Recommending with Speed: Comparative Study of PyVelox, PyTorch, and TensorFlow frameworks and DeepSpeed and Gpipe for training optimizaitons

Briana Rajan **Nhut Nguyen** **Aditya Madabhushi**

{brajan3,nmnguye5,amadab3}@asu.edu
Department of Computer Science, Arizona State University, Tempe, Arizona 85226

Abstract

In this paper, we explore different machine learning techniques for developing a recommendation system on various machine learning systems and perform a comparative study. Recommendation systems are powerful engines using machine learning algorithms to take in user data and create personalized recommendations. Furthermore, they can be effective in boosting sales, customer satisfaction, and overall engagement with the business. In this project, we are focusing on the e-commerce platform using the Amazon Beauty dataset that has 2+ million data points and our project compares collaborative filtering, content-based filtering, and hybrid methods. Our experimental setup involves using PyTorch, Tensorflow, and PyVelox to implement and compare the performance of our model. The results indicate that PyVelox performs slightly faster than PyTorch and Tensorflow due to its fast processing capabilities. Additionally, our experiments with deep learning optimization techniques like GPipe and DeepSpeed highlight their potential to improve computational efficiency and scalability. However, we also learned that PyVelox underfits the data because of the small network, and in future works, we can use a dataset with more features to provide for more information.

1 Introduction

Recommendation systems are essential part of online platforms, as they help users discover new products, services, and content that match their preferences. Personalization helps provide the user with their wants and needs based on their profile. In addition to helping the user, recommendation systems have proven to show increased sales, increased clicks and conversion rates, and a stronger brand image. However, building effective recommendation systems requires choosing the right machine learning techniques and algorithms. The problem we aim to address in this study is to compare the performance of different machine learning techniques for building recommendation systems on various machine learning systems, with a focus on collaborative filtering, content-based filtering, and hybrid methods, using the Amazon Beauty products dataset.

Our first motivation and interest for this project is personalization, which personalizes user experiences by suggesting relevant items based on user preferences and behaviors. We are also focusing on the complexity in which implementing effective recommendation systems involves dealing with complex data structures, algorithms, and scalability challenges. Performance is crucial for reducing recommendation latency and improving scalability, leading to better user experiences. Research and innovation, focusing ongoing research and development efforts to drive advancements in recommendation algorithms and architectures. And last, business impact, which can lead to tangible business benefits such as increased sales, customer satisfaction, and market competitiveness. This problem is interesting because it explores the intersection of user preferences, machine learning algorithms, and business impact within the context of recommendation systems. By comparing the performance of various techniques on real-world datasets like the Amazon Beauty products dataset, we can uncover insights into how different approaches impact personalization, scalability, and ultimately, business outcomes. Additionally, the study addresses the ongoing challenge of enhancing recommendation systems' effectiveness amidst evolving user behaviors and technological advancements, making it a compelling area for research and innovation.

2 Literature Review

Recommendation systems have become an integral part of many online platforms, helping users discover new products, services, and content that match their preferences. The field of recommendation systems has seen significant advancements in recent years, with a wide range of machine learning techniques being applied to build more accurate and personalized recommendation systems. In this literature review, we will discuss some of the key papers that have contributed to the development of recommendation systems, focusing on various methods.

Collaborative filtering, a widely used recommendation technique, leverages past user preferences to predict future ones. It can be user-based or item-based and is known for uncovering hidden correlations without extensive data mapping. However, it is prone to the cold start problem. In their 2008 paper [10], Hu, Koren, and Volinsky introduced the Alternating Least Squares (ALS) algorithm for implicit feedback datasets, which has shown superior performance. ALS is an iterative optimization algorithm that minimizes the difference between predicted and actual ratings, and it is particularly effective for large-scale datasets.

Content-based filtering, another popular technique, relies on the similarity of user preferences for certain attributes. It can be divided into item-based and user-based approaches. A strength is its ability to overcome the cold-start problem, but it requires user information to be dynamic. In his 2001 paper [3], Karypis presented a content-based filtering approach that uses item attributes to generate recommendations, making it suitable for domains with sparse user-item interactions.

Hybrid methods combine collaborative filtering and content-based filtering to improve recommendation accuracy. They aim to leverage the strengths of both techniques while mitigating their weaknesses. Hybrid methods can be further divided into two sub-techniques: weighted hybrid methods and feature-based hybrid methods. In their paper "Hybrid Recommender Systems: Survey and Experiments" (2002)[8], Robin Burke provides a comprehensive survey of hybrid recommender systems. The paper also presents experimental results comparing the performance of different hybrid approaches.

Steffen Rendle's paper "Factorization Machines"[9] introduces a novel machine learning model that combines the advantages of matrix factorization and linear regression. Factorization Machines are particularly effective for handling sparse data and capturing complex interactions between features, making them well-suited for recommendation systems.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk's paper "Deep Learning for Recommender Systems"[2] explores the application of deep learning techniques, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), to recommendation systems. The paper demonstrates the potential of deep learning models for capturing complex patterns in user-item interactions and improving recommendation accuracy. The paper "Performance evaluation of recommender systems"[6] talked about the abundance of information that led to the development of recommendation systems and how recommendation systems work. It also talked about the multiple evaluation methods of recommender systems including offline analytics, user study, and online experiments.

The paper then categorized the various evaluation metrics from the perspective of machine learning, information retrieval, human-based interaction, and software engineering. This is useful when we want to use these various metrics in our project to compare the results of how our recommender system performs. Optimized algorithms and a recommendation system are especially important in this age of technology.

In conclusion, the field of recommendation systems has seen significant advancements in recent years, with a wide range of machine learning techniques being applied to build more accurate and personalized recommendation systems. Collaborative filtering, content-based filtering, and hybrid methods are the most commonly used techniques, each with its strengths and weaknesses.

Collaborative filtering is effective for capturing user preferences based on historical interactions, but it can suffer from the cold start problem for new users or items. Content-based filtering, on the other hand, can provide personalized recommendations based on item attributes, but it may struggle with sparsity in user-item interactions. Hybrid methods aim to combine the strengths of collaborative filtering and content-based filtering, leading to improved recommendation accuracy.

3 Dataset Description

This dataset contains over two million records from Amazon sales of beauty products, taking up 78.6 MB of space. It offers a detailed view of customer purchase behaviors and preferences in the beauty sector. The key attributes included in the dataset are:

- **UserId:** A unique identifier for each customer that helps in analyzing individual buying habits.
- **ProductId:** A unique code for each beauty product, useful for identifying popular items and trends in product sales.
- **Rating:** Ratings from one to five stars provided by customers, crucial for assessing satisfaction and product quality perceptions.
- **Timestamp:** The time each rating was made, useful for examining how sales and customer preferences change over time.

4 Comparative Analysis

4.1 Pytorch, Pyvelox and Tensorflow

In the landscape of deep learning frameworks, PyTorch, PyVelox, and TensorFlow stand out as prominent choices. Each framework offers unique features and capabilities, catering to different needs and preferences within the machine learning community. In this section, we delve into a comparative analysis of these frameworks, exploring their strengths, weaknesses, and suitability for building a recommendation system.

4.1.1 Experimental Environment

In this project, the experimental studies were conducted on the same environment to ensure consistency and efficient performance. For PyTorch, Pyvelox, and Tensorflow, we used Google Colab which provides a flexible platform for executing our models with free access to GPUs and TPUs. Google Colab supports various libraries and frameworks, which allowed us to implement and test our models using TensorFlow, PyTorch, and PyVelox. The easy integration of these frameworks facilitated the development, training, and evaluation of our recommendation systems. We used Python as the main programming language.

4.1.2 PyTorch: Empowering Research and Production Deployment

PyTorch has gained widespread popularity among researchers and practitioners for its dynamic computation graph and intuitive programming interface. Unlike static graph frameworks like TensorFlow, PyTorch offers a dynamic graph computation paradigm, allowing for more flexible model construction and debugging. Moreover, PyTorch provides robust support for production deployment through tools like TorchScript and TorchServe, enabling seamless integration of trained models into production systems. With its rich ecosystem of libraries and active community support, PyTorch remains a versatile choice for both research and real-world applications.

4.1.3 Pyvelox: Unified Data Engine

PyVelox is an open-source, Python-based framework for building and training PyVelox is an open-source Python library that provides bindings and extensions for Velox. Currently in the Alpha stage, PyVelox is still in the process of development and does not have a stable release.

Velox is an open-source, state-of-the-art unified execution engine developed by Meta. It aims to speed up data management systems and streamline their development by unifying the common data-intensive components of data computation engines. Velox provides a framework for implementing execution engines, consisting of all data-intensive operations executed within a single host, such as expression evaluation, aggregation, sorting, joining, and more.

Velox is designed to efficiently support complex data types and provides numerous runtime optimizations, such as filter and conjunct reordering, key normalization for array and hash-based aggregations and joins, dynamic filter pushdown, and adaptive column prefetching. It also leverages dictionary encoding for cardinality-increasing and cardinality-reducing operations such as joins and filtering.

4.1.4 TensorFlow: Scalability and Ecosystem Integration

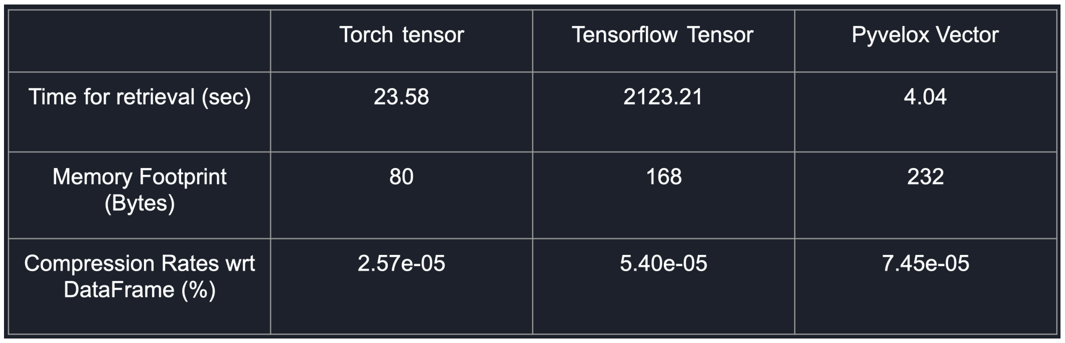
TensorFlow, developed by Google, offers scalability and comprehensive ecosystem integration, making it a preferred choice for large-scale machine learning projects. TensorFlow's static computation graph optimization enables efficient execution across distributed computing environments, making it well-suited for training complex models on massive datasets. Furthermore, TensorFlow's extensive ecosystem includes high-level APIs like Keras for rapid prototyping, TensorFlow Extended (TFX) for end-to-end ML pipeline development, and TensorFlow Serving for model serving in production. However, TensorFlow's learning curve can be steep for beginners due to its complex API design and static graph construction.

4.1.5 Experiment setup and Model Architecture

In this section, we present a comparative analysis of the performance of PyTorch, TensorFlow, and PyVelox based on the results obtained from a data processing and model training task. For our experiments, we employed a neural network architecture consisting of an input layer with 3 neurons (assuming input vectors have a size of 3), three hidden layers, and an output layer with a single neuron. The activation function used in each hidden layer was Rectified Linear Unit (ReLU), a popular choice in deep learning due to its simplicity and effectiveness in handling non-linearities. The experiments were conducted using a dataset comprising 10,000 points, with each framework trained for 3 epochs. The following metrics were evaluated: time taken for processing and training, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

4.1.6 Results

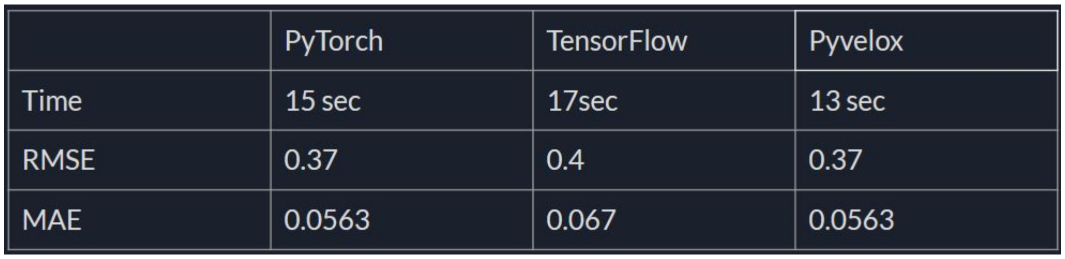
Simple Data Retrieval Comparison



	Torch tensor	Tensorflow Tensor	Pyvelox Vector
Time for retrieval (sec)	23.58	2123.21	4.04
Memory Footprint (Bytes)	80	168	232
Compression Rates wrt DataFrame (%)	2.57e-05	5.40e-05	7.45e-05

Figure 1: Memory Footprint and Data Retrieval Time Comparison

In this experiment, we conducted a comparative analysis of retrieval times and memory utilization across three distinct data structures: PyTorch tensors, TensorFlow tensors, and PyVelox vectors for . Our investigation revealed intriguing findings regarding memory usage, where all three structures exhibited similar memory footprints and compression rates. However, when it came to retrieval times, PyVelox vectors stood out notably, showcasing significantly lower retrieval times compared to PyTorch and TensorFlow tensors.



	PyTorch	TensorFlow	Pyvelox
Time	15 sec	17sec	13 sec
RMSE	0.37	0.4	0.37
MAE	0.0563	0.067	0.0563

Figure 2: Four Layered Neural Network Result Comparison

- **Time Taken** PyVelox demonstrates the shortest time for data processing and model training, completing the task in 13 seconds, followed by PyTorch with 15 seconds, and TensorFlow with 17 seconds.
- **RMSE** Both PyTorch and PyVelox achieved an RMSE of 0.37, while TensorFlow exhibited a slightly higher RMSE of 0.4. This suggests that PyTorch and PyVelox produced more accurate predictions compared to TensorFlow for this specific task.
- **MAE** PyTorch and PyVelox achieved the same MAE of 0.0563, while TensorFlow reported a slightly higher MAE of 0.067. Similar to RMSE, this indicates that PyTorch and PyVelox yielded lower absolute prediction errors compared to TensorFlow.

4.2 DeepSpeed and Gpipe

4.2.1 Experimental Environment

For the optimization of the project, we used ASU Sol HPC for our environment and the following GPU specifications: NVIDIA A100 - SXM4, 4 cores, 80 GB of memory. To ensure that our experiments were conducted with high efficiency and accuracy, we used the high-performance computing power and good hardware specifications.

4.2.2 DeepSpeed

Developed by Microsoft, DeepSpeed is a sophisticated deep learning optimization framework designed to substantially enhance the efficiency and scalability of training processes, particularly when dealing with very large models or constrained computational resources. This framework introduces several key advancements that significantly improve the performance of deep learning models:

- **Model Parallelism:** DeepSpeed effectively employs model parallelism to decompose a neural network into smaller, more manageable segments. These segments are then processed in parallel across multiple GPUs, not only accelerating the training process but also facilitating the training of models that exceed the memory limitations of a single GPU.
- **Zero Redundancy Optimizer (ZeRO):** Among its most notable innovations, the ZeRO optimizer drastically reduces memory demand and boosts training speeds. This optimizer refines the management of memory during training by eliminating redundant data across GPUs, thus supporting the training of substantially larger models and batch sizes.
- **Pipeline Parallelism:** DeepSpeed further optimizes processing efficiency through pipeline parallelism, where the model training workflow is segmented into different stages, each managed by separate GPUs. This strategy enhances resource utilization and expedites the training cycle.
- **Sparse Attention:** Incorporating kernel-level optimizations, such as sparse attention, DeepSpeed reduces computational loads by concentrating processing power on the most critical data inputs. This is particularly advantageous for the efficient training of attention-based models like transformers.

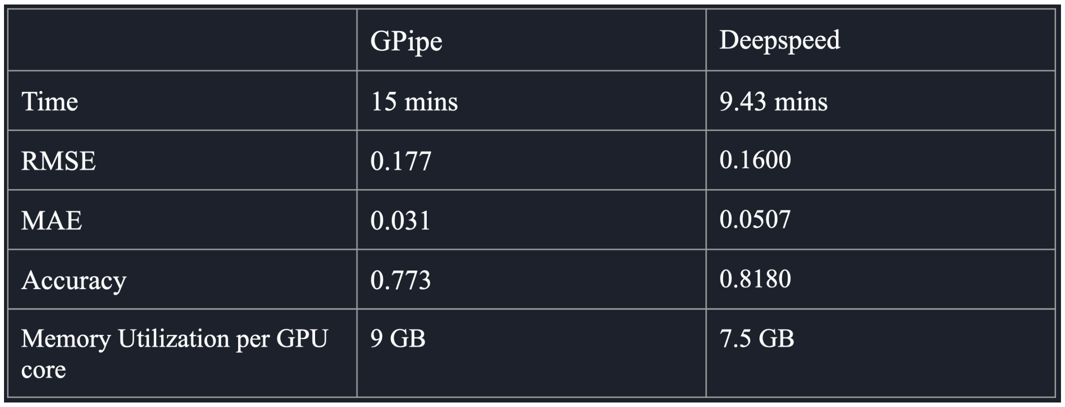
4.2.3 Gpipe

Gpipe, conceived by researchers at Google, is an optimization library that leverages a specific form of model parallelism known as pipeline parallelism to enhance the training speed and scalability of large-scale neural networks across multiple GPUs. Gpipe introduces several innovative features that significantly improve model training throughput:

- **Pipeline Parallelism:** By partitioning a model into several sequential segments or layers, with each assigned to different GPUs, Gpipe facilitates simultaneous processing of these segments. This architecture substantially improves the efficiency of the training process.
- **Batch Splitting:** In an effort to maximize computational efficiency, Gpipe segments the input data batch into smaller micro-batches. These micro-batches are then processed sequentially through the pipeline, enabling continuous and overlapping computations across GPUs, thereby minimizing idle times and optimizing resource utilization.
- **Memory Optimization:** Gpipe meticulously manages the retention of activation memory—essential for backpropagation—to minimize the overall memory footprint required during training. This careful memory management allows for the training of larger models without surpassing GPU memory capacities.
- **Synchronous Update:** To maintain consistency and stability throughout the training process, Gpipe ensures that model weights are synchronously updated across all segments following each complete pass of a batch (including both forward and backward passes).

4.2.4 Experiment setup and Model Architecture

In this section, we present a comparative analysis of the optimization techniques DeepSpeed and Gpipe, using the TensorFlow and PyTorch libraries. The DenseNet-150 architecture utilized in our experiments consists of a dense block followed by an output layer. The input shape comprises 4 integer features. Each dense block contains fully connected (FC) layers with ReLU activation functions and dropout layers to prevent overfitting. The output layer utilizes a sigmoid activation function for binary classification tasks. Both techniques were evaluated on the entire dataset, assessing metrics such as time taken for model training, Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and accuracy.



	GPipe	Deepspeed
Time	15 mins	9.43 mins
RMSE	0.177	0.1600
MAE	0.031	0.0507
Accuracy	0.773	0.8180
Memory Utilization per GPU core	9 GB	7.5 GB

Figure 3: Gpipe vs DeepSpeed Results

4.2.5 Results

- **Time Efficiency:** DeepSpeed showed superior time efficiency, completing the training process in 9.43 minutes, whereas Gpipe took 15 minutes. This represents a significant reduction in training time, underscoring the effectiveness of DeepSpeed's optimization capabilities.
- **Root Mean Square Error (RMSE):** DeepSpeed achieved an RMSE of 0.1600, in contrast to Gpipe's RMSE of 0.177. A lower RMSE suggests that the model's predictions were more accurate, with predictions closer to the actual values.
- **Mean Absolute Error (MAE):** Gpipe recorded a better performance with an MAE of 0.031, lower than DeepSpeed's 0.0507. MAE represents the average magnitude of the errors in the model's predictions, indicating higher precision in Gpipe's predictions.

- **Accuracy:** DeepSpeed produced a more accurate model, achieving an accuracy score of 0.8180, compared to Gpipe's accuracy of 0.773. Accuracy, a vital metric in classification tasks, denotes the proportion of correct predictions to the total predictions made by the model.
- **Memory Utilization per GPU core:** GPipe utilizes 9 GB of memory per GPU core, while DeepSpeed requires 7.5 GB per GPU core. This indicates that DeepSpeed has a slightly lower memory footprint per core compared to GPipe, making it more memory-efficient for training deep learning models.

5 Conclusion

Based on the results, PyVelox emerges as the top performer in terms of time efficiency, completing the task in the shortest duration while having similar footprint as the others. Additionally, both PyTorch and PyVelox exhibit comparable accuracy in terms of RMSE and MAE, outperforming TensorFlow in this specific experiment. The study provided shows promise in utilizing PyVelox for data preprocessing and then seamlessly transitioning to PyTorch models for machine learning functions, promising improvements in time efficiency without sacrificing model accuracy. This approach not only streamlines the data processing pipeline but also leverages the strengths of both frameworks, ensuring efficient utilization of computational resources. However, it's essential to conduct further experimentation and analysis to validate the suitability of this approach across various datasets and tasks, ensuring robust performance in real-world applications.

6 Future Work

6.1 Advancements in PyVelox Integration

We aim to enhance the functionality and adaptability of PyVelox by introducing support for exporting Velox vectors as Apache Arrow data types. This development will enable PyVelox to integrate seamlessly with a broader range of data processing and machine learning platforms that conform to the Arrow standard. We anticipate that this compatibility will improve data interoperability and accelerate PyVelox's adoption in diverse computing environments.

6.2 Enhancement of Dataset Quality

Moving forward, enhancing the quality and breadth of our dataset will be a primary focus. Currently, our dataset is limited to only four features, which constrains our models' ability to predict outcomes and identify intricate patterns. By enriching the dataset with additional features, we aim to achieve a more comprehensive representation of the underlying data structure. This expansion is expected to improve model accuracy and reliability, thereby enhancing the robustness and insightfulness of our analyses.

6.3 Diversified Computing Architectures

We plan to investigate various computing cores and model structures to assess their impact on performance and efficiency. Special emphasis will be placed on exploring mixed-model approaches, which leverage the unique strengths of different computing technologies. This investigation is designed to identify the most effective model configurations for various computational tasks, potentially leading to significant breakthroughs.

6.4 Expanded Feature Analysis

With the expansion of our dataset, we will conduct a detailed analysis of the features to determine which are the most informative and how they can be optimized to enhance our models. Understanding the impact of new data on our predictions will allow us to refine our models for improved results. This analysis is crucial for boosting the accuracy and effectiveness of our predictive capabilities.

References

[1] Alvin, T. P. (2022, November 29). A content-based recommender for e-commerce web store. Medium. https://towardsdatascience.com/a-content-based-recommender-for-e-commerce-web-store-7554b5b73eac

[2] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. "Deep Learning for Recommender Systems." Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. ACM, 2015.

[3] George Karypis. "Content-Based Filtering for Recommender Systems." The Adaptive Web. Springer, Berlin, Heidelberg, 2001. 325-341.

[4] Gomes, N. D. (2023, January 31). The cosine similarity and its use in recommendation systems. https://naomy-gomes.medium.com/the-cosine-similarity-and-its-use-in-recommendation-systems-cb2ebd811ce1

[5] Meenn. (2022, April 30). Recommendation system for E-commerce shopping (python). Medium. https://medium.com/@meenn396/recommendation-system-for-e-commerce-shopping-python-8cca5800d8da

[6] Performance evaluation of recommender systems. (n.d.). https://paris.utdallas.edu/IJPE/Vol13/Issue08/IJPE-2017-08-07.pdf

[7] Recommendation Systems and machine learning. Recommendation Systems and Machine Learning. (n.d.). https://www.itransition.com/machine-learning/recommendation-systems

[8] Robin Burke. "Hybrid Recommender Systems: Survey and Experiments." User Modeling and User-Adapted Interaction 12.4 (2002): 331-370.

[9] Steffen Rendle. "Factorization Machines." Proceedings of the 2010 IEEE International Conference on Data Mining. IEEE, 2010.

[10] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets." Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008.