Accelerating Model Training: Performance Antipatterns Eliminator Framework

Ravi Kumar Singh [1] Mayank Mishra [1] Rekha Singhal [1]

[1]TCS Research

Introduction and Motivation

An ML/DL pipeline spends most of the time in data preparation, as shown in figure 1, starting from source-specific, task-specific, and model-specific to inference-specific processing. The training-specific data preparation consists of task and model-specific components.Ignoring the specific training of data preparation is a common mistake made by both data scientists and developers, and it often creates significant obstacles during the process of model training. In the following section, we will elaborate on this issue with supporting evidence.

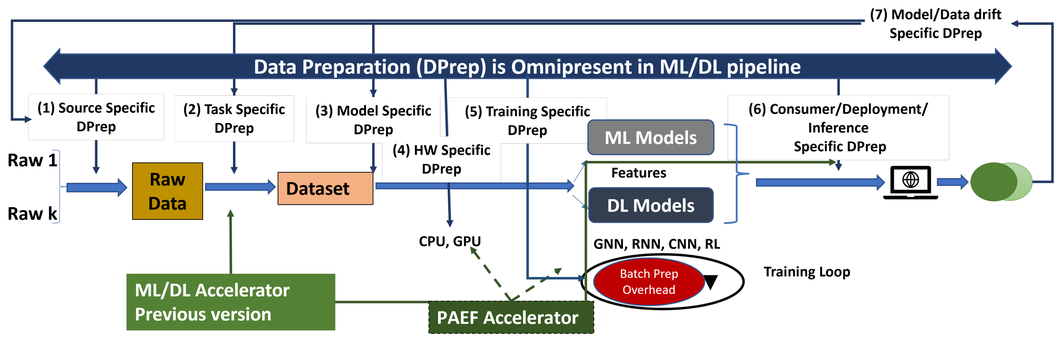
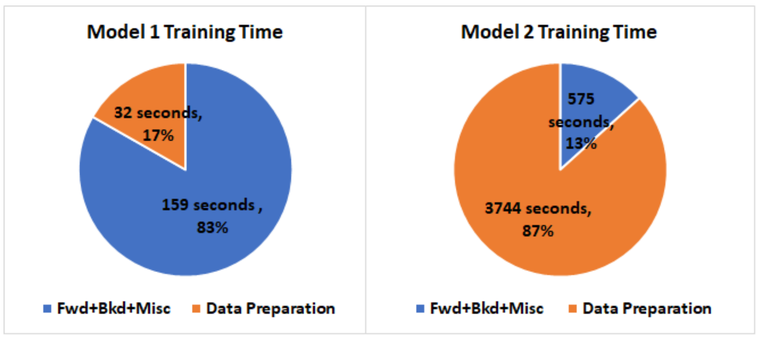


Figure 1. Data Preparation acceleration in ML/DL pipeline



As shown in figure two industrial recommendation models spend 17% and 87% time, respectively, on data preparation on forward and backward passes. As it will affect the training process,

A data scientist might not be aware of performance antipatterns caused by data movement and transformations between CPU and GPU during training. These performance antipatterns significantly reduce the training pipelines' performance. Traditional profilers, like Scalene [1], could not recognize the antipatterns since they are dispersed throughout CPU-GPU interactions.

Problem Defination: we propose **P**erformance **A**ntipatterns **E**liminator **F**ramework (**PAEF**), a framework to identify six performance antipatterns occurring due to data movements between CPU and GPU during training. Our framework co-relates profiles of CPU and GPU executions of the pipeline along with the static analysis of the code to identify the performance antipatterns.

PAEF Architecture

PAEF is a framework that takes ML/DL pipeline and its dataset as input and automatically accelerates the fed ML/DL model training by removing data preparation performance antipatterns. We elucidate the architecture of PAEF in this section using the component structure of PAEF, as shown in the figure2.

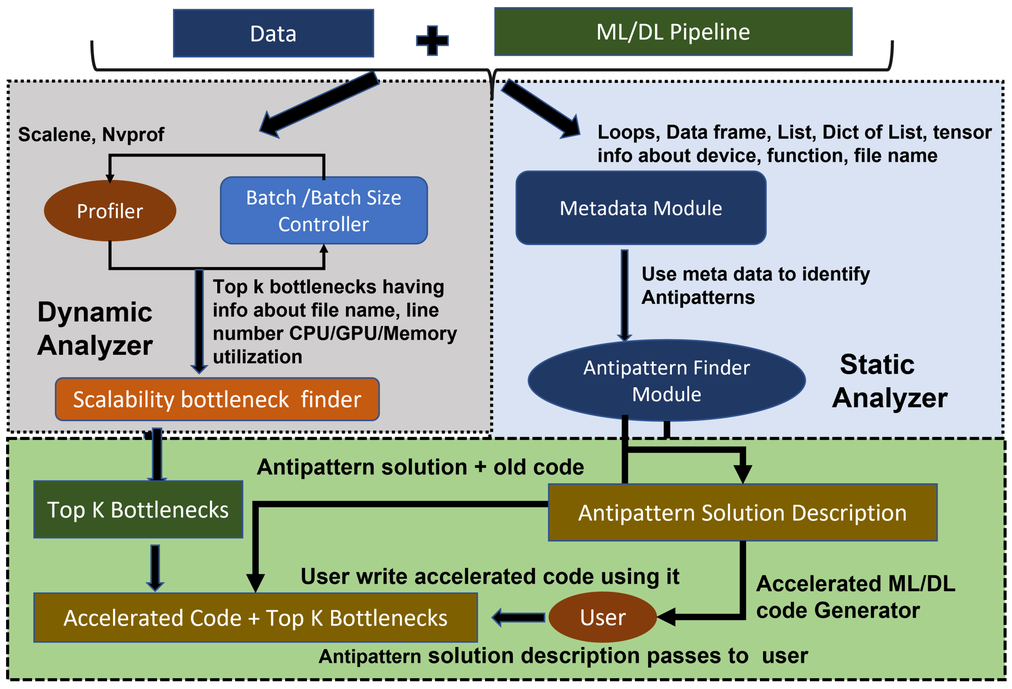


Figure 2. PAEF Architecture

It has three components: Static Analyzer (Metadata Module and Pattern Finder), Dynamic Analyzer, and Accelerated Code Generator.

Static Analyser The static analyzer consists of two component sub-modules based on their functionalities. They are named as MetaData Module and Antipattern Finder Module.

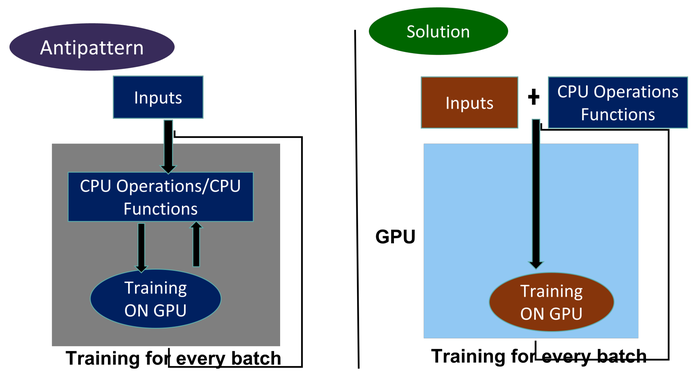
- **MetaData Module:** This module holds the meta-information(filename, device, line number, hardware, etc.) about the objects, functions used in the model training, etc., and employs files to save this metadata. Furthermore, to save the relevant metadata accordingly, it utilizes regular expressions and keyword searches to locate the starting and ending points of model training.
- **Antipattern Finder Module:** The Anti-patter Finder module identifies various performance antipatterns within the provided ML/DL pipeline, as its name implies. It uses the metadata data stored in the Metadata module for this purpose.

Dynamic Analyser: This module employs Scalene and Nvprof profilers to identify bottlenecks associated with "compute time" and "memory transfer time" in the ML/DL training pipeline. This module gives top K tuples(bottlenecks) according to user requirements like CPU/ GPU/ Memory utilization, time, as it takes a lot of time to check top k bottlenecks manually.

Accelerated Code Generator The Accelerated Code Generator module uses the output from the Static Analyzer to find and address performance antipatterns in the ML/DL training pipeline. It creates optimized code through two methods in one, and it generates optimized code. In other, it gives solution descriptions with hints and suggestions, and the user writes the optimized code based on these directions.

Performance Anti-patterns and Accelerations

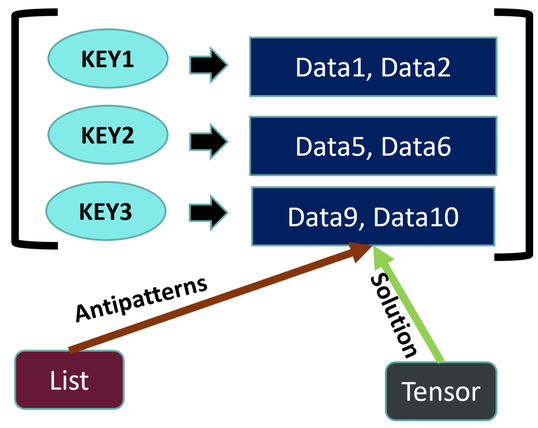
- CPU Operations Function Anti-pattern



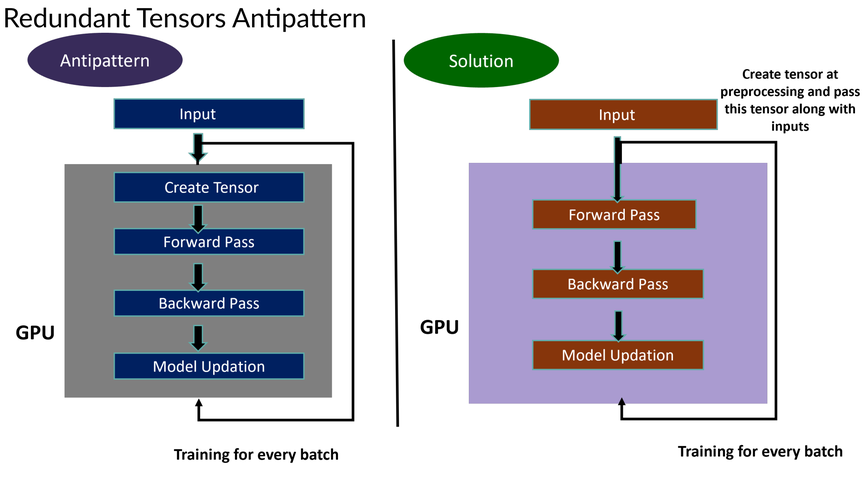
- **Problem:** When training a model on a GPU, certain functions can slow down the process and prevent the GPU from being fully utilized, as functions is running on the CPU only.
- **Solution:** Execute all functions with CPU operations before training so that the GPU can be fully utilized during training.

- Dict as a List Antipattern

- **Problem**:In python, if the dictionary values corresponding to every key are of the list type, we know from experiments that accessing the list can be slow on both CPU and GPU.
- **Solution:** To improve performance, the key is to change the Python list to Tensor type as a value in the dictionary before training(during data prepreparation). Accessing Tensor is extremely fast compared to a Python list.



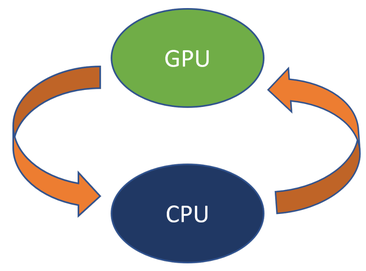
- Redundant Tensors Antipattern



- **Problem:** When a tensor is stored on the host (CPU) and then transferred back and forth between the CPU and GPU, it can significantly decrease training performance.
- **Solution:**To solve this problem, use the $detach()$ function instead of $tolist()$.

- Ping-Pong Antipattern

- **Problem:** When a tensor is stored on the host (CPU) and then transferred back and forth between the CPU and GPU, it can significantly decrease training performance.
- **Solution:**To solve this problem, use the $detach()$ function instead of $tolist()$.



EVALUATION

The Figure3 demonstrates the improvement in speed that was attained by detecting and resolving performance issues, as well as the increase in speed throughout the entire process of training our model using the PAEF framework, on both the CPU and GPU. Furthermore, the chart displays the speedup obtained by eliminating all six. antipatterns in both recommendation pipelines on both CPU and GPU.

Experimental Setup: The underlying hardware of a 16-core 64 GB VM where we performed experiments has a 48-core 128 GB VM, and the GPU was A100 40 GB memory 100 GB RAM.

Overall Model Training Speedup

Models	Hardware	Speedup
Niser	CPU	1.09
Niser	GPU	1.64
Bcd4rec(DDQN)	CPU	6.5
Bcd4rec(DDQN)	GPU	7.6
Bcd4rec(QR-DQN)	CPU	1.7
Bcd4rec(QR-DQN)	GPU	7

Antipatterns Speedup

Antipattern Name	Speedup (CPU)	Speedup(GPU)
Operator Fusion	47X	1041X
Ping Pong	173X	6790X
Dict as a List	61X	74X
List to Tensor	13X	48X
Redundant Tensor	1.012X	1.2X
CPU Operation Function	1.03X	1.4X

Figure 3. Speedup

Conclusions

The training-specific data preparation step in the ML/DL pipeline and identified it as a potential bottleneck for the training process. To address this issue, we proposed the Performance Antipattern Elimination Framework (PAEF), which utilizes static and dynamic analyzers to generate accelerated code for ML/DL training pipelines. This tool provides top K bottlenecks in the pipeline, resulting in a faster training process and improved model performance. We have demonstrated the generalizability of our approach and believe that PAEF can assist data scientists and developers in building efficient and optimized training pipelines.

References

[1] Emery D Berger.
Scalene: Scripting-language aware profiling for python.
arXiv preprint arXiv:2006.03879, 2020.

[2] Mayank Mishra, Archisman Bhowmick, and Rekha Singhal.
Fasca: Framework for automatic scalable acceleration of ml pipeline.
In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1867–1876, 2021.