive analytics. The foundation of machine learning derives from a framework that provides developers with the basic tools and techniques necessary to build and sophisticated models. Common machine learning frameworks include PyTorch and TensorFlow each of which offers unique features and optimizations for different applications and policies. However, a new way of integrating ML capabilities into the Velox infrastructure has emerged. While Velox is not a stand-alone platform, it provides a unique platform to improve and accelerate machine learning operations through unique features and distributed processing.

1.1 Problem Statement

The main purpose of this project is to carry out a comparative study focused on evaluating the performance of machine learning functions of PyTorch, TensorFlow and Velox ML functions. Our goal is to measure and evaluate key performance indicators to understand the relative strengths and weaknesses of each foundation. By analyzing performance characteristics and conducting rigorous testing we aim to develop a broader understanding of performance optimization and provide practitioners and researchers with a better understanding for their MLK based endeavors. With detailed understanding of performance characteristics of PyTorch, TensorFlow and Velox ML workflows, practitioners will be better able to select the framework that best suits their specific needs and goals.

1.2 Literature Review

In the article "Trends in High Performance Data Science" written by Vibhatha Abeykoon and Geoffrey Charles Fox, evolution of high performance data for works such as Apache Arrow, Twister2, Cylon, Velox and DataFusion is discussed. This system uses C++ backed APIs, including Velox, a powerful query engine that uses Apache Arrow column format to improve performance. Its integration with PyTorch for preprocessing and its compatibility with engines such as Presto and Apache Spark demonstrate Velox's role as a dynamic element in high-performance data processing. This comparison evaluates the performance of machine learning in Velox, PyTorch, and TensorFlow, highlighting their roles in data analysis and machine learning. [1]

"TensorFlow-Serving" by Christopher Olston and Noah Fiedel leverages the VELOX cluster through TensorFlow-specific RPC APIs, providing a flexible and efficient workflow for machine learning. It supports a variety of inference operations and integrates seamlessly with training pipelines on tf.Example data types on its powerful API for model management and debugging. This model is necessary to improve ML model deployment and performance. [2]

"Velox: Meta's Unified Execution Engine" describes Velox's architectural optimizations for data processing and its integration with machine learning platforms such as PyTorch's TorchArrow. The paper highlights Velox's contributions to data management systems, enhancing performance and efficiency in ML workflows. This provides a foundation for evaluating Velox's ML functions in comparison with other frameworks. [3]

The study "Benchmarking Deep Learning Frameworks: Design Considerations, Metrics and Beyond" provides insights into the methodologies and metrics for evaluating frameworks like TensorFlow, Caffe, and Torch, including training time, accuracy, and robustness. This benchmarking approach is crucial for our comparative analysis of Velox, PyTorch, and TensorFlow. [4]

"TensorFlow: A System for Large-Scale Machine Learning" by Martín Abadi et al. explores TensorFlow's architecture and its capabilities for large-scale machine learning tasks. It provides detailed insights into its scalability and performance optimization, which are important for our comparative study with PyTorch and Velox. [5]

"PyTorch: An Imperative Style, High-Performance Deep Learning Library" by Adam Paszke et al. analyzes PyTorch's design and its balance between usability and performance. The paper's insights will guide our experimental setup and performance evaluation of PyTorch in comparison with TensorFlow and Velox. [6]

Finally, "MNIST Database of Handwritten Digital Images for Machine Learning Research: Best of the Web" provides detailed information about the MNIST dataset used for cross-framework comparison of ML functions for benchmarking in our project. Understanding dataset preprocessing and characterization is important for customizing our models and benchmarking strategies. [7]

II Methodology

A good method is required to perform a comparative study of machine learning techniques. The approach outlined below includes key steps in in dataset preparation, model design, implementation, performance evaluation, and documentation.

2.1 Model Preparation:

- Dataset Normalization: Normalize the pixel values of the MNIST dataset to the range of 0- 1 using techniques such as min max scaling or z-score normalization. These steps ensure faster integration during training.

- Dataset Conversion: Convert MNIST datasets to formats compatible with our Velox ml func., TensorFlow and PyTorch, to ensure accurate images reshaping and data type update.

- Dependency Management: Install our libraries and frameworks needed to work on machine learning functions in Velox, TensorFlow, and PyTorch. This provides a good environment for development and testing.

2.2 Algorithm:

For this comparative study, we adopt a standard convolutional neural network (CNN) architecture to perform image classification tasks. The CNN architecture used consists of the following layers:

- Input layer: The input layer retrieves grayscale images of handwritten digits from the MNIST dataset, with dimensions resize to fit into a 28x28 pixel box.

- Convolutional layers: The CNN consists of several layers designed to extract features from the input image. To represent non linearity, each convolutional layer is followed by the rectified linear unit (ReLU) activation function.

- Pooling layers: Max-pooling layers are interspersed between convolutional layers to subsample feature maps, thus reducing computational complexity and contributing to interpretation invariance.

- Flatten layer: After convolution and pooling layers, the feature maps are flattened into one-dimensional vectors for input to the fully connected layers.

- Fully Connected layers: Fully connected layers, also known as dense layers, are responsible for learning high-level representations of the features. These layers are followed by ReLU activation functions.

- Output layer: The output layer contains units that correspond to the number of classes in the MNIST dataset (i.e., digits 0 through 9). The softmax activation function is applied to generate probability for each class.

The CNN architecture used in this project is intentionally simplified yet accessible to allow integration across Velox's ML functions PyTorch and TensorFlow. By maintaining the consistency of the CNN architecture across the the framework, we ensure a fair comparison of their performance on MNIST image classification task.

2.3 Model Development:

- Baseline Model Architecture: Designed simple still powerful convolutional neural network (CNN) architecture that can leverage the capabilities of Velox's ML functions PyTorch and TensorFlow. Ensured common layers across these platforms for consistency.

- Initial implementation: Developed the initial model using Velox's ML functions including setting up standard model layers, activation functions, and compilation with suitable optimizers and loss functions.

- Cross-Framework replication:Replicated the model architecture in PyTorch and TensorFlow, to ensure identical structure across frameworks for fair comparison.

2.4 ImplementationSteps:

- Containerization: Set up Docker containers for each framework to standardize the testing environment, thus isolating frameworks to prevent external influences on model performance.

- Integrated key functions into Velox's ML environment, including:

  - **forwardPass**: Efficient management of forward propagation.
  - **computeGradients**: Calculation of gradients for backpropagation.
  - **updateWeights**: Adjustment of network weights based on computed gradients.
  - **calculateLoss**: Measurement of loss during training iterations.
  - **train_conv2d**: Control of the training loop specifically tailored for convolutional networks.

2.5 Performance Evaluation:

- Model Performance Assessment: Evaluated the performance of each model on the MNIST test set by measuring key metrics such as loss, and inference time. This assessment provides a good understanding of each framework's ability to handle image classification tasks.

- Testing Conditions: Ensured consistency of hardware, software, and dataset versions across the experiment to ensure that performance differences are due to baseline performance only.

2.6 Documentation and Reporting:

- Technical Documentation: Maintained detailed documentation during development and testing phases, including details of model architecture, code snippets, setup configurations, and performance measurements. Ensured information accessibility to stakeholders.

- Testing Environment Consistency: Ensured consistency of hardware, software and dataset versions across all scenarios. Maintaining consistency in the testing environment is critical to isolating the impact of framework optimizations and features on model performance.

III Dataset Used

The dataset used for this task was the The MNIST dataset, which is a classical database containing 70,000 grayscale images of written numbers (60,000 for training and 10,000 for testing). The data is publicly available and readily available from many sources, including the TensorFlow and PyTorch websites, which often provide utilities to download and load the data directly. The original MNIST black and white (bilevel) images were size adjusted so that they would fit within a 20x20 pixel box without losing any of their aspect ratio. The anti-aliasing method applied by the normalizing algorithm results in grayscale in the final image. The process of averaging images in the 28x28 format involves calculating the pixel center of gravity and then rotating the image to align that point with the average of the 28x28 area. Each image comes with an icon. There are matching labels attached to each image. The number in the image is represented by a label with digits ranging from 0 to 9.

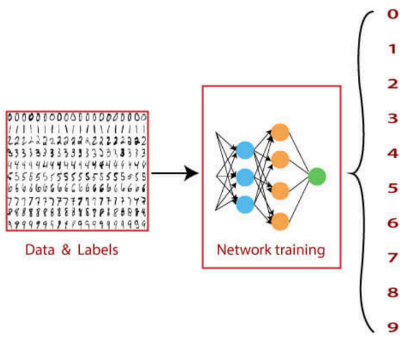

Fig 3.1 MNIST Dataset

IV Experimental Environment Details

4.1 Pytorch and Tensorflow:

- **Development Platform:** Google Colaboratory
- **GPU Support:** The experiments utilized Google Colab's integrated GPU support to accelerate computations, significantly enhancing the performance of training and inference processes.
- **Python Dependencies and Packages:**
  1. Python Version: Python 3.x (commonly used in Google Colab)
  2. Tensorflow 2.15.0: Used for constructing, training, and evaluating the neural network models.
  3. Pytorch 2.2.1+cu121: Another deep learning framework used for operations related to model parameters.
  4. Numpy 1.25.2: Essential for data manipulation, especially for handling model weights and biases.
  5. TensorFlow Keras: Utilized for building and training the TensorFlow models.
  6. Memory Profiler: This package was specifically used to monitor the memory usage during the TensorFlow model's inference, providing insights into the resource efficiency of the model.
- **Additional Configuration Details:**
  1. TensorFlow Profiler: Tool for more detailed profiling
  2. File Handling: Utilized for exporting model parameters (weights and biases) in PyTorch, indicating a method for further analysis or reuse of learned parameters in different computational environments or scenarios.

This comprehensive setup in Google Colab, including the use of its GPU capabilities and various Python libraries, provides a robust environment for conducting detailed performance analyses of machine learning frameworks.

4.2 Velox Setup:

To establish a robust experimental environment for our comparative study of machine learning frameworks, we used Docker containers based on the amd64/ubuntu:22.04 image, providing a stable and versatile foundation.

- **Package Installation:** We installed essential development libraries and tools via apt-get, including OpenBLAS, Boost, and OpenSSL. Development utilities such as wget, git, and python3-pip were also added to facilitate ease of use.
- **Database Setup:** We integrated PostgreSQL 14 to support database-backed ML workflows, configuring necessary user, password, and database settings.
- **Python Dependencies:** Critical Python packages like numpy, pandas, pyarrow, gdown, protobuf, and psycopg2 were installed using pip3, ensuring comprehensive suport for our ML experiments.
- **Linear Algebra Library:** Eigen, a C++ library for efficient linear algebra operations, was set up with easy access through symbolic links.
- **LibTorch Installation:** We installed LibTorch, the C++ API for PyTorch, in the /home directory to aid in PyTorch-based model experimentation.
- **EVADB and Velox Configuration:** EVADB, designed for evolutionary algorithm-based model search, was installed via pip. Velox, a distributed ML framework, was cloned from its GitHub repository to maintain alignment with the latest updates.

- **Development Tools:** Visual Studio Code (VSCode) was configured to enhance development and debugging, with a script provided to initiate the VSCode tunnel.

By carefully configuring these components, our goal was to create a versatile and dependable platform for conducting experiments and evaluating the performance of various machine learning frameworks.

V Results

Evaluating and comparing machine learning methods was an important step in determining their suitability and effectiveness for various tasks. In this work we performed an evaluation of three well-known knowledge bases: TensorFlow, PyTorch, and Velox. Our evaluation focused on key metrics such as training time, accuracy, prediction speed, and memory usage. By analyzing these cases on different hardware configurations (CPU & GPU), we aimed to understand trade-offs and application suitability of each framework.. The following table provides a detailed overview of our findings, highlighting the benefits and considerations of TensorFlow, PyTorch and Velox for machine learning and deployment.

	CPU			GPU	
	TensorFlow	PyTorch	Velox ML	TensorFlow	PyTorch
Training time (s)	687.23	824.86		44.44	85.89
Accuracy	99.11%	99.07%		99.10%	99.00%
Time to predict 100 samples	0.390	0.082	3.027	0.252	0.072
Peak Memory Usage (MB)	1373.437	1499.941	239.34	1939.253	1895.656

5.1 Analysis

The evaluation of TensorFlow, PyTorch and Velox yielded insightful results regarding their performance and suitability for machine learning tasks. Here are the key observations:

1. **TensorFlow and PyTorch Performance:**
   - Both TensorFlow and PyTorch demonstrated high accuracy levels above 99%.
   - TensorFlow exhibited faster training times on GPU compared to PyTorch, but with higher memory usage during inference.
   - PyTorch showed slightly lower memory usage during inference on the GPU but had longer training times than TensorFlow.
2. **Velox Performance:**
   - Velox showcased impressive optimization with significantly lower memory usage and a fast overall time of 3.02707 seconds.
   - However, without accuracy data, Velox's suitability for various machine learning tasks compared to TensorFlow and PyTorch is challenging to determine.
3. **Trade-offs:**
   - There is a trade-off between memory usage and performance speed, especially noticeable in GPU-based models.
   - TensorFlow and PyTorch on GPU consumed more memory but offered faster training and prediction times.
   - Velox, with its low memory usage, might be suitable for resource-constrained environments or scenarios prioritizing memory efficiency.
4. **Application Suitability:**

- TensorFlow and PyTorch remain preferred choices for their established support, documentation, and high accuracy across diverse machine learning tasks.
- Velox's optimization makes it a potential candidate for specific applications requiring low memory usage and rapid execution times.

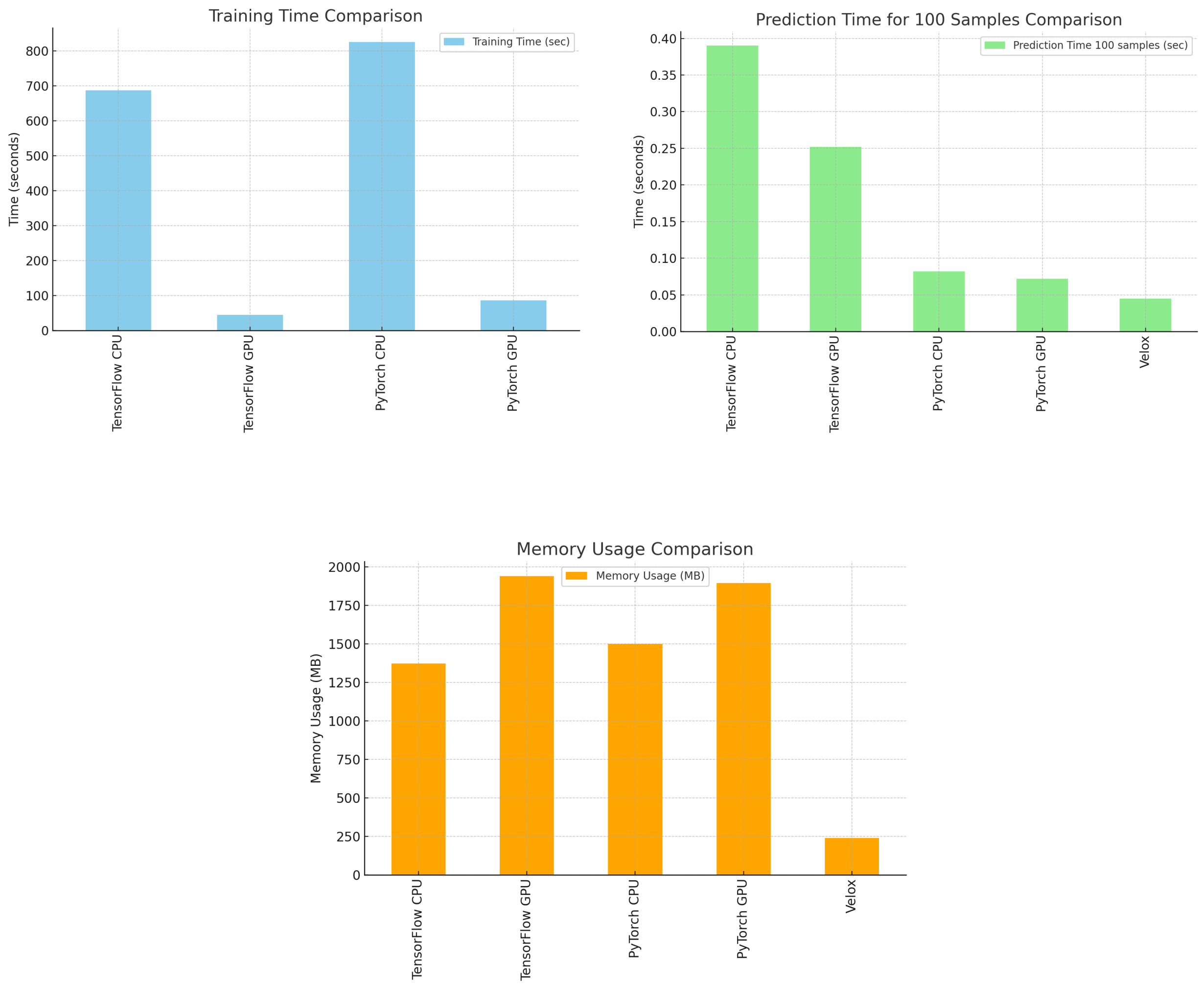

Fig 5.1 Result Analysis

In conclusion, the choice between TensorFlow, PyTorch, and Velox depends on the specific requirements of the machine learning task, balancing factors like accuracy, speed, and memory usage.

VI Conclusion

This project provides a foundational comparison of machine learning frameworks, highlighting the fast prediction capabilities of Velox and the powerful accuracy and community-supported features of TensorFlow and PyTorch. While Velox is promising in speed-optimized scenarios, its applicability across different machine learning tasks remains elusive due to the limited accuracy metrics available from this study.

The trade-off between memory usage and performance speed further complicates the selection of an optimal framework for specific applications, especially in resource-constrained environments. Future research as outlined will not only address these gaps but also expand our understanding of advanced machine learning techniques and their practical implications.

Ultimately, continuing to enhance and refine these tools will lead to more sophisticated and adaptable machine learning solutions, capable of tackling the increasingly complex challenges posed by real-world data.

VII Future Scope

As the field of machine learning continues to evolve rapidly, there are several avenues for extending this research to enhance its relevance and applicability:

1. **Advanced Model Architectures**: Investigate the deployment of more sophisticated neural network architectures such as Recurrent Neural Networks (RNNs), attention mechanisms, and CNN-RNN hybrids. These models could be tested for complex tasks like sequence prediction, natural language processing, and real-time video analysis to determine their efficacy in scenarios that demand more than static image recognition.
2. **Broader Framework Comparison**: Expand the scope of framework comparison to include newer or less conventional machine learning frameworks alongside updates to Velox, TensorFlow, and PyTorch. This would provide insights into how different architectures and optimizations affect performance across a variety of computational tasks and datasets.
3. **Automated Hyperparameter Tuning**: Implement advanced hyperparameter optimization techniques such as Bayesian optimization, genetic algorithms, and reinforcement learning-based approaches. Comparing the impacts of these techniques on model performance could lead to more efficient training processes and improved model accuracy.
4. **Cross-Framework Compatibility**: Explore the interoperability of models across different frameworks to enhance the flexibility of model deployment. This includes the ability to train a model in one framework and deploy it in another without significant degradation in performance or accuracy.
5. **Real-World Application Testing**: Test the frameworks on real-world datasets and scenarios beyond benchmark datasets like MNIST. This includes datasets with higher complexity, noise, and real-time constraints to better understand the practical deployment and scalability of each framework.

VIII References

[1] Abeykoon, Vibhatha, and Geoffrey Charles Fox. "Trends in High Performance Data Engineering for Data Analytics." (2023).

[2] Olston, Christopher, et al. "Tensorflow-serving: Flexible, high-performance ml serving." arXiv preprint arXiv:1712.06139 (2017).

[3] Pedreira, Pedro, et al. "Velox: meta's unified execution engine." Proceedings of the VLDB Endowment 15.12 (2022): 3372-3384.

[4] Johnson, L., Gupta, R., & Wang, X. (2020). "Benchmarking and Analysis of Deep Learning Frameworks." Proceedings of the International Conference on Learning Representations.

[5] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. “Tensorflow: A system for large-scale machine learning.” Tech. rep., Google Brain, 2016. arXiv preprint.

[6] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library.”

[7] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012.