Technical Perspective on "MEMPHIS: Holistic Lineage-based Reuse and Memory Management for Multi-backend ML Systems"

Arun Kumar
University of California, San Diego and RapidFire AI, Inc.
akk018@ucsd.edu

ML deployments in real-world settings are often heterogeneous, spanning local multi-core CPU operations, GPU-accelerated computations, and distributed execution on platforms such as Apache Spark. This heterogeneity arises from practical necessity: a feature engineering step might run locally, a large matrix multiplication offloads to Spark when data exceeds driver memory, and DNN layers execute on GPUs. Or an initial homogeneous setup might evolve over time into a heterogeneous one. This multi-backend reality creates a systems challenge that has received surprisingly little research attention: *How to holistically manage computation reuse and memory across backends with fundamentally different execution models, memory hierarchies, and caching primitives?*

The Problem and Prior Work. Exploratory data science workloads often involve redundant computations; hyperparameter tuning, cross-validation, and AutoML repeatedly execute similar computations with slight variations. Prior work at the data+ML systems intersection proposed reuse techniques but largely in isolation: coarse-grained pipeline-level reuse via compile-time materialization (e.g., HELIX), input data pipeline caching for DNNs (e.g., tf.Data, Cachew), prediction caching for inference (Clipper), and DNN feature layer reuse (e.g., Vista, Nautilus). These target specific workload-backend combinations and do not compose well. The authors' prior LIMA framework introduced fine-grained lineage-based reuse but was limited to local CPU operations. In production deployments supporting multiple teams, be it at Big Tech, enterprises, or academic research centers, I have seen how ML/AI users often struggle with such fragmentation, manually orchestrating caching across backends with ad-hoc scripts.

The MEMPHIS Approach. This paper by Phani and Boehm, which received the EDBT 2025 Best Paper Award, introduces a new systems approach they call MEMPHIS. The key insight is that fine-grained lineage tracing, i.e., tracking provenance of every intermediate at the operator level, can serve as a unified abstraction that transcends backend boundaries. MEMPHIS maintains a hierarchical lineage cache mapping traces to backend-specific objects: in-memory matrix blocks on the driver, distributed RDDs in Spark, and GPU pointers on accelerators.

The technical contributions address each backend's distinct challenges. For Spark, lazy evaluation means eager caching degrades performance; the paper shows eager materialization of 12K RDDs is 10× slower than no caching. MEMPHIS introduces delayed caching that defers persistence until repeated hits, lazy garbage collection for dangling broadcast references, and driver-side action reuse to bypass redundant job execution. For GPUs, allocation and copy overhead dominate (4.6× and 9× longer than computation respectively). MEMPHIS combines reuse with memory recycling using Live/Free pointer lists, prioritizing exact-size recycling to avoid fragmentation while using a scoring function based on recency, lineage height, and compute cost. The compiler is extended with asynchronous prefetch operators, workload-aware eviction injection, and an operator linearization algorithm to maximize concurrent cross-backend execution.

Practical Significance. What makes this work compelling is its comprehensive systems approach and empirical rigor. The authors tackle full integration challenges: how lineage tracing interacts with Spark's BlockManager, how GPU eviction must account for CUDA synchronization barriers, and how compiler rewrites inject checkpoints based on estimated reuse potential. This requires not just deep understanding of each system's tradeoffs but also synthesizing across them to make the approach work.

Implemented in Apache SystemDS and evaluated on 7 diverse ML tasks–grid search (9.6× speedup), Poisson matrix factorization (7.9×), Hyperband, data cleaning pipelines, dropout tuning, machine translation, and transfer learning–comparisons against PyTorch, LIMA, HELIX, and others show consistent improvements. The open-source implementation and reproducibility package strengthen practical value.

Limitations and Future Directions. MEMPHIS is deeply integrated with SystemDS's compilation model. Extending to other setups, e.g., Ray-based, would require heavy-duty engineering. The paper also focused on multi-core CPU and Spark backends for clusters, which is a great start, but multi-GPU clusters are also common, as are newer accelerators such as TPUs–all this merits further study. Cross-backend object migration was implemented but data movement overhead outweighed benefits. At the algorithmics level, operator scheduling remains heuristic-based; cost-based reuse-aware placement is future work. Overall, there are many opportunities for future research across the systems stack building on this foundation.

Broader Context. MEMPHIS demonstrates that holistic reasoning about reuse across the full software stack yields substantial benefits over optimizing components in isolation. This work reflects a major trend: as ML/AI infrastructure grows in complexity and scale, core database systems principles–query optimization, tiered caching, memory management, lineage tracking, etc.–become critical for efficient operation at scale. For instance, it is not a coincidence that OpenAI's first acquisition was a DBMS startup. I hope the DB community keeps pursuing such cross-pollination to power the ongoing AI boom while mitigating resource/energy wastage.