Helix: Holistic Optimization for Accelerating Iterative Machine Learning

Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, Aditya Parameswaran
University of Illinois (UIUC)

Motivation

❑**Machine learning app. development is iterative**
- Developers *iteratively* modify the workflow to improve performance through trial-and-error, changing data preproc., the mode, and post proc.

❑**Redundant computation across iterations**
- Existing tools rerun the workflow end-to-end in every iteration, *regardless of the change*
- Drain on *resources* and *developer productivity*

What is Helix?

Fast and user-friendly tool for accelerating ML application development from scratch!

❑**Accelerates iterative workflow development**
- By *removing redundant computation* across iterations. Up to ***19x*** speed-up over 10 iterations.

❑**End-to-end machine learning system**
- Handles *specification* and *execution* of ML workflows encompassing all components.

❑**Simple, intuitive, succinct DSL in Scala**
- Imperative code directly embedded for UDFs
- Small number of composable and customizable operators to handle most ML workflow operators
- Separate data structs. optimized for data prep. and ML

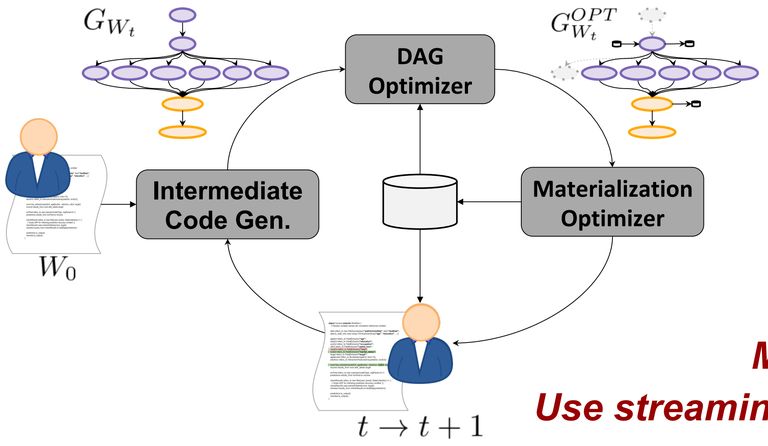
Iterative Execution Optimization

Speeding up Iterative Execution via Intermediates Reuse

❑**DAG Optimizer**

Enabled by declarativity + operator DAG model
- Detect changes
- Prune redundant operators

MinCut on operator DAG ⇦
- Compute optimal reuse policies



❑**Mat. Optimizer**

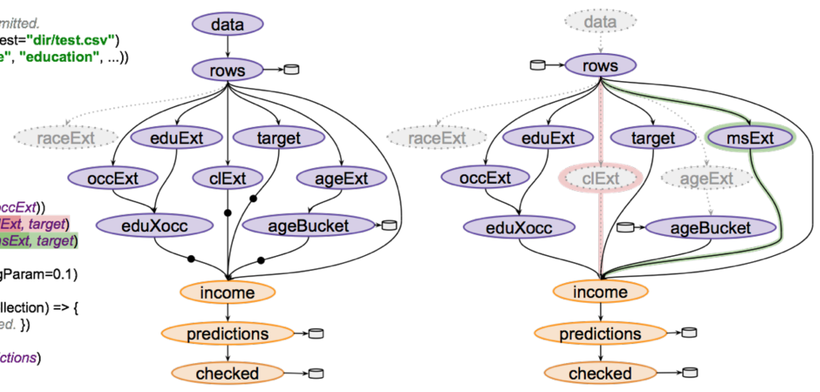
Selectively materialize operator outputs to be reused in future iterations for speedup

Materialize-all can be prohibitive
Use streaming heuristic effective in practice

Declarative Programming Interface

```
1.  object Census extends Workflow {
2.    // Declare variable names (for consistent reference) omitted.
3.    data refers_to new FileSource(train="dir/train.csv", test="dir/test.csv")
4.    data is_read_into rows using CSVScanner(Array("age", "education", ...))
5.    ageExt refers_to FieldExtractor("age")
6.    eduExt refers_to FieldExtractor("education")
7.    occExt refers_to FieldExtractor("occupation")
8.    clExt refers_to FieldExtractor("capital_loss")
9.    raceExt refers_to FieldExtractor("race")
    + msExt refers_to FieldExtractor("marital_status")
10.   target refers_to FieldExtractor("target")
11.   ageBucket refers_to Bucketizer(ageExt, bins=10)
12.   eduXocc refers_to InteractionFeature(Array(eduExt, occExt))
13. - rows has_extractors(eduExt, ageBucket, eduXocc, clExt, target)
    + rows has_extractors(eduExt, ageBucket, eduXocc, msExt, target)
14.   income results_from rows with_labels target
15.   incPred refers_to new Learner(modelType="LR"", regParam=0.1)
16.   predictions results_from incPred on income
17.   checkResults refers_to new Reducer( (preds: DataCollection) => {
18.     // Scala UDF for checking prediction accuracy omitted. })
19.   checkResults uses extractorName(rows, target)
20.   checked results_from checkResults on testData(predictions)
21.   checked is_output()
22. }
```

Data Preprocessing / Machine Learning

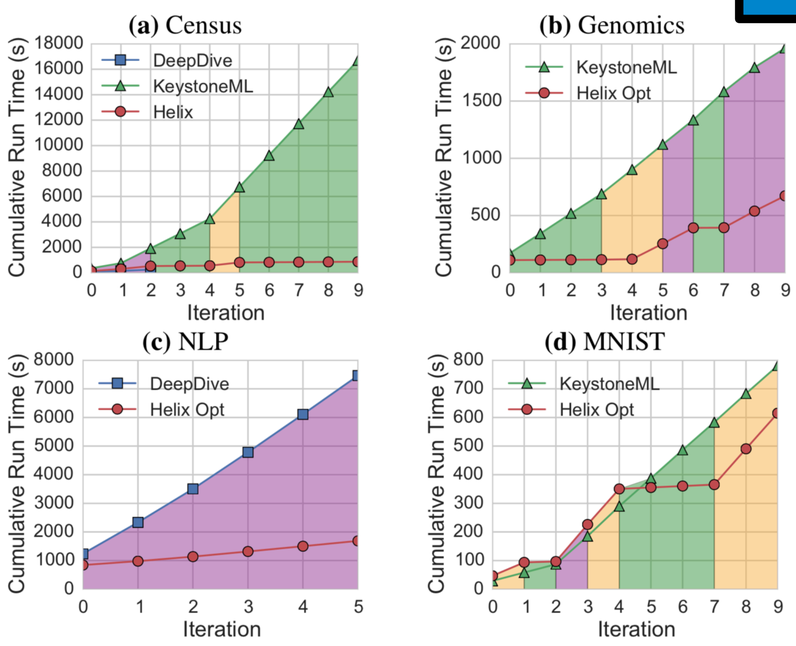


a) Census Workflow Program b) Optimized DAG for original workflow c) Optimized DAG for modified workflow

Example workflow for predicting income from census data.

Concise code with high level operators amenable to quick iteration
Allows Helix to manage reuse at the desired level of granularity

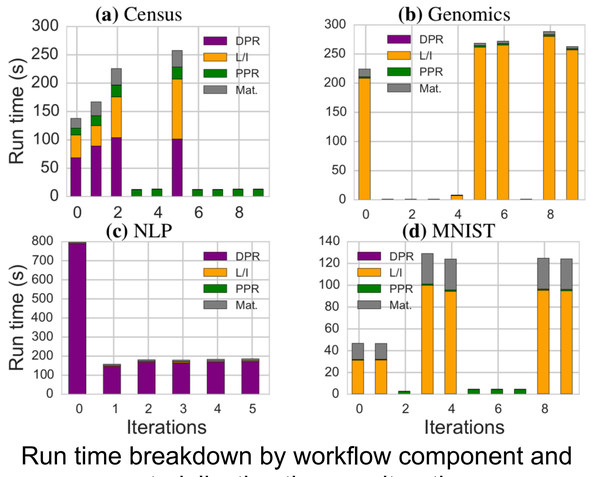
Empirical Evaluation



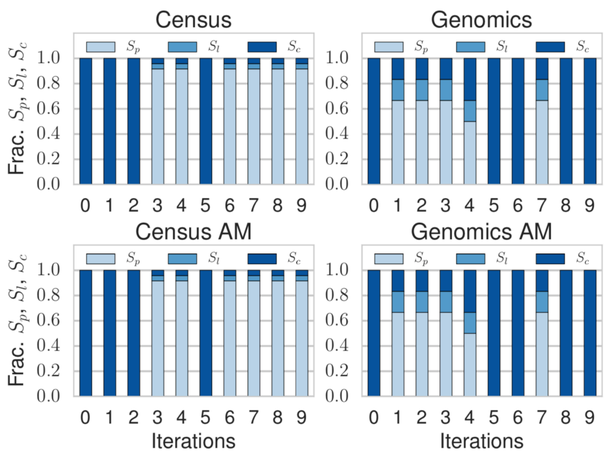
Cumulative run time for four workflows: Census (classification on structured data), Genomics (Clustering on word embeddings), NLP, MNIST (Computer Vision).
In each iteration, an operator of type
- data preprocessing (DPR) → low reuse
- machine learning (L\I) → medium reuse
- post processing (PPR) → high reuse

is modified and the workflow is rerun.
KeystoneML: No materialization or reuse of intermediate results
DeepDive: Materialize all intermediates & manual reuse

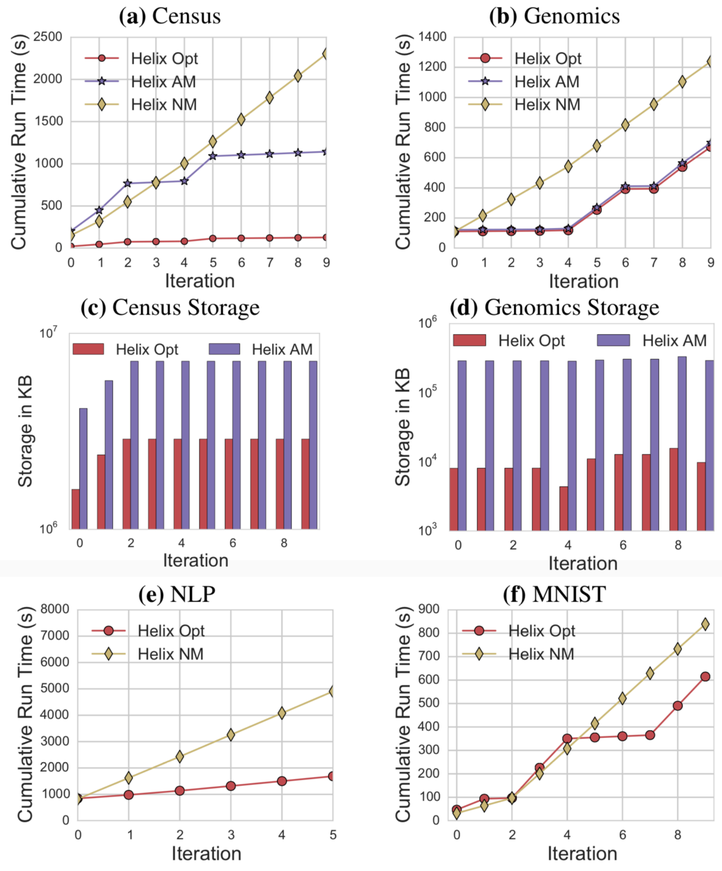
Speed up, up to 19x, from mat. and reuse in all four applications, including mat. overhead.



Run time breakdown by workflow component and materialization time per iteration.



Fraction of states in S_p (pruned), S_l (load), S_c (compute) as determined by Helix Opt. and Helix AM (mat. everything).



Cumulative run time and storage use against materialization heuristics AM (mat. everything) and NM (mat. nothing).

Helix Opt. achieves the same reuse as materialize-all (max reuse) with < 50% storage

More on Helix at the project website @***https://helix-ml.github.io*** & in the technical report @***https://arxiv.org/pdf/1812.05762.pdf***