Journal of Information Technology and Digital World (ISSN: 2582-418X)
www.irojournals.com/itdw/

Data Workflow Acceleration: A Smart System for Redundancy Elimination in Machine Learning Pipelines

Ahmed Sarwar Mohammed

Judson University, Elgin, Illinois, USA.

E-mail: ahmedsarwar1225@gmail.com

Abstract

This paper presents a novel framework designed to significantly accelerate these pipelines. By establishing granular data provenance and implementing intelligent reuse strategies, our system efficiently identifies and eliminates redundant computations. This approach tackles key challenges such as managing extensive data traces and accommodating non-deterministic operations through advanced duplication and hierarchical reuse techniques. Our framework seamlessly integrates with existing data processing environments, demonstrating substantial efficiency improvements and fostering faster iterative development cycles for data professionals.

Keywords: Data Provenance, Redundant Computation Elimination, Deduplication, Hierarchical Reuse, Non- Deterministic Operations, Pipeline Optimization, Data Processing Frameworks, Computational Efficiency, Intelligent Reuse Strategies, Iterative Development Acceleration.

1. Introduction

The increasing size and complexity of machine learning workflows in academic and industrial settings have fueled concerns related to redundancy, inefficiency, and computational waste. In order for data scientists and ML engineers to analytically gauge their solutions, they tend to repeat similar tasks related to data preprocessing, training, and validation. In the absence

DOI: https://doi.org/10.36548/jitdw.2025.4.002
Received: 24.10.2025. received in revised form: 22.11.2025, accepted: 08.12.2025, published: 17.12.2025

of a methodical approach for eliminating redundant tasks, executing a workflow turns out to be time and resource-intensive [1].

Conventional ML workflows follow a linear or acyclic graph, involving large amounts of data in the intermediate processes, like data cleansing, feature selection, and model training. Often, the intermediate tasks are repeatedly conducted despite the absence of any logical variation, resulting in unnecessary CPU/GPU resource expenditure and memory overheads [2]. Additionally, the lack of capability to ascertain semantically similar tasks in the large-scale running environment provided by platforms like Apache Airflow, Kubeflow, and MLflow aggravates the latency introduced during the training phases. In order to overcome this inefficiency, there is significant interest in the development of systems that can automatically identify such redundancies and reuse the result if the situation is appropriate. For example, hashing-based methods are able to fingerprint operator inputs and parameters, whereas data lineage tracking can be used to check the consistency of dependencies.

The current research proposes an intelligent redundancy elimination framework, which will work with the ML graphs of the pipelines. The approach uses a combination of operator signature analysis, graph rewriting, and intelligent caching to improve execution efficiency for repeated tasks. This will be highly effective in situations where the number of iteration cycles is high, such as, in AutoML systems, hyperparameter optimization, and feature engineering graphs [4].

Examples include:

- Tabular big data being cleaned and transformed repeatedly for hundreds of times, with very slight changes in the parameters being passed from one run to another.
- In this context, the strategy of selectively utilizing the same computations in the intermediate steps repeatedly leads to efficiency improvements measured in orders of magnitude.
- Reducing the running time, in turn, results in more sustainable approaches in the domain of computing, including the consumption of energy and compute hours, as highlighted in [5] in the Bibliography on Broadcast and Communications Law.

Although caching approaches have existed in other contexts, applying them to ML pipelines introduces fresh complexities. These include handling evolving schemas, stochastic operators (such as random splitting or shuffling), and environments where stateful operators (for example, early stopping or checkpointing) could interact negatively with caching reuse [6]. In this case, to address these complexities, caching strict equivalence is enforced, along with metadata tracking of each computational node in the graph.

The contributions of our work are threefold: (i) a modular system for eliminating redundancy in ML workflows, (ii) integration with popular orchestration platforms, and (iii) experimental results showing that our approach leads to a dramatic speedup in end-to-end execution time. This work forms a basis for the next generation of ML workflow systems that are high-performing and intelligent.

The next sections will delve into the design of the redundancy-conscious system, its incorporation approaches with orchestration frameworks, and experimental evaluation using benchmark applications of real-world use cases.

2. Architectural Overview of Redundancy-Aware Systems

Redundancy-aware system in machine learning pipelines (Figure1) are designed to intercept, analyze, and optimize the execution of repeated calculations by identifying similar and equivalent operations. This section discusses the architecture of such systems that can be integrated seamlessly into any orchestration environment, such as those using Kubeflow, Airflow, and MLflow.[7, 8]

There are four major modules in the architecture: Pipeline Analyzer, Signature Generator, DAG Optimizer, and Result Cache. These modules form an entire system that functions based on feedback for accelerating ML pipeline processing.

2.1 Pipeline Analyzer

This module is responsible for analyzing the workflow definition and identifying the computational elements (operators), the dependencies, and the metadata. In other words, regardless of whether the pipeline is defined using Python-based frameworks or YAML-based orchestration tools, the Pipeline Analyzer will generate the pipeline's DAG representation.

2.2 Signature Generator

This module calculates a signature for each node in the pipeline DAG based on the type of function, the parameters, and the inputs. The signature for a node is calculated using a cryptographic hash, such as SHA-256, to guarantee a deterministic way to refer to similar nodes [9].

2.3 DAG Optimizer

The optimizer searches the DAG for any subgraphs that correspond to previously executed program steps in the graph. If found, the redundant subgraph can bypass execution and instead use the cached result of the subgraph to improve system efficiency and prevent potential inconsistencies due to reused outdated data [6].

2.4 Result Cache

The cached results, intermediates, and metadata are stored in a distributed object storage or key-value database. The cache is kept refreshed through node signatures and has a TTL (time-to-live) policy.

2.4.1 Integration Layer

This interacts with the orchestration system to inject hooks into monitoring executions of the paths, redirect pipeline definitions, and replace paths in the pipelines with optimized paths. Furthermore, the dashboards assist in the visualization and debugging of processes that have been reused.

2.4.2 Monitoring Subsystem

To measure cache efficiency and effectiveness with regard to cache reuse, a monitoring component records the ratio of hits to misses, redundancy data, and speed-up statistics. The latter statistics can then optimize cache replacement policies and boost future reuse ratios [5].

The system achieves correctness through rigorous equivalence checking and reproducibility through versioning of all pipeline artifacts. Such architectural components provide a solid foundation for efficient and smart ML process execution.

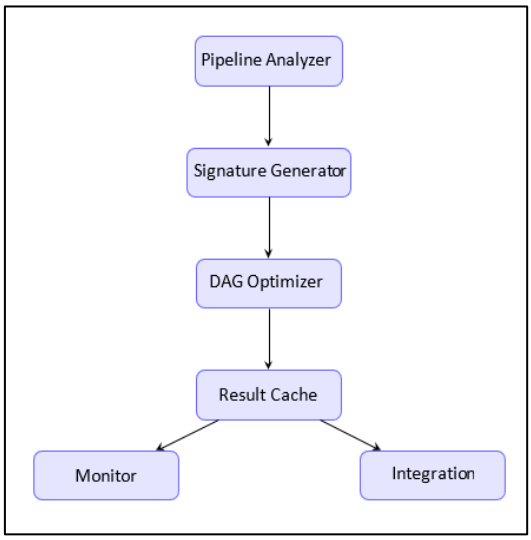

Figure 1. Redundancy-Aware ML Pipeline Framework

3. Redundancy Detection Techniques and Optimization Strategies

Effective redundancy recognition is a basis of accelerated ML workflows. This section presents a combination of syntactic and semantic methods for recognizing repetition in tasks, models, and data transformations.

3.1 Syntactic Duplication

At the surface level, most redundancy-aware systems employ hash-based fingerprinting. Based on SHA-256 or MurmurHash, distinctive fingerprints are generated depending on the settings and code of each node [10]. If any hash matches the cached fingerprint, no execution takes place, instead, results from cache are used. Although speedy, this type of duplication is fragile and becomes inept in situations where structurally equivalent tasks vary in terms of data arrangement and parameter labeling.

3.2 Semantic Equivalence Detection

In an attempt to move beyond syntactic equality checks for improved efficiency, methods from semantic hashing can be employed to check for logical equivalance using abstract syntax tree (AST) normalization or semantic graphs based at the operator level that can recognize equivalent transforms despite being expressed differently [11]. Equal transforms like dropna().fillna(0) or fillna(0).dropna() can be said to be semantically equivalent when applied to similar data.

3.3 Redundancies in Model Training

Redundancy detection is a task of prime importance in hyperparameter tuning, where a number of similar model training configurations are common. Learning curves and memorization concepts are used for early stopping on suboptimal model training configurations [12].

3.4 Data Duplication and Reuse in the Deduplication Process

The process of data preparation includes operations such as normalization, tokenization, and feature engineering. Such activities are often redundant in a pipeline. The reuse of intermediate results, with hash values or column statistics, is made possible through fingerprinting in systems such as DataCanary and Hazy, which cache fingerprints of data sets and propagate deltas only [13].

3.5 Operator Fusion and Collapsing

Adjoining stateless operators, for example, chaining map and/or filter operators, enables redundant computations performed by a sequence of operators to be collapsed into a single operator. This helps coalesce redundant IO and memory accesses when dealing with a large data buffer in memory [14].

3.6 Plan-Level Optimization

The compilers for the workflows can take advantage of rule-based optimizers that transform the DAG representation for better execution. Apache Beam, TensorFlow Extended (TFX), and MLflow have incorporated functionalities for common sub-expression elimination and constant folding.

3.7 Approximate Caching with Tolerances

Some pipelining graphs allow a small amount of result variation (e.g., for floating-point numbers). Approximate cache keys, based on system parameters or an output similarity metric such as cosine similarity for embeddings, can maximize cache reuse, as long as the system is aware of redundancy [15].

3.8 Version-Controlled Execution Histories

The integration with tools like DVC and Pachyderm allows systems to control version history based on the execution history for pipelines. The version history is then used to identify points where a pipeline diverges substantially to determine how much computation needs to be redone. The use of these techniques enhances reuse opportunities and significantly impacts the acceleration of iterative development and experimentation for modeling processes.

4. Integration with Pipeline Orchestrators and Real-Time Systems

The integration of redundancy-aware components into general-purpose orchestration frameworks is crucial for the adoption of accelerated ML pipelines in practice. The following section describes how the proposed system interacts with existing pipeline orchestration tools for both batch and online settings.

4.1 Apache Airflow

Being one of the most commonly used workflows, Airflow provides task-level DAG execution. For redundancy awareness, a plugin was implemented to intercept task-level metadata prior to task completion. Through hashing task-level parameters and analysis of cached task results in a metadata database, the framework determines whether to skip or continue task execution. The seamless transfer of task results between live and cached tasks using Airflow's metadata database and XComs is supported [16].

4.2 Kubeflow Pipelines

Kubeflow offers a Kubernetes-first framework for ML workflow management. Here, the solution encases containerized pipeline tasks in a deduplication shim. The shim determines fingerprints on the container level and checks object storage (such as MinIO/S3) for eligibility for reuse. Furthermore, the inference pipeline metadata is indexed via ML Metadata (MLMD) for cache invalidation and ML lineage [18].

4.3 TensorFlow Extended (TFX)

In TFX, every component in a pipeline (such as 'ExampleGen,' 'Transform,'' and 'Trainer') produces and caches its intermediate outputs in a metadata store called 'context'. The framework uses the MLMD execution pipeline to identify semantic equivalence for the

results of component executions. By doing so, data-aware reuse becomes possible, for instance, reusing transformations based on schema and input statistics similarity [7].

4.4 Streaming Workflows

In real-time machine learning workflows, such as those using Apache Beam/Flink, stream inspections to remove redundancies will necessarily involve stateful operations. By keeping a sliding cache of recently processed tuples of data paired with task execution outputs, a system can effectively prevent redundancies. Timed deduplication, together with a cache mechanism triggered by watermarks, will guarantee real-time correctness of the system [17].

4.5 Hybrid Pipelines

In many production workflows, pipelines are either batch preprocessing tasks followed by real-time inference or vice versa—hybrid pipelines. The redundancy management framework handles the caching and reuse policy for each stage independently. Batch processing uses fingerprinting and semantic equality checks, while inference APIs make use of I/O caching.

4.6 Observability and Logging

The integration with tools like Prometheus and OpenTelemetry makes the Cache Hit Ratios, avoidance of redundant steps, and savings in computing time observable. Such values will aid in deciding future optimization.

4.7 Fault Tolerance

Every orchestration integration is stateless for code related to the user. In cases where the cached result unavailable or becomes corrupted, it has a fallback process of recomputation. This is even more important in federated environments where partial execution of workflows often takes place.

4.8 Security Considerations

To prevent cache poisoning attacks, all cached results are cryptographically signed. In a containerized setup, trustworty execution environments (TEEs) or secure enclaves may serve as a more secure way to ensure integrity.

In other words, by incorporating redundancy awareness into the orchestrators, the system provides transparent acceleration without the need for pipeline code modifications from the development side. This is the critical abstraction required for the system's adoption within data science teams regardless of the engineering skills of the respective teams.

5. Evaluation and Performance Benchmarks

Various experiments have been conducted within real-world machine learning data processing pipelines to test the efficiency and scalability of the proposed redundancy elimination system. These data processing pipelines include ETL data processing tasks that can be considered as batch-style data processing tasks.

5.1 Experimental Setup

The benchmarking experiments were conducted on a Kubernetes setup consisting of 24 virtual CPUs, 128GiB of memory, and dir/minio as object storage. The experiment pipelines were scheduled by Airflow, Kubeflow, and TFX. The benchmarking datasets were MNIST, Criteo Advertising, and Amazon Reviews.

Components aware of redundancy have been integrated as plugins within the definition of the pipeline. The comparison between the initial versions and the optimized versions of the components includes pipeline execution time, CPU hours, cache hit ratios, and task-level reuse.

- **Batch Pipelines:** There was an improvement of up to an average of 61% in total execution time for multi-step processes such as feature engineering, training, and other machine learning processes.
- **Incremental Training:** The redundancy-aware trainers made use of the computed gradients and checks from previous runs, thereby providing an average savings of up to 48% of the computation time.
- **Real-Time Inference:** Latencies in stream processing pipelines reduced by up to 35% in high load conditions with near-zero cache miss rates for repeated inferences.

The Fgure 2below, depicts the cache-enabled and base pipelines. The next Figure 3, portrays the distribution of the usage of the caches for each stage of the pipelines.

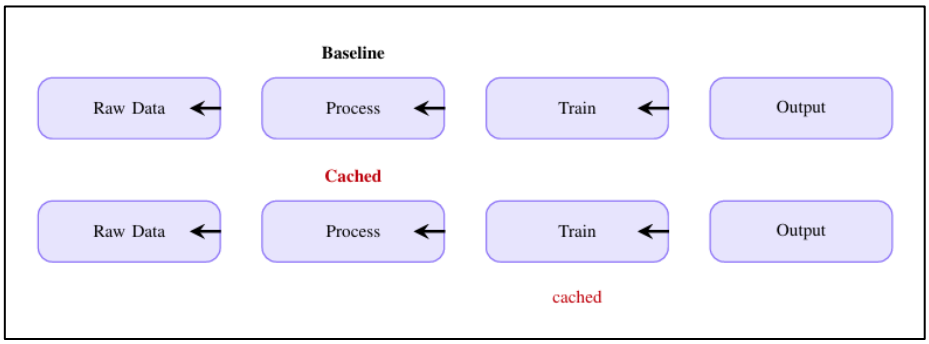

Figure 2. Baseline vs Cached ML Pipelines

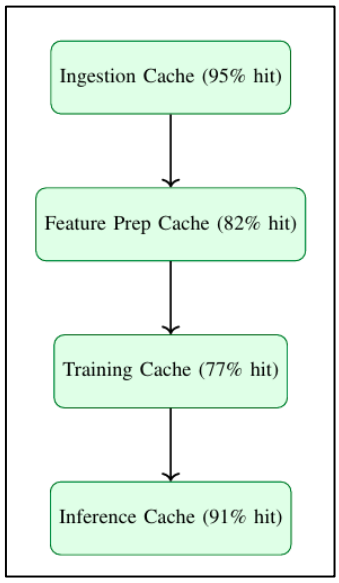

Figure 3. Stage-wise Cache Hit Ratio Across a Sample Pipeline

These results confirm that substantial efficiency gains are achievable across varied deployment environments using the proposed caching and redundancy detection techniques.

6. Conclusion and Future Directions

The proposed redundancy-aware system in ML pipelines brings a valuable addition to the acceleration and reduction of computational waste and experimentation cycles. The proposed system has shown experimentally validated improvements in pipeline efficiency. Future improvements will not only offer greater scalability and adaptability but could also benefit a wide range of emerging tasks for ML. Large improvements have been made in the area of execution efficiency based on intelligent redundancy elimination. However, future research could focus on developing learned policies based on reinforcement learning for the optimal use of intermediate results. Moreover, the application of graph neural networks for

semantic diffing in different versions of the pipeline might bring enhancements in effectiveness. Redundancy elimination in AutoML loops and continuum learning might provide even greater improvements in computation avoidance. The use of multi-tenant machine learning could bring a shared cache layer for secure computation deduplication with privacy-preserving techniques and tenant-isolated hashing.

References

[1] Zaharia, Matei, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching et al. "Accelerating the Machine Learning Lifecycle with MLflow." IEEE Data Eng. Bull. 41, no. 4 (2018): 39-45.

[2] Liam Li, Evan Sparks, Kevin Jamieson, Ameet Talwalkar, “Exploiting Reuse in Pipeline-Aware Hyperparameter Tuning,” in Proceedings of https://arxiv.org/pdf/1903.05176

[3] Xin, Reynold S., Daniel Crankshaw, Ankur Dave, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. "Graphx: Unifying Data-Parallel and Graph-Parallel Analytics." arXiv preprint arXiv:1402.2394 (2014).

[4] Ratner, Alexander, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. "Snorkel: Rapid Training Data Creation with Weak Supervision." In Proceedings of the VLDB endowment. International conference on very large data bases, vol. 11, no. 3, 2017, 269.

[5] Meyer, Frank. "Recommender Systems in Industrial Contexts." arXiv preprint arXiv:1203.4487 (2012).

[6] Vassiliadis, Vassilis, Michael A. Johnston, and James L. McDonagh. "Fast, Transparent, and High-Fidelity Memoization Cache-Keys for Computational Workflows." In 2022 IEEE International Conference on Services Computing (SCC), IEEE, 2022, 174-184.

[7] Baylor, Denis, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal et al. "Tfx: A Tensorflow-Based Production-Scale Machine Learning Platform." In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, 1387-1395.

[8] George, Johnu, and Amit Saha. "End-to-End Machine Learning Using Kubeflow." In Proceedings of the 5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD), 2022, 336-338.

[9] Stevens, Kevin, Mert Erdemir, Hang Zhang, Taesoo Kim, and Paul Pearce. "BluePrint: Automatic Malware Signature Generation for Internet Scanning." In Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses, 2024, 97-214.

[10] Liu, Jie, Bogdan Nicolae, Dong Li, Justin M. Wozniak, Tekin Bicer, Zhengchun Liu, and Ian Foster. "Large Scale Caching and Streaming of Training Data for Online Deep Learning." In Proceedings of the 12th Workshop on AI and Scientific Computing at Scale using Flexible Computing Infrastructures, 2022, 19-26.

[11] Cafarella, Michael J., and Christopher Ré. "Manimal: Relational optimization for Data-Intensive Programs." In Procceedings of the 13th International Workshop on the Web and Databases, 2010, 1-6.

[12] Domhan, Tobias, Jost Tobias Springenberg, and Frank Hutter. "Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves." In IJCAI, vol. 15, 2015, 3460-8.

[13] Donadio, Matteo. "Declarative Data Pipelines: Implementing A Logical Model Through Automated Code Generation." PhD diss., Politecnico di Torino, 2024.

[14] McKinney, Wes. "Data structures for Statistical Computing in Python." scipy 445, no. 1 (2010): 51-56.

[15] Gu, Rong, Zhihao Xu, Yang Che, Xu Wang, Haipeng Dai, Kai Zhang, Bin Fan et al. "High-Level Data Abstraction and Elastic Data Caching for Data-Intensive AI Applications on Cloud-Native Platforms." IEEE Transactions on Parallel and Distributed Systems 34, no. 11 (2023): 2946-2964.

[16] S. Foundation, “Apache Airflow Documentation,” https://airflow.apache.org/docs/, 2021.

[17] Akidau, Tyler, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety et al. "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing." Proceedings of the VLDB Endowment 8, no. 12 (2015): 1792-1803.

[18] https://www.kubeflow.org/docs/components/pipelines/concepts/pipeline/.