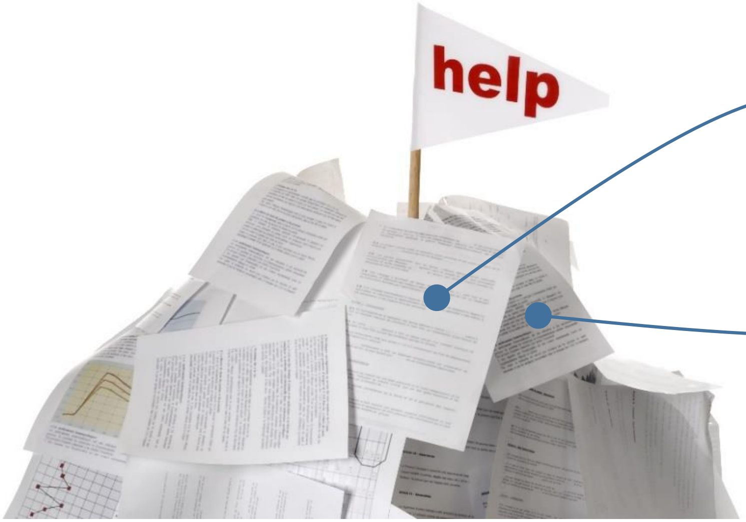
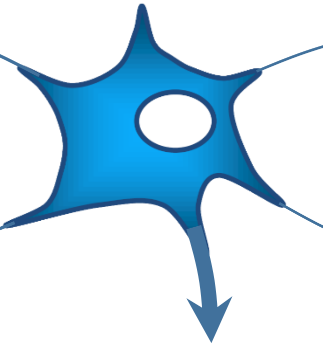
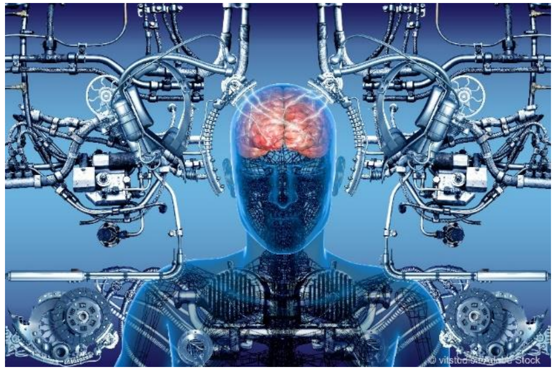
Developing a common language to facilitate reproducible research and technology transfer: challenges and solutions

The 1st ACM Conference on Reproducibility and Replicability



Collective Mind





Grigori Fursin

MLCommons taskforce on automation and reproducibility

cTuning.org **github.com/mlcommons/ck** cKnowledge.org

Why do I care about reproducibility, replicability and technology transfer?

Prequel to this keynote (learning and prototyping phase)

"Reproducing 150 Research Papers and Testing Them in the Real World: Challenges and Solutions"
ACM Tech Talk 2021: learning.acm.org/techtalks/reproducibility

"Collective Knowledge: organizing research projects as a database of reusable components and portable workflows with common APIs"

Philosophical Transactions of the Royal Society 2021: arxiv.org/abs/2011.01149

This keynote presents a new automation and reproducibility language (Collective Mind) being developed by the MLCommons task force on automation and reproducibility since 2022: cKnowledge.org/mlcommons-taskforce

Why do I care about reproducibility, replicability and technology transfer?

My 1st summer R&D project: implement Hopfield Neural Network as a semiconductor device and test in the real world. Started in 1996.

Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. Hopfield

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

ABSTRACT **Computational properties of use to biological organisms or to the construction of computers can emerge as collective properties of systems having a large number of simple equivalent components (or neurons). The physical meaning of content-addressable memory is described by an appropriate phase space flow of the state of a system. A model of such a system is given, based on aspects of neurobiology but readily adapted to integrated circuits. The collective properties of this model produce a content-addressable memory which correctly yields an entire memory from any subpart of sufficient size. The algorithm for the time evolution of the state of the system is based on asynchronous parallel processing. Additional emergent collective properties include some capacity for generalization, familiarity recognition, categorization, error correction, and time sequence retention. The collective properties are only weakly sensitive to details of the modeling or the failure of individual devices.**

Given the dynamical electrochemical properties of neurons and their interconnections (synapses), we readily understand schemes that use a few neurons to obtain elementary useful biological behavior (1–3). Our understanding of such simple circuits in electronics allows us to plan larger and more complex circuits which are essential to large computers. Because evolution has no such plan, it becomes relevant to ask whether the ability of large collections of neurons to perform "computational" tasks may in part be a spontaneous collective consequence of having a large number of interacting simple neurons.

In physical systems made from a large number of simple elements, interactions among large numbers of elementary components yield collective phenomena such as the stable magnetic orientations and domains in a magnetic system or the vortex patterns in fluid flow. Do analogous collective phenomena in a system of simple interacting neurons have useful "computational" correlates? For example, are the stability of memories, the construction of categories of generalization, or time-sequential memory also emergent properties and collective in origin? This paper examines a new modeling of this old and fundamental question (4–8) and shows that important computational properties spontaneously arise.

All modeling is based on details, and the details of neuroanatomy and neural function are both myriad and incompletely known (9). In many physical systems, the nature of the emergent collective properties is insensitive to the details inserted in the model (e.g., collisions are essential to generate sound waves, but any reasonable interatomic force law will yield appropriate collisions). In the same spirit, I will seek collective properties that are robust against change in the model details.

The model could be readily implemented by integrated circuit hardware. The conclusions suggest the design of a delocalized content-addressable memory or categorizer using extensive asynchronous parallel processing.

The general content-addressable memory of a physical system

Suppose that an item stored in memory is "H. A. Kramers & G. H. Wannier *Phys. Rev.* **60**, 252 (1941)." A general content-addressable memory would be capable of retrieving this entire memory item on the basis of sufficient partial information. The input "& Wannier, (1941)" might suffice. An ideal memory could deal with errors and retrieve this reference even from the input "Vannier, (1941)". In computers, only relatively simple forms of content-addressable memory have been made in hardware (10, 11). Sophisticated ideas like error correction in accessing information are usually introduced as software (10).

There are classes of physical systems whose spontaneous behavior can be used as a form of general (and error-correcting) content-addressable memory. Consider the time evolution of a physical system that can be described by a set of general coordinates. A point in state space then represents the instantaneous condition of the system. This state space may be either continuous or discrete (as in the case of N Ising spins).

The equations of motion of the system describe a flow in state space. Various classes of flow patterns are possible, but the systems of use for memory particularly include those that flow toward locally stable points from anywhere within regions around those points. A particle with frictional damping moving in a potential well with two minima exemplifies such a dynamics.

If the flow is not completely deterministic, the description is more complicated. In the two-well problems above, if the frictional force is characterized by a temperature, it must also produce a random driving force. The limit points become small limiting regions, and the stability becomes not absolute. But as long as the stochastic effects are small, the essence of local stable points remains.

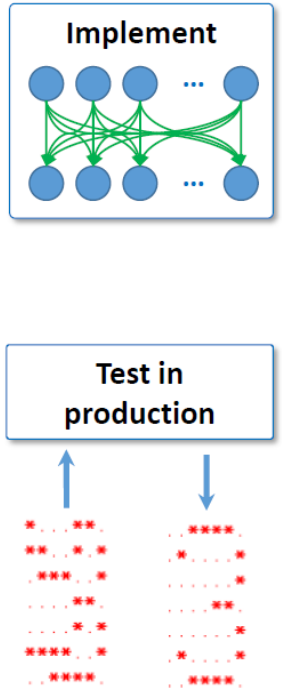
Consider a physical system described by many coordinates $X_1 \cdots X_N$, the components of a state vector X. Let the system have locally stable limit points $X_a, X_b, \cdots$. Then, if the system is started sufficiently near any X_a, as at $X = X_a + \Delta$, it will proceed in time until $X \approx X_a$. We can regard the information stored in the system as the vectors $X_a, X_b, \cdots$. The starting point $X = X_a + \Delta$ represents a partial knowledge of the item X_a, and the system then generates the total information X_a.

Any physical system whose dynamics in phase space is dominated by a substantial number of locally stable states to which it is attracted can therefore be regarded as a general content-addressable memory. The physical system will be a potentially useful memory if, in addition, any prescribed set of states can readily be made the stable states of the system.

The model system

The processing devices will be called neurons. Each neuron i has two states like those of McCullough and Pitts (12): $V_i = 0$

The publication costs of this article were defrayed in part by page charge payment. This article must therefore be hereby marked "*advertisement*" in accordance with 18 U. S. C. §1734 solely to indicate this fact.



My original background was in physics and electronics

Why do I care about reproducibility, replicability and technology transfer?

My 1st summer R&D project: implement Hopfield Neural Network as a semiconductor device and test in the real world.
Started in 1996 and finished in 1999 while exposing lots of issues ...

Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. Hopfield

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

ABSTRACT Computational properties of use to biological organisms or to the construction of computers can emerge as collective properties of systems having a large number of simple equivalent components (or neurons). The physical meaning of content-addressable memory is described by an appropriate phase space flow of the state of a system. A model of such a system is given, based on aspects of neurobiology but readily adapted to integrated circuits. The collective properties of this model produce a content-addressable memory which correctly yields an entire memory from any subpart of sufficient size. The algorithm for the time evolution of the state of the system is based on asynchronous parallel processing. Additional emergent collective properties include some capacity for generalization, familiarity recognition, categorization, error correction, and time sequence retention. The collective properties are only weakly sensitive to details of the modeling or the failure of individual devices.

Given the dynamical electrochemical properties of neurons and their interconnections (synapses), we readily understand schemes that use a few neurons to obtain elementary useful biological behavior (1–3). Our understanding of such simple circuits in electronics allows us to plan larger and more complex circuits which are essential to large computers. Because evolution has no such plan, it becomes relevant to ask whether the ability of large collections of neurons to perform "computational" tasks may in part be a spontaneous collective consequence of having a large number of interacting simple neurons.

In physical systems made from a large number of simple elements, interactions among large numbers of elementary components yield collective phenomena such as the stable magnetic orientations and domains in a magnetic system or the vortex patterns in fluid flow. Do analogous collective phenomena in a system of simple interacting neurons have useful "computational" correlates? For example, are the stability of memories, the construction of categories of generalization, or time-sequential memory also emergent properties and collective in origin? This paper examines a new modeling of this old and fundamental question (4–8) and shows that important computational properties spontaneously arise.

All modeling is based on details, and the details of neuroanatomy and neural function are both myriad and incompletely known (9). In many physical systems, the nature of the emergent collective properties is insensitive to the details inserted in the model (e.g., collisions are essential to generate sound waves, but any reasonable interatomic force law will yield appropriate collisions). In the same spirit, I will seek collective properties that are robust against change in the model details.

The model could be readily implemented by integrated circuit hardware. The conclusions suggest the design of a delocalized content-addressable memory or categorizer using extensive asynchronous parallel processing.

The general content-addressable memory of a physical system

Suppose that an item stored in memory is "H. A. Kramers & G. H. Wannier *Phys. Rev.* **60**, 252 (1941)." A general content-addressable memory would be capable of retrieving this entire memory item on the basis of sufficient partial information. The input "& Wannier, (1941)" might suffice. An ideal memory could deal with errors and retrieve this reference even from the input "Vannier, (1941)". In computers, only relatively simple forms of content-addressable memory have been made in hardware (10, 11). Sophisticated ideas like error correction in accessing information are usually introduced as software (10).

There are classes of physical systems whose spontaneous behavior can be used as a form of general (and error-correcting) content-addressable memory. Consider the time evolution of a physical system that can be described by a set of general coordinates. A point in state space then represents the instantaneous condition of the system. This state space may be either continuous or discrete (as in the case of N Ising spins).

The equations of motion of the system describe a flow in state space. Various classes of flow patterns are possible, but the systems of use for memory particularly include those that flow toward locally stable points from anywhere within regions around those points. A particle with frictional damping moving in a potential well with two minima exemplifies such a dynamics.

If the flow is not completely deterministic, the description is more complicated. In the two-well problems above, if the frictional force is characterized by a temperature, it must also produce a random driving force. The limit points become small limiting regions, and the stability becomes not absolute. But as long as the stochastic effects are small, the essence of local stable points remains.

Consider a physical system described by many coordinates $X_1 \cdots X_N$, the components of a state vector X. Let the system have locally stable limit points $X_a, X_b, \cdots$. Then, if the system is started sufficiently near any X_a, as at $X = X_a + \Delta$, it will proceed in time until $X \approx X_a$. We can regard the information stored in the system as the vectors $X_a, X_b, \cdots$. The starting point $X = X_a + \Delta$ represents a partial knowledge of the item X_a, and the system then generates the total information X_a.

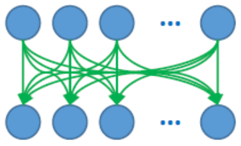
Any physical system whose dynamics in phase space is dominated by a substantial number of locally stable states to which it is attracted can therefore be regarded as a general content-addressable memory. The physical system will be a potentially useful memory if, in addition, any prescribed set of states can readily be made the stable states of the system.

The model system

The processing devices will be called neurons. Each neuron i has two states like those of McCullough and Pitts (12): $V_i = 0$

The publication costs of this article were defrayed in part by page charge payment. This article must therefore be hereby marked "*advertisement*" in accordance with 18 U. S. C. §1734 solely to indicate this fact.

Implement



Prepare datasets

Implement model in C from scratch

Implement training in C and bash

Implement validation in C and bash

Training too slow: parallelize

Too many crashes: add fault-tolerance

Keep track of experiments

Get first results: not matching results from the paper

Debug

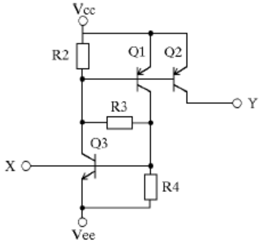
Model is finally working but too slow and inaccurate for production on real data

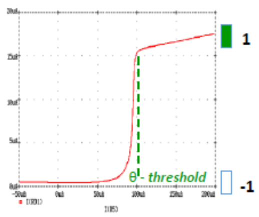
Collect more realistic datasets

Optimize matrix multiply in assembler

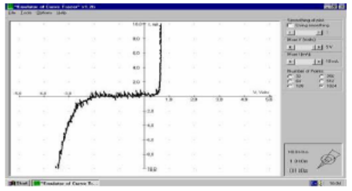
Implement semiconductor NN

Simulate electronic circuits





Develop automation and visualization tools

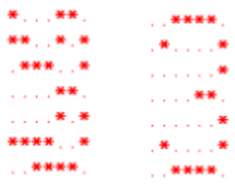


Hardware implementation: too expensive and inflexible

Software implementation: too slow

Need more automation, realistic data sets and further optimization

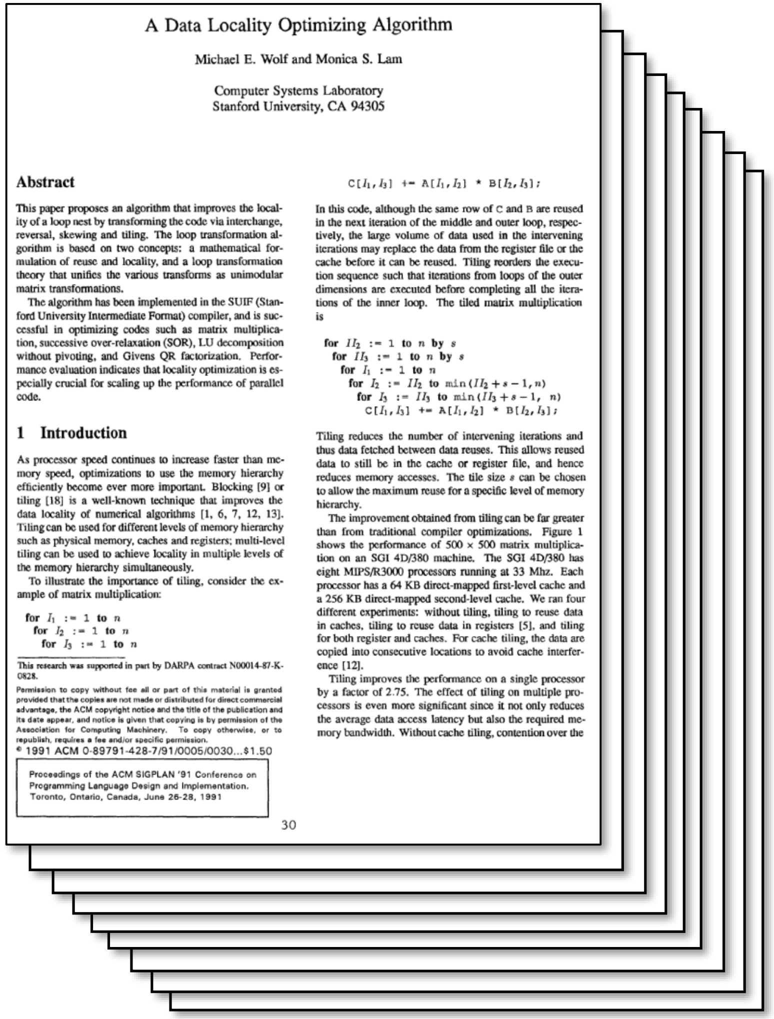
Test in production



My original background was in physics and electronics before switching to computer science

Why do I care about reproducibility, replicability and technology transfer?

My 2nd R&D project: implement self-optimizing compiler using auto-tuning and machine learning and test it in the real world. Started in 1999.



A Data Locality Optimizing Algorithm

Michael E. Wolf and Monica S. Lam

Computer Systems Laboratory
Stanford University, CA 94305

Abstract

This paper proposes an algorithm that improves the locality of a loop nest by transforming the code via interchange, reversal, skewing and tiling. The loop transformation algorithm is based on two concepts: a mathematical formulation of reuse and locality, and a loop transformation theory that unifies the various transforms as unimodular matrix transformations.

The algorithm has been implemented in the SUIF (Stanford University Intermediate Format) compiler, and is successful in optimizing codes such as matrix multiplication, successive over-relaxation (SOR), LU decomposition without pivoting, and Givens QR factorization. Performance evaluation indicates that locality optimization is especially crucial for scaling up the performance of parallel code.

1 Introduction

As processor speed continues to increase faster than memory speed, optimizations to use the memory hierarchy efficiently become ever more important. Blocking [9] or tiling [18] is a well-known technique that improves the data locality of numerical algorithms [1, 6, 7, 12, 13]. Tiling can be used for different levels of memory hierarchy such as physical memory, caches and registers; multi-level tiling can be used to achieve locality in multiple levels of the memory hierarchy simultaneously.

To illustrate the importance of tiling, consider the example of matrix multiplication:

```
for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1,I3] += A[I1,I2] * B[I2,I3];
```

In this code, although the same row of C and B are reused in the next iteration of the middle and outer loop, respectively, the large volume of data used in the intervening iterations may replace the data from the register file or the cache before it can be reused. Tiling reorders the execution sequence such that iterations from loops of the outer dimensions are executed before completing all the iterations of the inner loop. The tiled matrix multiplication is

```
for II2 := 1 to n by s
  for II3 := 1 to n by s
    for I1 := 1 to n
      for I2 := II2 to min(II2+s-1,n)
        for I3 := II3 to min(II3+s-1, n)
          C[I1,I3] += A[I1,I2] * B[I2,I3];
```

Tiling reduces the number of intervening iterations and thus data fetched between data reuses. This allows reused data to still be in the cache or register file, and hence reduces memory accesses. The tile size s can be chosen to allow the maximum reuse for a specific level of memory hierarchy.

The improvement obtained from tiling can be far greater than from traditional compiler optimizations. Figure 1 shows the performance of 500 × 500 matrix multiplication on an SGI 4D/380 machine. The SGI 4D/380 has eight MIPS/R3000 processors running at 33 Mhz. Each processor has a 64 KB direct-mapped first-level cache and a 256 KB direct-mapped second-level cache. We ran four different experiments: without tiling, tiling to reuse data in caches, tiling to reuse data in registers [5], and tiling for both register and caches. For cache tiling, the data are copied into consecutive locations to avoid cache interference [12].

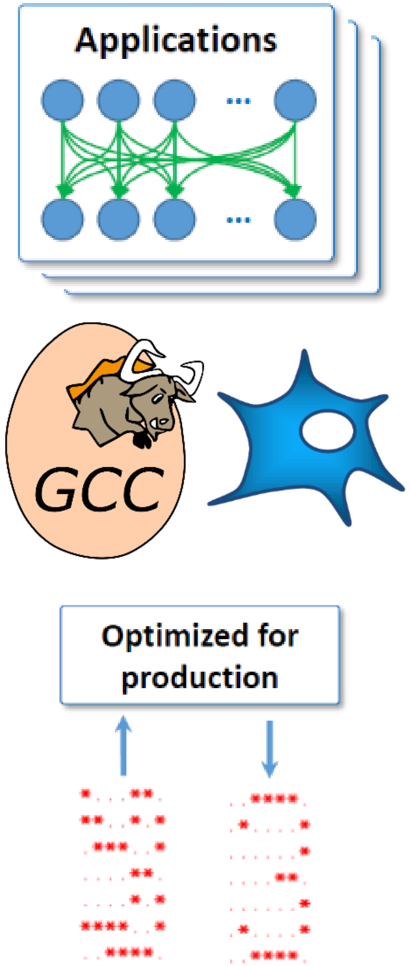
Tiling improves the performance on a single processor by a factor of 2.75. The effect of tiling on multiple processors is even more significant since it not only reduces the average data access latency but also the required memory bandwidth. Without cache tiling, contention over the

This research was supported in part by DARPA contract N00014-87-K-0828.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1991 ACM 0-89791-428-7/91/0005/0030...$1.50

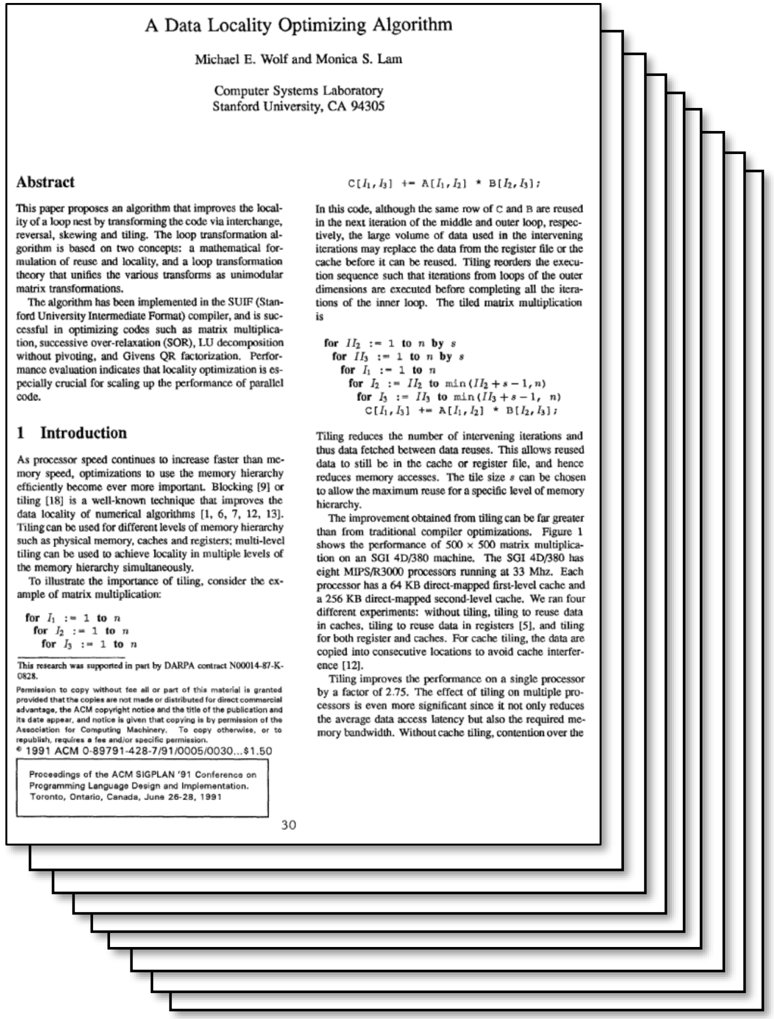
Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation. Toronto, Ontario, Canada, June 26-28, 1991

30



Why do I care about reproducibility, replicability and technology transfer?

My 2nd R&D project: implement self-optimizing compiler using auto-tuning and machine learning and test it in the real world. Started in 1999.



A Data Locality Optimizing Algorithm

Michael E. Wolf and Monica S. Lam

Computer Systems Laboratory
Stanford University, CA 94305

Abstract

This paper proposes an algorithm that improves the locality of a loop nest by transforming the code via interchange, reversal, skewing and tiling. The loop transformation algorithm is based on two concepts: a mathematical formulation of reuse and locality, and a loop transformation theory that unifies the various transforms as unimodular matrix transformations.

The algorithm has been implemented in the SUIF (Stanford University Intermediate Format) compiler, and is successful in optimizing codes such as matrix multiplication, successive over-relaxation (SOR), LU decomposition without pivoting, and Givens QR factorization. Performance evaluation indicates that locality optimization is especially crucial for scaling up the performance of parallel code.

1 Introduction

As processor speed continues to increase faster than memory speed, optimizations to use the memory hierarchy efficiently become ever more important. Blocking [9] or tiling [18] is a well-known technique that improves the data locality of numerical algorithms [1, 6, 7, 12, 13]. Tiling can be used for different levels of memory hierarchy such as physical memory, caches and registers; multi-level tiling can be used to achieve locality in multiple levels of the memory hierarchy simultaneously.

To illustrate the importance of tiling, consider the example of matrix multiplication:

```
for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1,I3] += A[I1,I2] * B[I2,I3];
```

In this code, although the same row of C and B are reused in the next iteration of the middle and outer loop, respectively, the large volume of data used in the intervening iterations may replace the data from the register file or the cache before it can be reused. Tiling reorders the execution sequence such that iterations from loops of the outer dimensions are executed before completing all the iterations of the inner loop. The tiled matrix multiplication is

```
for II2 := 1 to n by s
  for II3 := 1 to n by s
    for I1 := 1 to n
      for I2 := II2 to min(II2+s-1,n)
        for I3 := II3 to min(II3+s-1, n)
          C[I1,I3] += A[I1,I2] * B[I2,I3];
```

Tiling reduces the number of intervening iterations and thus data fetched between data reuses. This allows reused data to still be in the cache or register file, and hence reduces memory accesses. The tile size s can be chosen to allow the maximum reuse for a specific level of memory hierarchy.

The improvement obtained from tiling can be far greater than from traditional compiler optimizations. Figure 1 shows the performance of 500 × 500 matrix multiplication on an SGI 4D/380 machine. The SGI 4D/380 has eight MIPS/R3000 processors running at 33 Mhz. Each processor has a 64 KB direct-mapped first-level cache and a 256 KB direct-mapped second-level cache. We ran four different experiments: without tiling, tiling to reuse data in caches, tiling to reuse data in registers [5], and tiling for both register and caches. For cache tiling, the data are copied into consecutive locations to avoid cache interference [12].

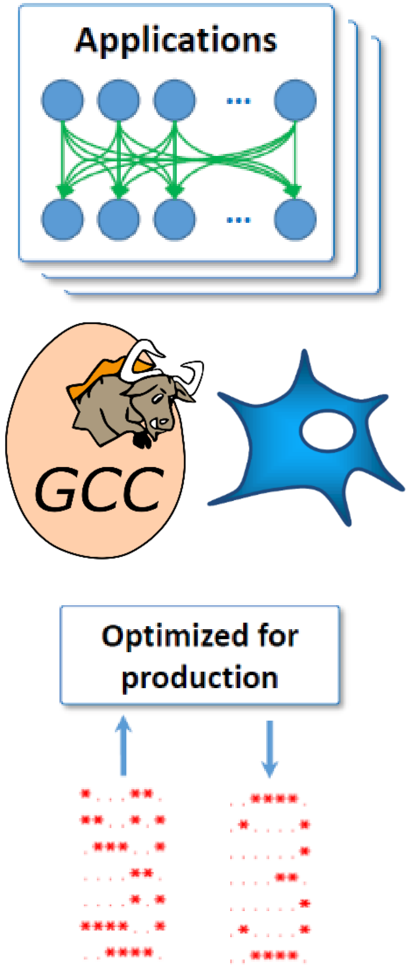
Tiling improves the performance on a single processor by a factor of 2.75. The effect of tiling on multiple processors is even more significant since it not only reduces the average data access latency but also the required memory bandwidth. Without cache tiling, contention over the

This research was supported in part by DARPA contract N00014-87-K-0828.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1991 ACM 0-89791-428-7/91/0005/0030...$1.50

Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation. Toronto, Ontario, Canada, June 26-28, 1991

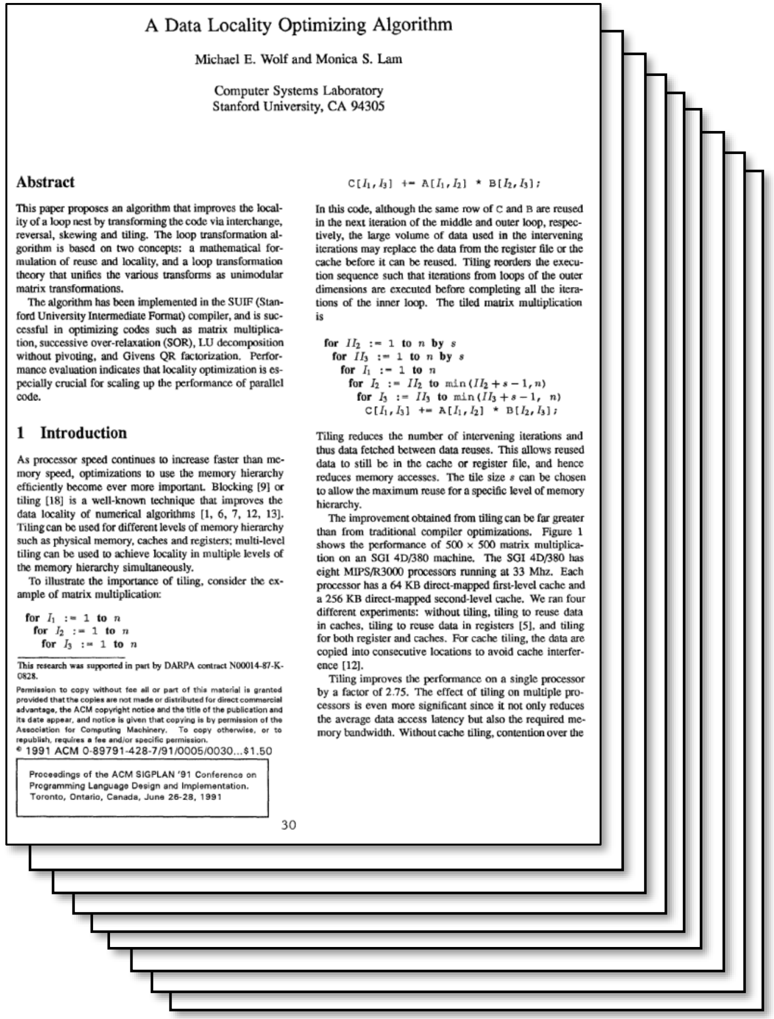
30



Created cTuning.org platform to crowd-benchmark programs and crowd-train ML models inside a compiler to predict the most efficient optimizations.

Why do I care about reproducibility, replicability and technology transfer?

My 2nd R&D project: implement self-optimizing compiler using auto-tuning and machine learning and test it in the real world.
Started in 1999 and finished in 2009 while exposing lots of issues …



A Data Locality Optimizing Algorithm

Michael E. Wolf and Monica S. Lam

Computer Systems Laboratory
Stanford University, CA 94305

Abstract

This paper proposes an algorithm that improves the locality of a loop nest by transforming the code via interchange, reversal, skewing and tiling. The loop transformation algorithm is based on two concepts: a mathematical formulation of reuse and locality, and a loop transformation theory that unifies the various transforms as unimodular matrix transformations.

The algorithm has been implemented in the SUIF (Stanford University Intermediate Format) compiler, and is successful in optimizing codes such as matrix multiplication, successive over-relaxation (SOR), LU decomposition without pivoting, and Givens QR factorization. Performance evaluation indicates that locality optimization is especially crucial for scaling up the performance of parallel code.

1 Introduction

As processor speed continues to increase faster than memory speed, optimizations to use the memory hierarchy efficiently become ever more important. Blocking [9] or tiling [18] is a well-known technique that improves the data locality of numerical algorithms [1, 6, 7, 12, 13]. Tiling can be used for different levels of memory hierarchy such as physical memory, caches and registers; multi-level tiling can be used to achieve locality in multiple levels of the memory hierarchy simultaneously.

To illustrate the importance of tiling, consider the example of matrix multiplication:

```
for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1,I3] += A[I1,I2] * B[I2,I3];
```

In this code, although the same row of C and B are reused in the next iteration of the middle and outer loop, respectively, the large volume of data used in the intervening iterations may replace the data from the register file or the cache before it can be reused. Tiling reorders the execution sequence such that iterations from loops of the outer dimensions are executed before completing all the iterations of the inner loop. The tiled matrix multiplication is

```
for II2 := 1 to n by s
  for II3 := 1 to n by s
    for I1 := 1 to n
      for I2 := II2 to min(II2+s-1,n)
        for I3 := II3 to min(II3+s-1, n)
          C[I1,I3] += A[I1,I2] * B[I2,I3];
```

Tiling reduces the number of intervening iterations and thus data fetched between data reuses. This allows reused data to still be in the cache or register file, and hence reduces memory accesses. The tile size s can be chosen to allow the maximum reuse for a specific level of memory hierarchy.

The improvement obtained from tiling can be far greater than from traditional compiler optimizations. Figure 1 shows the performance of 500 × 500 matrix multiplication on an SGI 4D/380 machine. The SGI 4D/380 has eight MIPS/R3000 processors running at 33 Mhz. Each processor has a 64 KB direct-mapped first-level cache and a 256 KB direct-mapped second-level cache. We ran four different experiments: without tiling, tiling to reuse data in caches, tiling to reuse data in registers [5], and tiling for both register and caches. For cache tiling, the data are copied into consecutive locations to avoid cache interference [12].

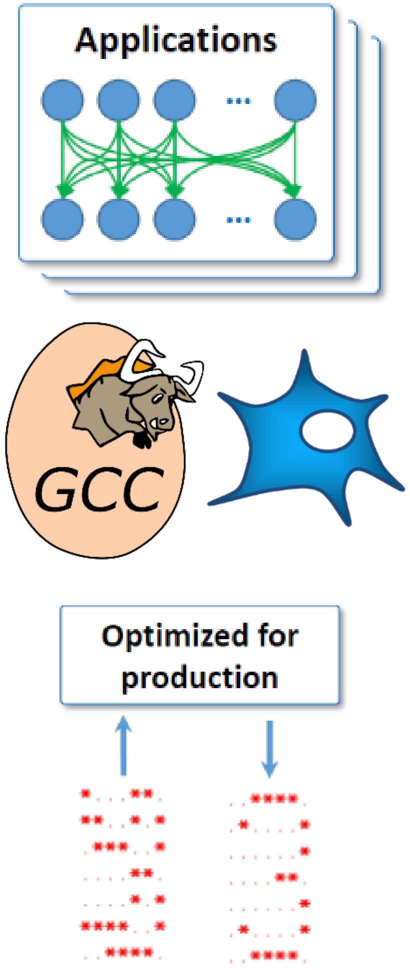
Tiling improves the performance on a single processor by a factor of 2.75. The effect of tiling on multiple processors is even more significant since it not only reduces the average data access latency but also the required memory bandwidth. Without cache tiling, contention over the

This research was supported in part by DARPA contract N00014-87-K-0828.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1991 ACM 0-89791-428-7/91/0005/0030...$1.50

Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation. Toronto, Ontario, Canada, June 26-28, 1991

30



Created **cTuning.org platform to crowd-benchmark programs and crowd-train ML models inside a compiler to predict the most efficient optimizations.**

Major problems:

- Not enough information to reproduce related research papers and build upon them.
- Difficult/impossible to reproduce performance numbers from the community across continuously changing software and hardware.
- Unlike physics, no common experimental methodology and tools to measure and compare performance (and other metrics) across different research papers and projects.
- Not enough benchmarks and data sets to train my models – papers rarely share their artifacts.

2009-2014: persuading community to share and evaluate artifacts for published papers

reproducibility.cs.arizona.edu – weak reproducibility

A comprehensive study of ~600 papers to examine if related code was shared and can be built.

evaluate.inf.usi.ch/artifacts , artifact-eval.org – strong reproducibility

The original and successful introduction of the artifact evaluation process at ACM conferences.

Artifacts are evaluated after papers are accepted and before the camera-ready deadline.

Paper receive the reproducibility badge only if the related artifact is consistent, complete, well documented and easy to reuse.

cTuning.org – experimented with different incentives to participate in artifact evaluation

Cooperative process between authors and evaluators to learn how to reproduce/replicate results.

Try new publication models with open reviewing: arxiv.org/pdf/1406.4020.pdf , adapt-workshop.org .

Learn how to unify and automate this process particularly for empirical results (performance, power) similar to physics.

Learn how to make it easier to transfer research to production with the latest/different software, hardware and data.

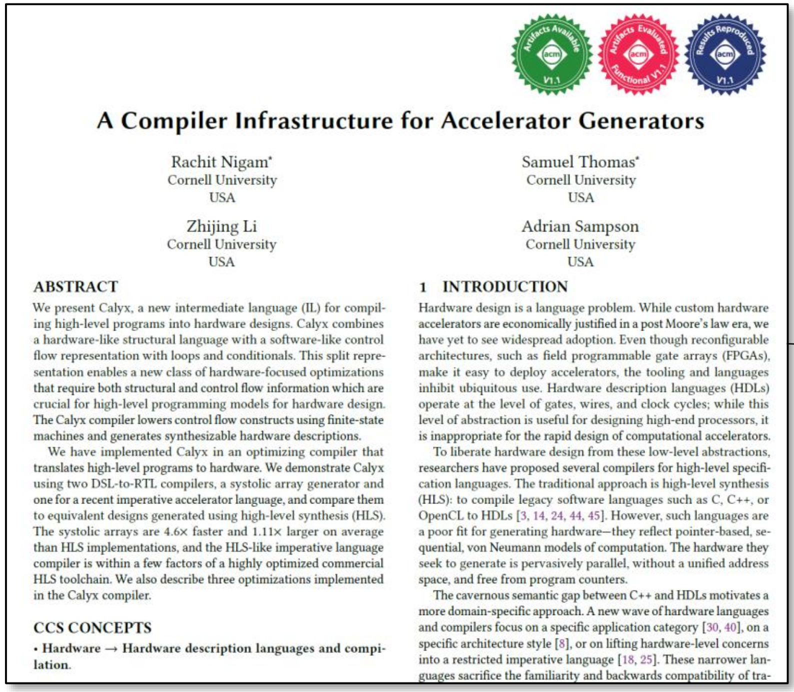
Bruce R. Childers, Grigori Fursin, Shriram Krishnamurthi, Andreas Zeller:
Artifact Evaluation for Publications (Dagstuhl Perspectives Workshop 15452). Dagstuhl Reports 5(11): 29-35 (2015)

2014 - established cTuning foundation: a non-profit R&D organization headquartered in Paris

- Goal: connect academia and industry to develop a common methodology and tools to reproduce research projects and bring them to the real world
- Helped to prepare and unify ACM artifact reviewing and badging methodology: https://www.acm.org/publications/policies/artifact-review-and-badging-current
- Helped to organize artifact evaluation at 15+ ACM/IEEE conferences including CGO, PPoPP, ASPLOS, PACT, MLSys, MICRO …

2014 - established cTuning foundation: a non-profit R&D organization headquartered in Paris

- Goal: connect academia and industry to develop a common methodology and tools to reproduce research projects and bring them to the real world
- Helped to prepare and unify ACM artifact reviewing and badging methodology: https://www.acm.org/publications/policies/artifact-review-and-badging-current
- Helped to organize artifact evaluation at 15+ ACM/IEEE conferences including CGO, PPoPP, ASPLOS, PACT, MLSys, MICRO …
- Introduced Artifact Appendix and Checklist (was picked up by other conferences including SuperComputing since then)
  - https://cTuning.org/ae/appendix.html
  - https://cTuning.org/ae/checklist.html



A Compiler Infrastructure for Accelerator Generators

Rachit Nigam* Cornell University USA
Samuel Thomas* Cornell University USA
Zhijing Li Cornell University USA
Adrian Sampson Cornell University USA

ABSTRACT

We present Calyx, a new intermediate language (IL) for compiling high-level programs into hardware designs. Calyx combines a hardware-like structural language with a software-like control flow representation with loops and conditionals. This split representation enables a new class of hardware-focused optimizations that require both structural and control flow information which are crucial for high-level programming models for hardware design. The Calyx compiler lowers control flow constructs using finite-state machines and generates synthesizable hardware descriptions.

We have implemented Calyx in an optimizing compiler that translates high-level programs to hardware. We demonstrate Calyx using two DSL-to-RTL compilers, a systolic array generator and one for a recent imperative accelerator language, and compare them to equivalent designs generated using high-level synthesis (HLS). The systolic arrays are 4.6× faster and 1.11× larger on average than HLS implementations, and the HLS-like imperative language compiler is within a few factors of a highly optimized commercial HLS toolchain. We also describe three optimizations implemented in the Calyx compiler.

CCS CONCEPTS

• Hardware → Hardware description languages and compilation.

1 INTRODUCTION

Hardware design is a language problem. While custom hardware accelerators are economically justified in a post Moore's law era, we have yet to see widespread adoption. Even though reconfigurable architectures, such as field programmable gate arrays (FPGAs), make it easy to deploy accelerators, the tooling and languages inhibit ubiquitous use. Hardware description languages (HDLs) operate at the level of gates, wires, and clock cycles; while this level of abstraction is useful for designing high-end processors, it is inappropriate for the rapid design of computational accelerators.

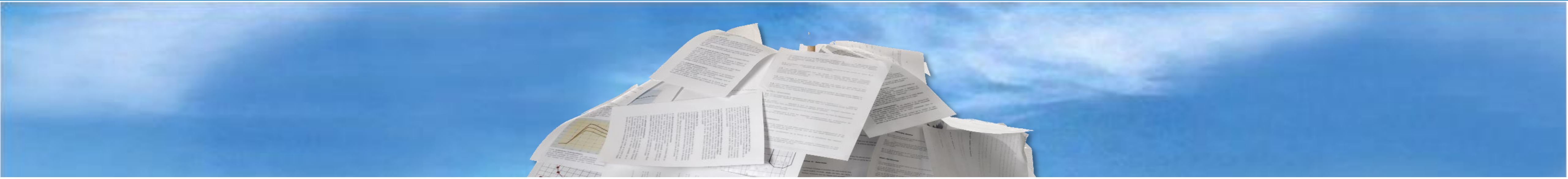
To liberate hardware design from these low-level abstractions, researchers have proposed several compilers for high-level specification languages. The traditional approach is high-level synthesis (HLS): to compile legacy software languages such as C, C++, or OpenCL to HDLs [3, 14, 24, 44, 45]. However, such languages are a poor fit for generating hardware—they reflect pointer-based, sequential, von Neumann models of computation. The hardware they seek to generate is pervasively parallel, without a unified address space, and free from program counters.

The cavernous semantic gap between C++ and HDLs motivates a more domain-specific approach. A new wave of hardware languages and compilers focus on a specific application category [30, 40], on a specific architecture style [8], or on lifting hardware-level concerns into a restricted imperative language [18, 25]. These narrower languages sacrifice the familiarity and backwards compatibility of tra-

Artifact Appendix (up to 2 pages)

1. Abstract
2. **Artifact check-list**
3. How to obtain?
4. Prepare software
5. Prepare hardware
6. Prepare data sets
7. Prepare models
8. Installation
9. Experiment workflow
10. Evaluation and expected result
11. Notes

Learnings from evaluating artifacts and reproducing results from 200+ research papers



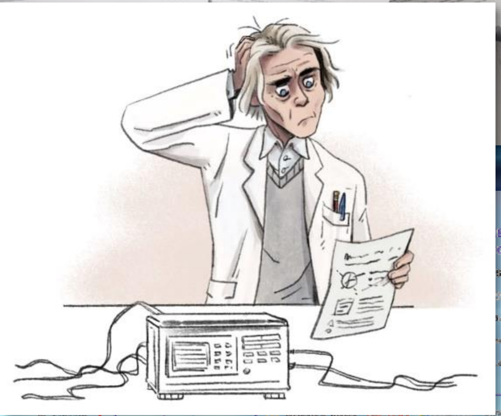
Feedback from the authors

- Still takes a few weeks to prepare artifacts, containers, Jupyter notebooks and write Artifact Appendix
- Still considered as a painful and one-shot experience (even waste of time if researcher doesn’t plan to continue)
- Still need to write their own tools for measurements, experiments and visualization
- Find criteria for artifact reusability is very vague

Feedback from the evaluators

- Can take weeks of painful and repetitive interactions between teams to
  - decrypt Artifact Appendices, README files, scripts and containers to understand how to use them
  - measure performance outside containers or on a different system with different software and hardware
  - visualize, compare and validate results (often manually)
  - ensure apple-to-apple comparison of results from other papers with different set of artifacts and tools (mini-artifact evaluation for other papers)
- Find criteria for artifact reusability is very vague

cKnowledge.org: learnings from validating research projects in the real world



Can take months to prepare for production
and find trade-off between other characteristics and operational costs
(accuracy vs latency vs throughput vs power consumption vs training/usage costs)

Different set of (DevOps/MLOps) tools
Rapidly evolving software and hardware
Different data

Industry doesn't have time to decrypt numerous papers even with Artifact Appendix, ad-hoc containers and Jupyter notebooks

If it doesn't work within 30 min, will unlikely want to fix it

Can traditional workflow automation frameworks help?

Workflows Community Summits (January and April, 2021)

arxiv.org/abs/2110.02168

A Community Roadmap for Scientific Workflows Research and Development

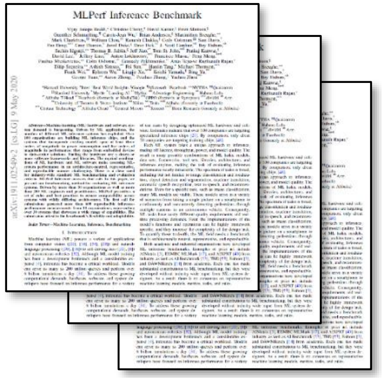
Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, Bartosz Balis, Tainã Coleman, Frederik Coppens, Frank Di Natale, Bjoern Enders, Thomas Fahringer, Rosa Filgueira, Grigori Fursin, Daniel Garijo, Carole Goble, Dorran Howell, Shantenu Jha, Daniel S. Katz, Daniel Laney, Ulf Leser, Maciej Malawski, Kshitij Mehta, Loïc Pottier, Jonathan Ozik, J. Luc Peterson, Lavanya Ramakrishnan, Stian Soiland-Reyes, Douglas Thain, Matthew Wolf

Challenges:

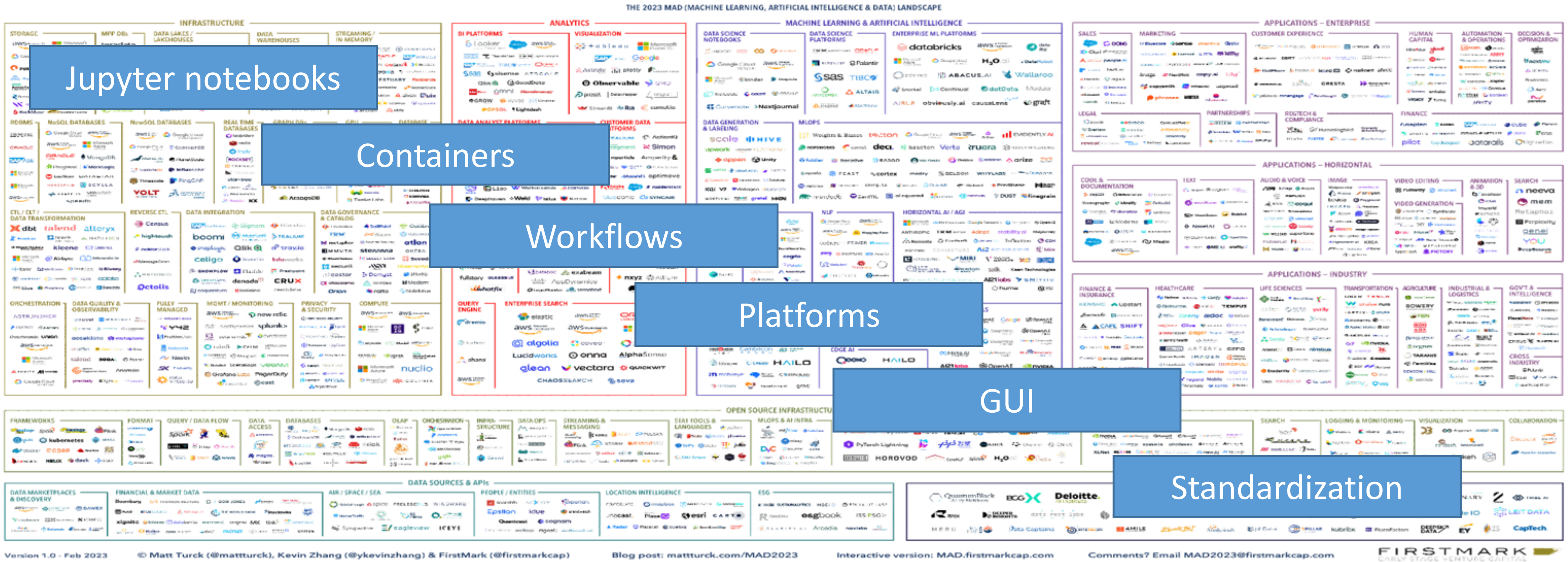
- Already more than 100 workflow frameworks and tools out there
- Researchers do not have time to learn complex automation platforms and workflow framework for experiment automation
- If researchers move to industry, they likely have totally different tools and workflow automation frameworks
- Artifact evaluators do not want to learn different workflow frameworks for different papers

Maybe we are trying to solve the wrong problem?

Papers, READMEs
Artifact Appendices



Describes a set
of actions



Jupyter notebooks

Containers

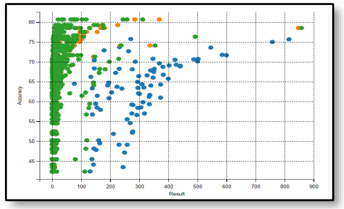
Workflows

Platforms

GUI

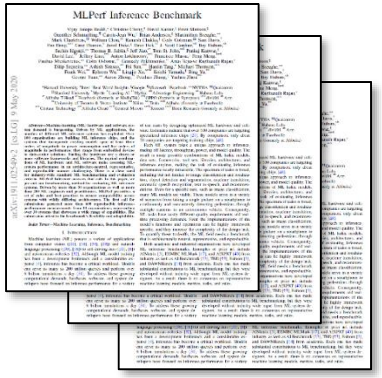
Standardization

to obtain results

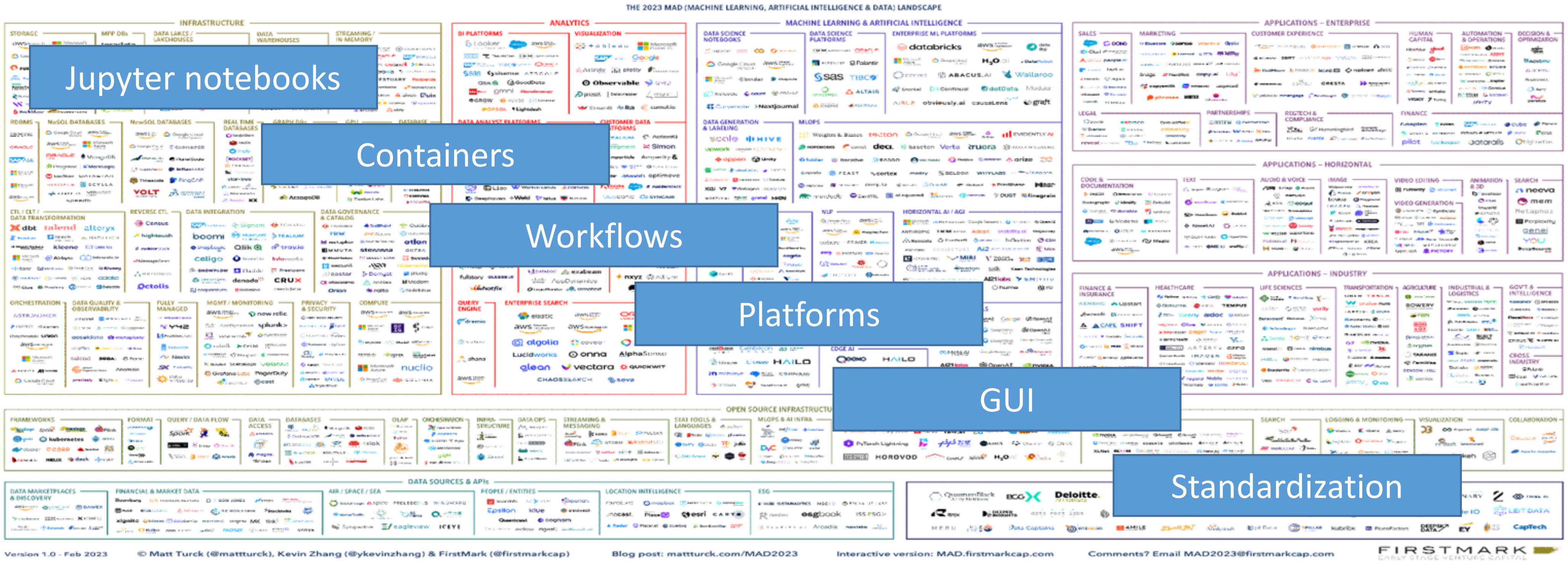


Maybe we are not looking in the right place?

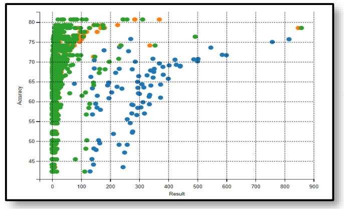
Papers, READMEs
Artifact Appendices



Describes a set of actions

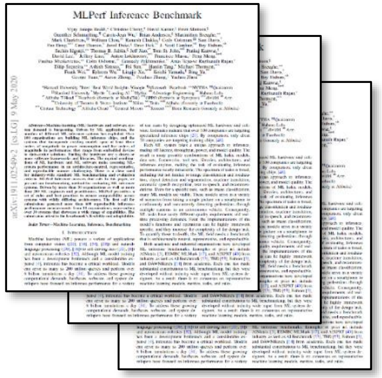


to obtain results



In the end comes down to some OS commands and OS/python scripts to perform the same actions ...

Papers, READMEs
Artifact Appendices

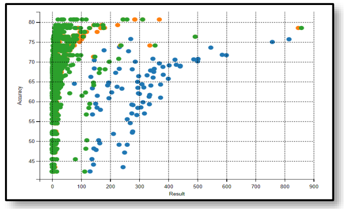


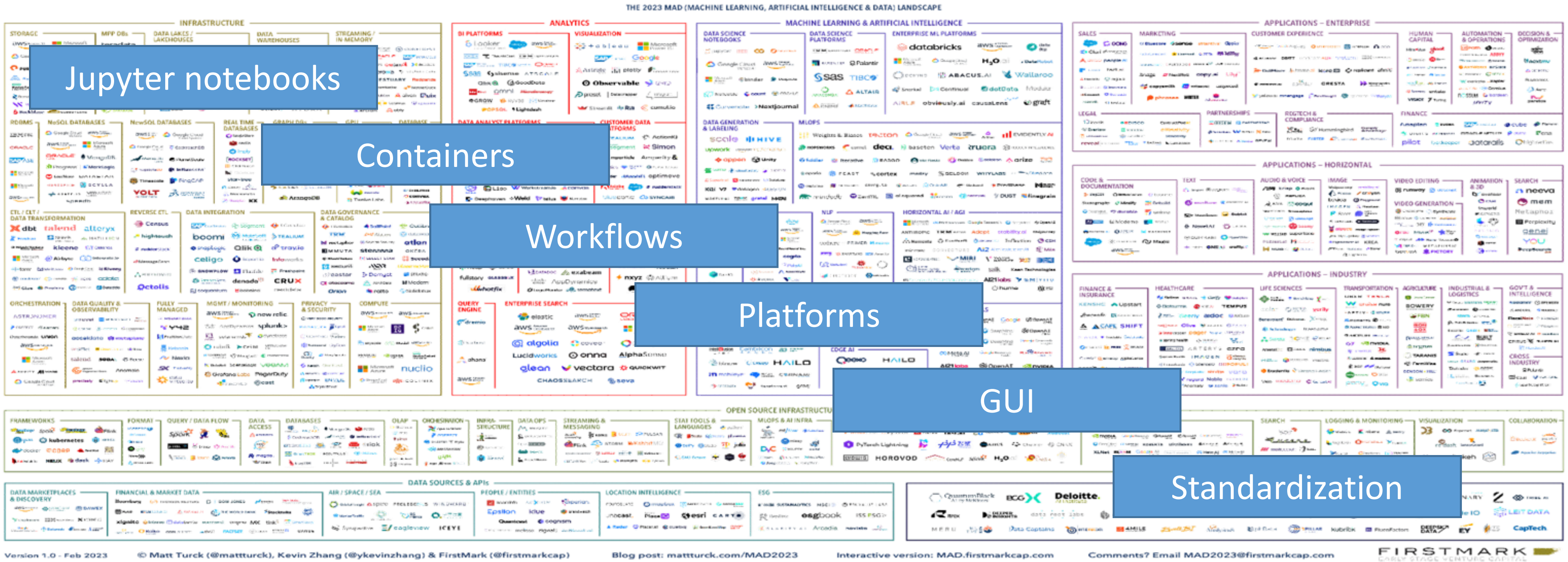
Describes a set of actions

Get model ResNet50 fp32 for image classification

...

to obtain results





Jupyter notebooks

Containers

Workflows

Platforms

GUI

Standardization

Analyzing ~200 research papers with Artifact Appendix ...

Ad-hoc OS commands and OS/python scripts

Input file

Input environment variables

download-resnet50.sh

get-resnet-model-hugging-face.py

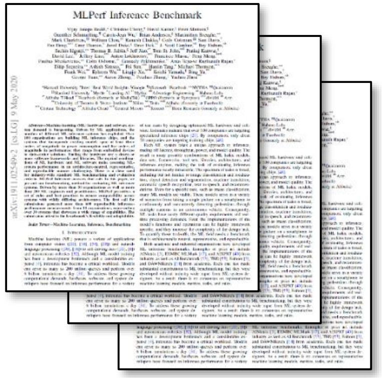
prepare-resnet-50-fp32.bat

Output file

New environment variables

What is we have reusable, portable and tech. agnostic "blocks" to automate any research?

Papers, READMEs
Artifact Appendices

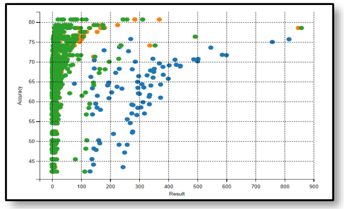


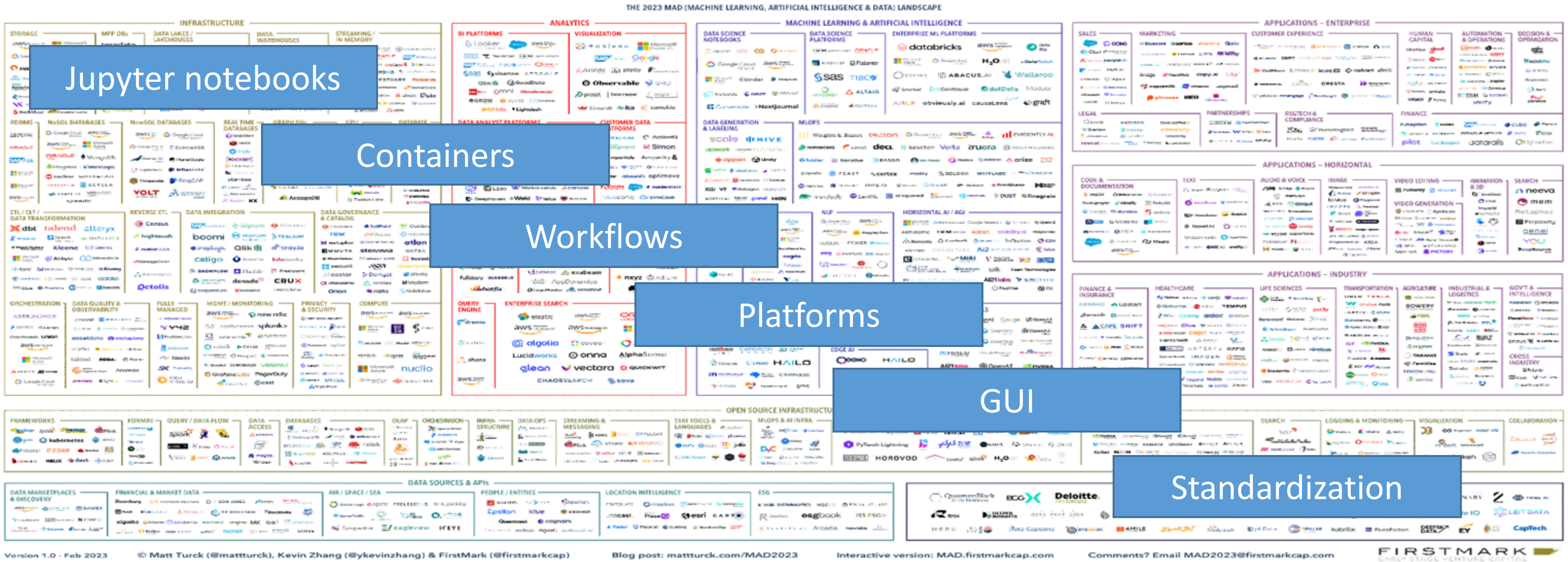
Describes a set of actions

Get model ResNet50 fp32 for image classification

…

to obtain results





Add simple YAML or JSON file with human-readable tags "get,ml-model,resnet50,image-classification"

OS commands and OS/python scripts

Input file | Input environment variables

customize.py

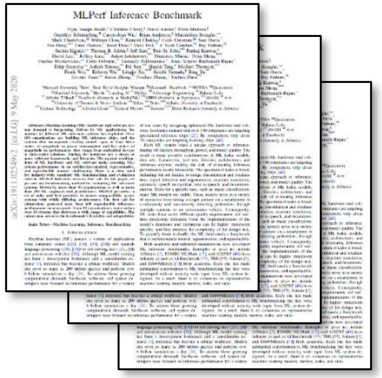
run.sh | run.bat

Output file | New environment variables

mac OS

That can automatically adapt to any platform by setting required env vars and calling some native script?

Papers, READMEs
Artifact Appendices

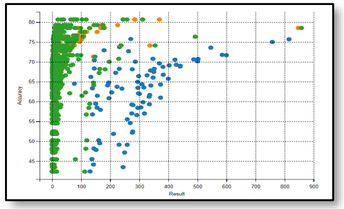


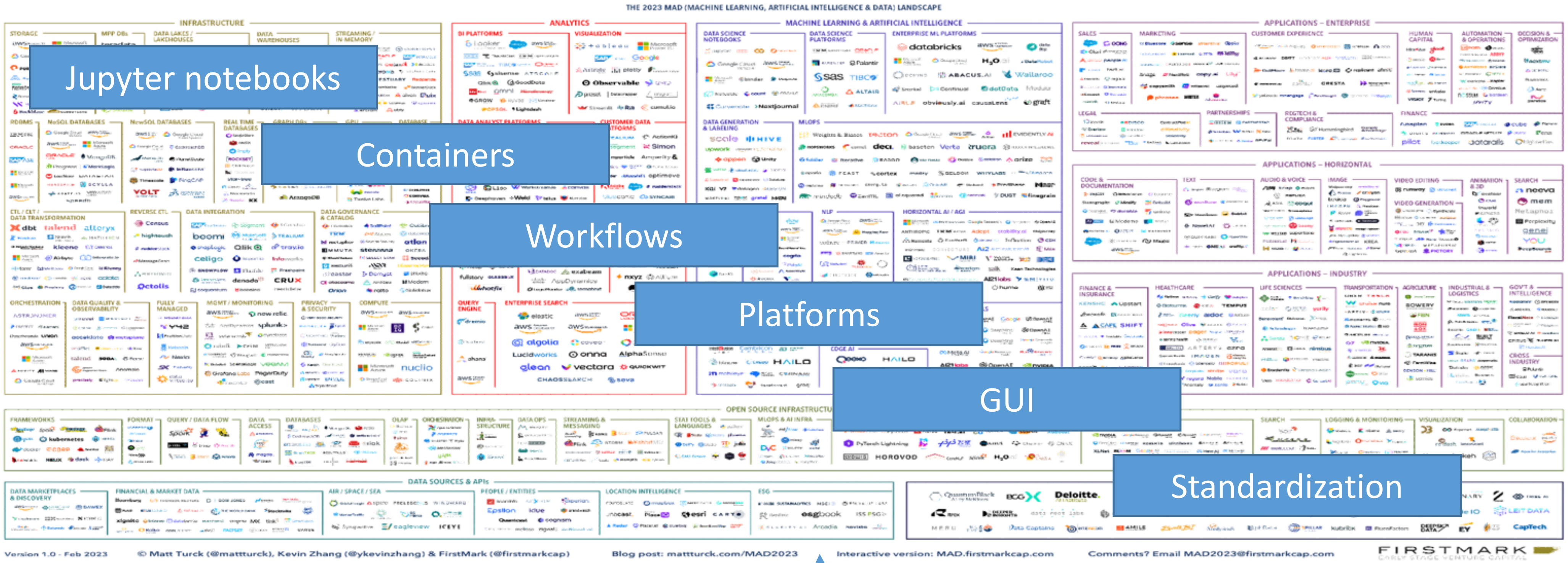
Describes a set of actions

Get model ResNet50 fp32 for image classification

…

to obtain results





Jupyter notebooks

Containers

Workflows

Platforms

GUI

Standardization

Add simple interface to run scripts: ***cm run script --tags=get,ml-model,resnet50 --env.PRECISION=fp32***

Add simple YAML or JSON file with human-readable tags "get,ml-model,resnet50,image-classification"

OS commands and OS/python scripts

Input file | Input environment variables

customize.py

run.sh | run.bat

Output file | New environment variables

2022: started prototyping Collective Mind interface (CM) to unify access to native scripts

Can create CM script by simply adding **_cm.json** or **_cm.yaml** with just a few keys in a project and marking it for CM search

Example: https://github.com/mlcommons/ck/tree/master/cm-mlops

script/get-ml-model-resnet50/**_cm.json**

```
{
  "tags": ["get",  "raw“, "ml-model",  "resnet50", "ml-model-resnet50", "image-classification"],
  "uid": "56203e4e998b4bc0",
  "alias": "get-ml-model-resnet50“
  "variations": {"fp32": { "env": {“CM_ML_MODEL_PRECISION": "fp32“ …
  …
```

.cmr.yaml

```
alias: mlcommons@ck
uid: a4705959af8e447a
```

2022: started prototyping Collective Mind interface (CM) to unify access to native scripts

Can create CM script by simply adding **_cm.json** or **_cm.yaml** with just a few keys in a project and marking it for CM search

Example: https://github.com/mlcommons/ck/tree/master/cm-mlops

script/get-ml-model-resnet50/**_cm.json**

```
{
  "tags": ["get",  "raw“, "ml-model",  "resnet50", "ml-model-resnet50", "image-classification"],
   "uid": "56203e4e998b4bc0",
   "alias": "get-ml-model-resnet50“
  "variations": {"fp32": { "env": {“CM_ML_MODEL_PRECISION": "fp32“ …
  …
```

.cmr.yaml

```
alias: mlcommons@ck
uid: a4705959af8e447a
```

Need simple Python library (**cmind**) with min requirements (Python 3+, git, wget) and unified/human-readable CLI (**cm**)

github.com/mlcommons/ck/blob/master/docs

```
python3 –m pip install cmind
cm pull repo mlcommons@ck        # Clone repository to $HOME/CM/repos
cm run script --tags=get,ml-model,resnet50,_fp32,_onnx --json --save_env
   or    cm run script “get ml-model resnet50 _fp32” --json -v
   or    cm run script 56203e4e998b4bc0 --json
   or    cm run script get-ml-model-resnet50 --json
```

Teamed up with Arjun Suresh to add common scripts from ML and Systems papers

https://github.com/mlcommons/ck/blob/master/docs/list_of_scripts.md

https://github.com/mlcommons/ck/tree/master/cm-mlops/script

First unify scripts to prepare experimental setup

Attempt to detect artifact/package and install if missing
(to automatically adapt experiments to a given platform)

Fix portability issues and non-determinism across diverse OS (Ubuntu, MacOS, Red Hat, Windows …) and hardware (x64, Arm64, Nvidia GPUs …)

cm run script “detect os” --json

cm run script “detect cpu” --json

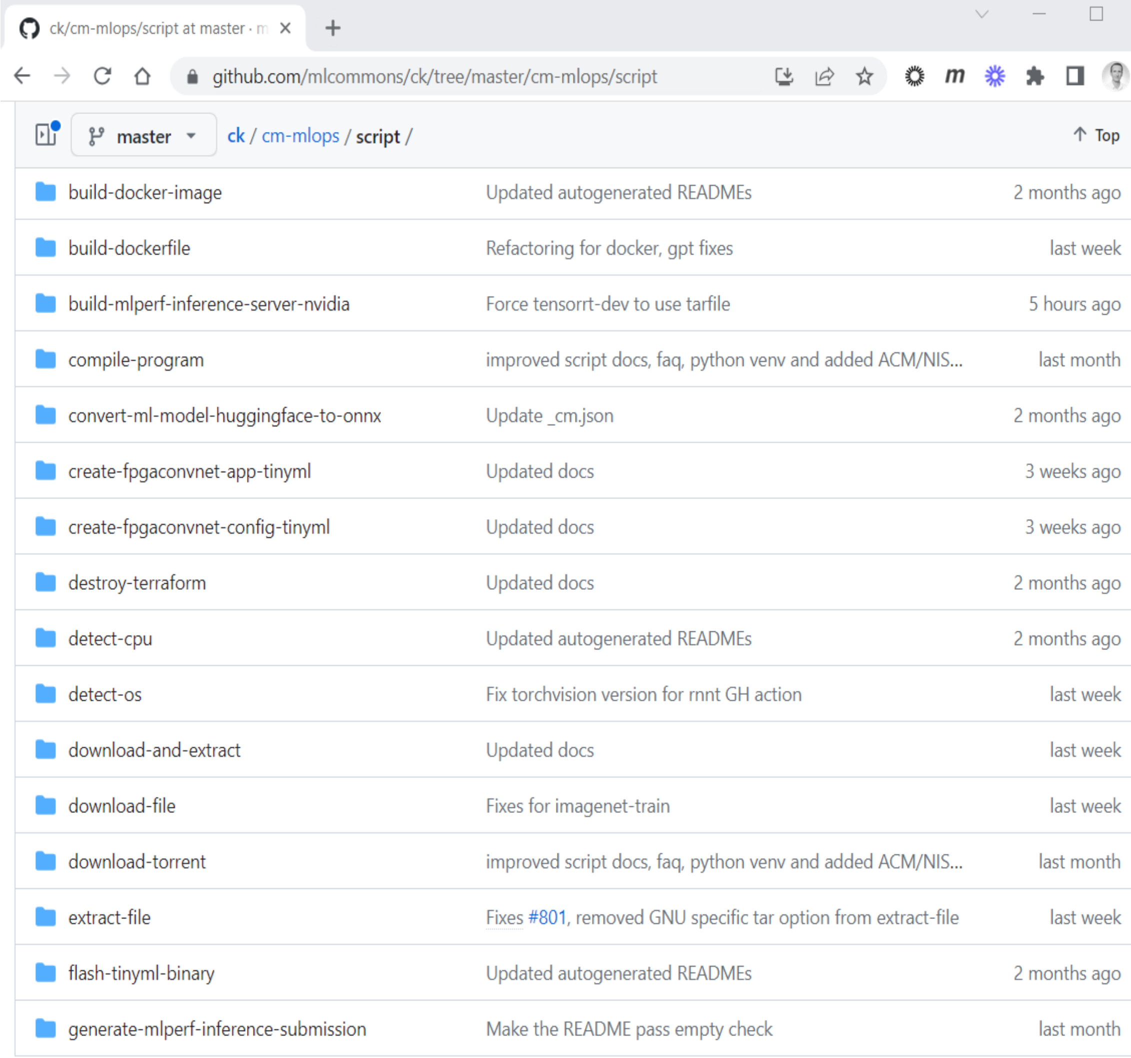
cm run script “download file” --url=https://zenodo.org/record/4735647/files/resnet50_v1.onnx

cm run script “get python3”

cm run script “install python-venv” --name=my-cool-project

cm run script "get generic-python-lib _onnxruntime" --version_min=1.10.0

cm run script “get cuda”



Added simple database functions to find and manage reusable scripts in R&D projects

python3 –m pip install cmind
cm pull repo mlcommons@ck

List/find scripts by UID, alias and tags:

cm find script
cm ls script
cm find script 5b4e0237da074764
*cm find script *-ml-model-**
cm find script --tags=resnet50

Load meta description of a given script

cm load script get-ml-model-resnet50 --json

Add and run dummy script

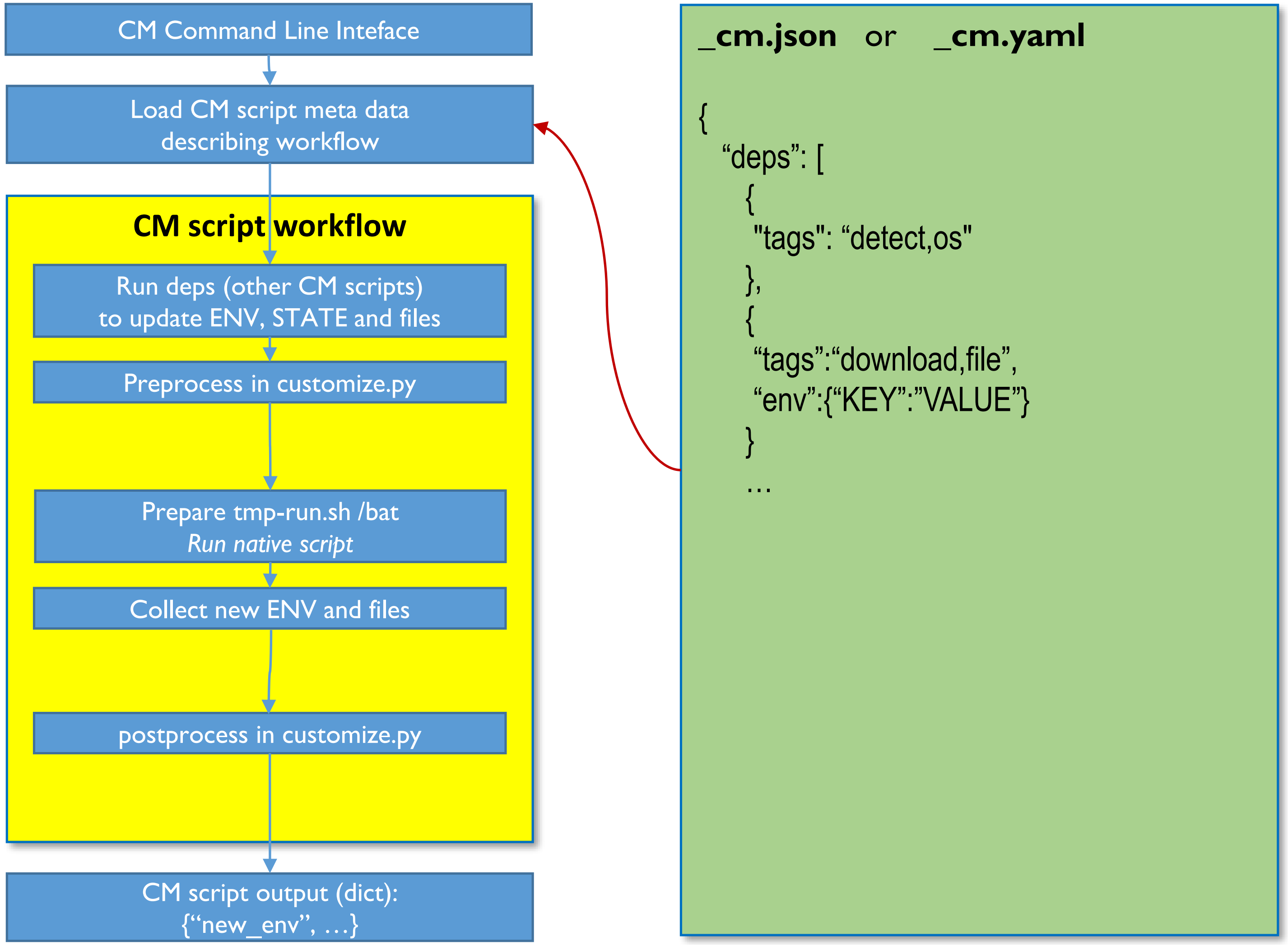
cm add script my-new-cool-script --tags=my,new,cool-script
cm run script --tags=my,new,cool-script --env.KEY=VALUE --json

Delete script

cm rm script --tags=my,new,cool-script
cm delete script --tags=my,new,cool-script

…

Preprocess files and env variables via customize.py and chain multiple CM scripts together



Added functions to cache script output

python3 –m pip install cmind
cm pull repo mlcommons@ck

cm run script "get ml-model resnet50 _fp32 _onnx " --json

Use the same database functions to manage CM "cache"

cm list cache
cm find cache --tags=ml-model,resnet50,_fp32
cm rm cache --tags=ml-model

Clean all cache entries (careful):

cm rm cache -f
cm run script "get ml-model resnet50 _fp32 _onnx " --json
cm run script "get ml-model resnet50 _fp32 _onnx " --json

script/get-ml-model-resnet50/_cm.json

```
{
  "cache":True
  …
```

cache/60d4845558c643aa/**_cm.json**

```
{
   "tags":["ml-model","resnet50","_fp32" …
   …
```

cache/60d4845558c643aa/**resnet50_v1.onnx**

cache/60d4845558c643aa/**cm-cached-state.json**

```
{
  "new_env": {
     "CM_ML_MODEL_FILE_WITH_PATH": "…\\resnet50_v1.onnx"
        …
```

Can convert all R&D projects into a database of reusable artifacts and automations with a common API

Unified and human-readable CM Command Line Interface to access R&D projects:

cm {action} {automation} (artifact name|uid|--tags) @input.json

{action} – is taken from module.py unless a database function to manage related artifacts

Unified and human-readable CM Python interface to access R&D projects:

```
import cmind

r = cmind.access({'action':'run',
                  'automation':'script',
                  'artifact':'get-ml-model-resnet50',
                  'out':'con'})
if r['return']>0: cmind.error(r)
```

{artifact}/{artifact name}/_cm.json and/or _cm.yaml

```
{
  "automation_uid":"artifact",
  "automation_alias":" aea483bd635b49f5",
  "alias":"artifact name",
  "uid":"23f332d0a3ef428f",
  "tags":[…
```

automation/{artifact}/_cm.json and/or _cm.yaml

```
{
  "alias":"artifact"
  "uid":"aea483bd635b49f5"
  …
```

module.py

```
def action(input):
    return {'return':0, …}
      or
    return {'return':1, 'error':'some error'}
```

https://github.com/mlcommons/ck/tree/master/cm-mlops/automation

Implemented modular and portable image classification pipeline that can run on any platform

Fully automatic, no need to fix paths, adapts to your platform, can run on CPU or GPU, can trace information flow

https://github.com/mlcommons/ck/blob/master/docs/tutorials/modular-image-classification.md

_cm.json or _cm.yaml


```yaml
alias: app-image-classification-onnx-py
uid: 3d5e908e472b417e

automation_alias: script
automation_uid: 5b4e0237da074764

tags:
- app
- image-classification
- onnx
- python

deps:
- tags: detect,os
- tags: get,sys-utils-cm
- names:
  - python
  - python3
  tags: get,python3
- tags: get,cuda
  names:
  - cuda
  enable_if_env:
    USE_CUDA:
    - yes
```

...

```yaml
- tags: get,dataset,imagenet,image-classification,original
- tags: get,dataset-aux,imagenet-aux,image-classification
- tags: get,ml-model,resnet50,_onnx,image-classification

- tags: get,generic-python-lib,_onnxruntime
  skip_if_env:
    USE_CUDA:
    - yes
- tags: get,generic-python-lib,_onnxruntime_gpu
  enable_if_env:
    USE_CUDA:
    - yes

variations:
  cuda:
    env:
      USE_CUDA: yes
```


Can generate Artifact Appendix or README

python3 –m pip install cmind
cm pull repo mlcommons@ck

cm run script "python app image-classification onnx“

cm run script "detect os" --out=json
cm run script "get python" --version_min=3.9.1
cm run script "install python-venv" --name=my-virtual-env
cm run script "get ml-model resnet50 _onnx _fp32"
cm run script "get original imagenet dataset _2012-500"
cm run script "get generic-python-lib _onnxruntime"
--version=1.12.0

Implemented modular and portable image classification pipeline that can run on any platform

Fully automatic, no need to fix paths, adapts to your platform, can run on CPU or GPU, can trace information flow

https://github.com/mlcommons/ck/blob/master/docs/tutorials/modular-image-classification.md

_cm.json or _cm.yaml


```yaml
alias: app-image-classification-onnx-py
uid: 3d5e908e472b417e

automation_alias: script
automation_uid: 5b4e0237da074764

tags:
- app
- image-classification
- onnx
- python

deps:
- tags: detect,os
- tags: get,sys-utils-cm
- names:
  - python
  - python3
  tags: get,python3
- tags: get,cuda
  names:
  - cuda
  enable_if_env:
    USE_CUDA:
    - yes
```

…

```yaml
- tags: get,dataset,imagenet,image-classification,original
- tags: get,dataset-aux,imagenet-aux,image-classification
- tags: get,ml-model,resnet50,_onnx,image-classification

- tags: get,generic-python-lib,_onnxruntime
  skip_if_env:
    USE_CUDA:
    - yes
- tags: get,generic-python-lib,_onnxruntime_gpu
  enable_if_env:
    USE_CUDA:
    - yes

variations:
  cuda:
    env:
      USE_CUDA: yes
```


Can generate containers / use in Jupyter notebooks

python3 –m pip install cmind
cm pull repo mlcommons@ck

cm run script "python app image-classification onnx“

cm run script "detect os" --out=json
cm run script "get python" --version_min=3.9.1
cm run script "install python-venv" --name=my-virtual-env
cm run script "get ml-model resnet50 _onnx _fp32"
cm run script "get original imagenet dataset _2012-500"
cm run script "get generic-python-lib _onnxruntime"
--version=1.12.0

Can use CM to run multiple experiments, record them and reply them (on-going)

https://github.com/mlcommons/ck/blob/master/cm-mlops/automation/experiment/README-extra.md

cm run experiment --tags=my,experiment,hello-world -- echo "Hello World!"

cm find experiment --tags=my,experiment,hello-world

cm pack repo mlcommons@ck

cm reply experiment --tags=my,experiment,hello-world

cm run script "gui _graph"

Explore multiple variables :

explore.yaml:

```
explore:
  VAR1: [1,2,3]
  VAR2: ["a","b"]
  VAR3: "[2**i for i in range(0,6)]"
```

cm run experiment --tags=my,experiment,hello-world @explore.yaml -- echo "—batch-size={{VAR1}} {{VAR2}} {{VAR3}}"

CM format supports FAIR principles and can be extended in many dimensions …

nature.com/articles/sdata201618

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature > scientific data > comment > article

Open Access | Published: 15 March 2016

The FAIR Guiding Principles for scientific data management and stewardship

Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao & Barend Mons — Show fewer authors

Scientific Data **3**, Article number: 160018 (2016) | Cite this article

An Addendum to this article was published on 19 March 2019

Findability, **A**ccessibility, **I**nteroperability, and **R**euse of digital assets

The FAIR principles emphasize machine-actionability (i.e., the capacity of computational systems to find, access, interoperate, and reuse data with none or minimal human intervention)

CM considers both code and data as digital assets

But what about empirical measurements (performance, power …)?

ML Commons

MLPerf

MLCommons is an open engineering consortium with 50+ SW/HW companies and universities developing a common methodology and tools for apple-to-apple benchmarking, comparison and optimization of ML Systems: mlcommons.org/en/news/mlcommons-launch

cTuning joined as a founding member

Founding Members

Menu

Alibaba Group 阿里巴巴集团	AMD	arm	Bai du 百度	cerebras	DELL EMC
d-MATRIX Intelligence Delivered	FACEBOOK AI	FUJITSU	FURIOSA	GIGA COMPUTING	Google
GRAPHCORE	Hewlett Packard Enterprise	inspur	intel AI	MEDIATEK	Microsoft
mobilint	Myrtle.ai	Nettrix 宁畅	NEUCHIPS	NVIDIA	oppo
Qualcomm		SambaNova	SAMSUNG Exynos	SYNTIANT	Tenstorrent

https://mlcommons.org/en/get-involved

LoadGen from MLPerf inference benchmark ensures realistic usage scenarios and reproducible measurements

https://arxiv.org/abs/1911.02549 https://github.com/mlcommons/inference/tree/master/loadgen

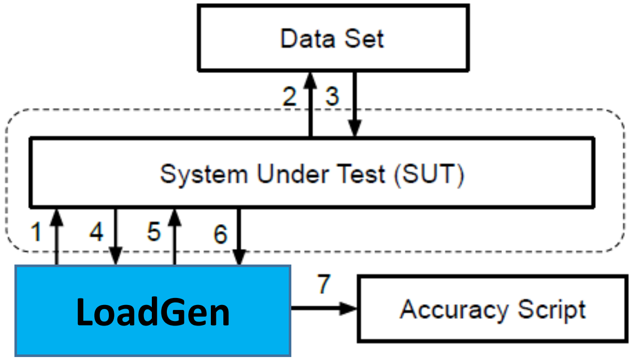


Fig. 3. MLPerf Inference system under test (SUT) and associated components. First, the LoadGen requests that the SUT load samples (1). The SUT then loads samples into memory (2–3) and signals the LoadGen when it is ready (4). Next, the LoadGen issues requests to the SUT (5). The benchmark processes the results and returns them to the LoadGen (6), which then outputs logs for the accuracy script to read and verify (7).

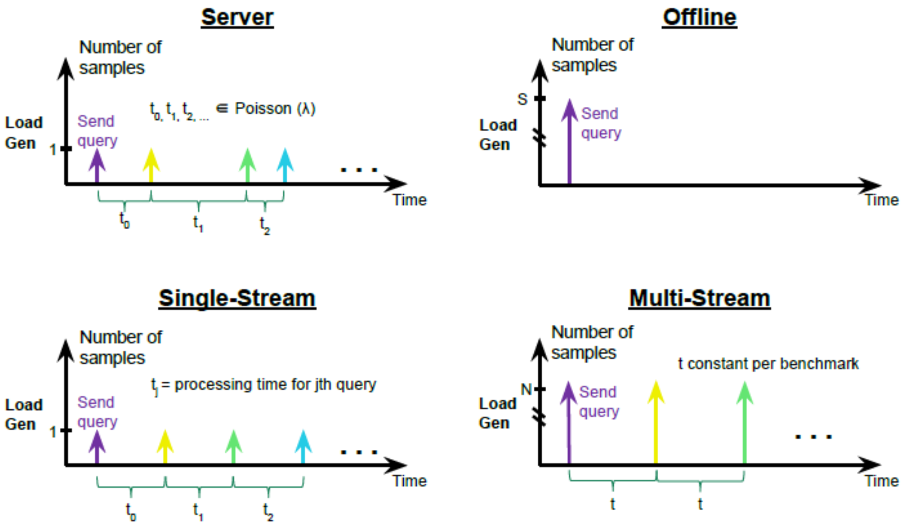
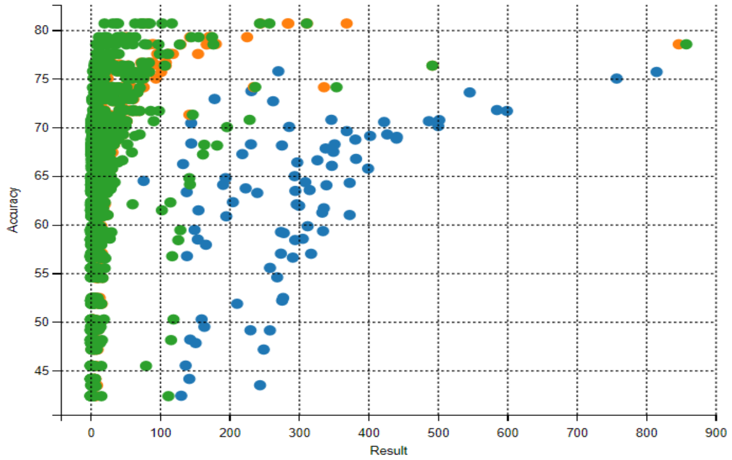


Fig. 4. Timing and number of queries from the LoadGen.

The Closed division is intended to compare hardware platforms or software frameworks “apples-to-apples” and requires using the same model as the reference implementation.

The Open division is intended to foster innovation and allows using a different model or retraining.



Trade off accuracy vs latency/throughput vs power consumption vs costs depending on production requirements and constraints

Déjà vu: challenging to run, reproduce, compare and reuse MLPerf inference benchmarks

Every hardware/software vendor uses its own benchmark implementation, harness, setup, container and README

MLPerf benchmarks (vision, language, speech, recommendation …)

NVIDIA ReadMe	**Qualcomm ReadMe**	**Intel ReadMe**	…
Nvidia Container	**Qualcomm Container**	**Intel Container**	…
Fix all deps	Fix all deps	Fix all deps	
Install extra deps	Install extra deps	Install extra deps	
Download model	Download model	Download model	
Connect or download dataset	Connect or download dataset	Connect or download dataset	
Preprocess dataset	Preprocess dataset	Preprocess dataset	
Prepare run-time system	**Prepare run-time system**	**Prepare run-time system**	
Prepare/compile model	**Prepare/compile model**	**Prepare/compile model**	
Run **NVIDIA implementation** of the MLPerf benchmark	Run **Qualcomm implementation** of the MLPerf benchmark	Run **Intel implementation** of the MLPerf benchmark	
Run LoadGen	Run LoadGen	Run LoadGen	
Prepare MLPerf submission	Prepare MLPerf submission	Prepare MLPerf submission	

Generate huge XLS file with all results: performance, accuracy, power consumption
Examples: mlcommons.org/en/inference-datacenter-30 mlcommons.org/en/inference-edge-30

Compare and reproduce results	Compare and reproduce results	Compare and reproduce results	
Optimize the whole stack	Optimize the whole stack	Optimize the whole stack	…

Déjà vu: challenging to run, reproduce, compare and reuse MLPerf inference benchmarks

Every hardware/software vendor uses its own benchmark implementation, harness, setup, container and README

MLPerf benchmarks (vision, language, speech, recommendation …)

NVIDIA ReadMe	**Qualcomm ReadMe**	**Intel ReadMe**	…
Nvidia Container	**Qualcomm Container**	**Intel Container**	…
Fix all deps	Fix all deps	Fix all deps	
Install extra deps	Install extra deps	Install extra deps	
Download model	Download model	Download model	
Connect or download dataset	Connect or download dataset	Connect or download dataset	
Preprocess dataset	Preprocess dataset	Preprocess dataset	
Prepare run-time system	**Prepare run-time system**	**Prepare run-time system**	
Prepare/compile model	**Prepare/compile model**	**Prepare/compile model**	
Run **NVIDIA implementation** of the MLPerf benchmark	Run **Qualcomm implementation** of the MLPerf benchmark	Run **Intel implementation** of the MLPerf benchmark	
Run LoadGen	Run LoadGen	Run LoadGen	
Prepare MLPerf submission	Prepare MLPerf submission	Prepare MLPerf submission	

Generate huge XLS file with all results: performance, accuracy, power consumption
Examples: mlcommons.org/en/inference-datacenter-30 mlcommons.org/en/inference-edge-30

Compare and reproduce results	Compare and reproduce results	Compare and reproduce results	
Optimize the whole stack	Optimize the whole stack	Optimize the whole stack	…

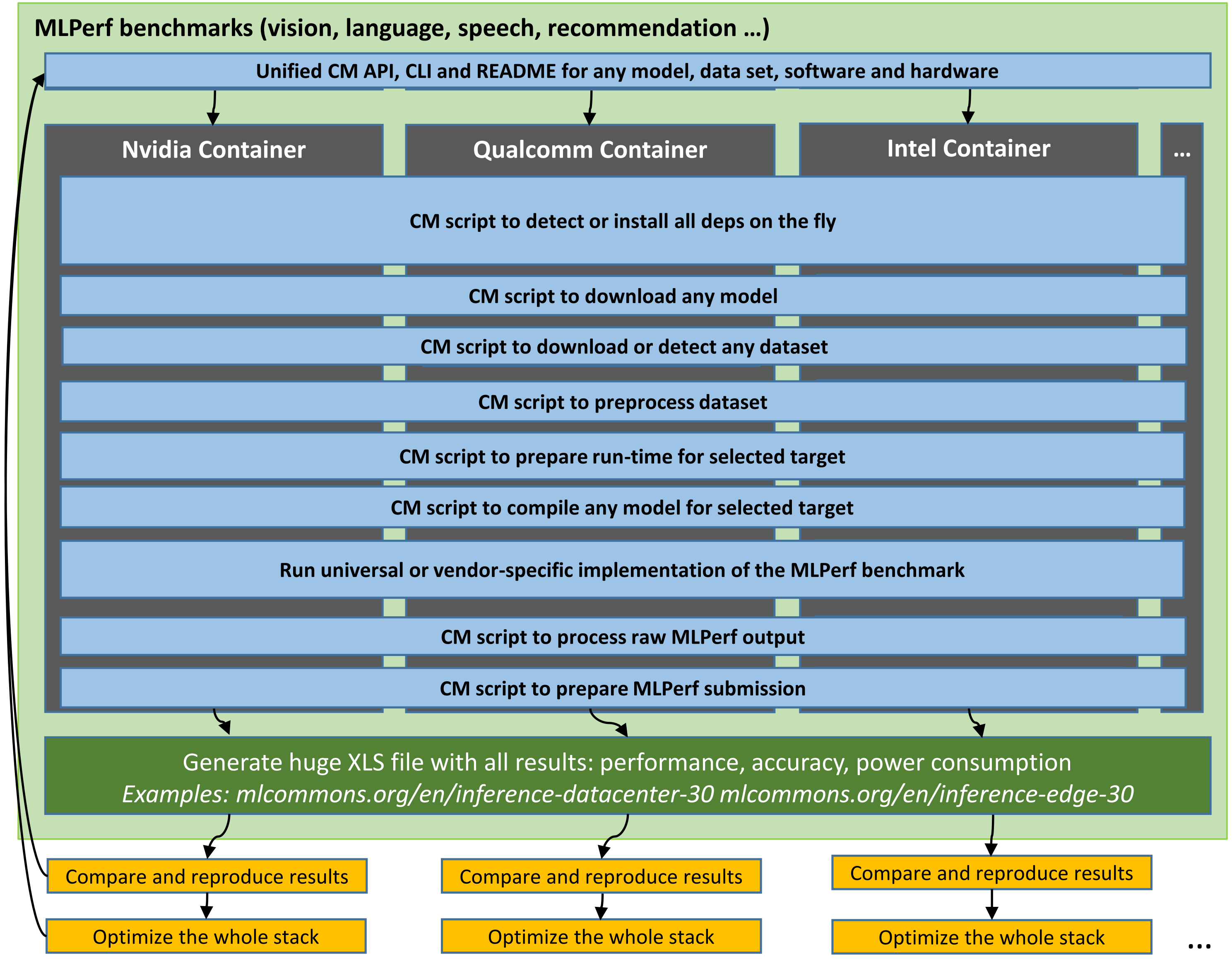
Established the MLCommons Task Force on Automation and Reproducibility.

The goal is to make it easier to run and customize MLPerf benchmarks across continuously changing software and hardware using the CM language:

cknowledge.org/mlcommons-taskforce

Reused, extended and added CM scripts to cover all MLPerf inference steps

Convert MLPerf into a database of portable and reusable script wrappers (CM script) with a unified CLI and common Python API



github.com/mlcommons/ck/tree/master/cm-mlops/script

Output to the CM format; add derived metrics (power efficiency, usage cost …)
Visualize on-prem (private) *or from* ***github.com/mlcommons/cm_inference_results***

Assembled universal MLPerf workflow from CM scripts and generated READMEs and modular CM-MLPerf containers

https://github.com/mlcommons/ck/tree/master/docs/mlperf/inference

Unified CM API, CLI and README for any model, data set, software and hardware

Automatically generated modular CM-MLPerf container

- **Portable CM script to detect or install all deps on the fly**
- **Portable CM script to download any model**
- **Portable CM script to download or detect any dataset**
- **Portable CM script to preprocess dataset**
- **Portable CM script to prepare run-time for selected target**
- **Portable CM script to compile any model for selected target**
- **Run universal or vendor-specific implementation of the MLPerf benchmar**
- **Portable CM script to process raw MLPerf output**
- **Portable CM script to prepare MLPerf submission**

```
python3 -m pip install cmind
cm pull repo mlcommons@ck

cmr "get sys-utils-cm" --quiet
cmr "get python" --version_min=3.8
cmr "get mlperf inference src“
cmr "get mlperf loadgen" --adr.compiler.tags=gcc
cmr "get dataset object-detection open-images original _validation _500“
cmr "get preprocessed dataset object-detection open-images _validation _500 _NCHW“
cmr "get generic-python-lib _onnxruntime" --version_min=1.10.0
cmr "get ml-model object-detection retinanet _onnx“

cmr "app mlperf inference generic _python _retinanet _onnxruntime _cpu" \
        --scenario=Offline --mode=accuracy --test_query_count=10 --rerun
```

Output to the CM format; add derived metrics (power efficiency, usage cost …)
Visualize on-prem (private) *or from* ***github.com/mlcommons/ck_mlperf_results***

Started new project: CM automation and reproducibility assistant based on ChatGPT

chatgpt · Streamlit

access.cknowledge.org/assistant/

Automation and Reproducibility Assistant

What would you like to automate using CM language:

How to run Nvidia implementation of the MLPerf BERT benchmark using CM automation language

ChatGPT suggestion (took 12.16 sec):

To run Nvidia implementation of the MLPerf BERT benchmark using CM automation language, you can use the following Collective Mind (CM) commands:

1. To do a test run to detect and record the system performance:

```
cmr "generate-run-cmds inference _find-performance _all-scenarios" \
--model=bert-99 --implementation=nvidia-original --device=cuda --backend=tensorrt
--category=edge --division=open --quiet
```

Note: Use `--division=closed` to run all scenarios for the closed division (compliance tests are skipped for `_find-performance` mode). Also, use `--category=datacenter` to run datacenter scenarios (only for bert-99.9), and `--model=bert-99.9` to run the high-accuracy model (only for datacenter).

2. To do a full accuracy and performance runs for all the scenarios:

```
cmr "generate-run-cmds inference _full-performance _all-scenarios" \
--model=bert-99 --implementation=nvidia-original --device=cuda --backend=tensorrt
```

Retraining ChatGPT
on CM commands

Used CM for MLPerf replicability study at Student Cluster Competition at SC'22

https://github.com/mlcommons/ck/blob/master/docs/tutorials/sc22-scc-mlperf.md

Connected CM MLPerf workflow with W&B live dashboard:
https://wandb.ai/cmind/cm-mlperf-sc22-scc-retinanet-offline/table

10 teams out of 11 managed to run RetinaNet benchmark
with different ONNX runtimes and CPUs/GPUs within 1 hour
using CM language:

```
cmr "run mlperf inference generate-run-cmds _submission _short _dashboard" \
    --adr.python.version_min=3.8 \
    --adr.compiler.tags=gcc \
    --adr.openimages-preprocessed.tags=_500 \
    --submitter="Community" \
    --hw_name=default \
    --lang=python \
    --model=retinanet \
    --backend=onnxruntime \
    --device=cpu \
    --scenario=Offline \
    --test_query_count=10 \
    --clean
```

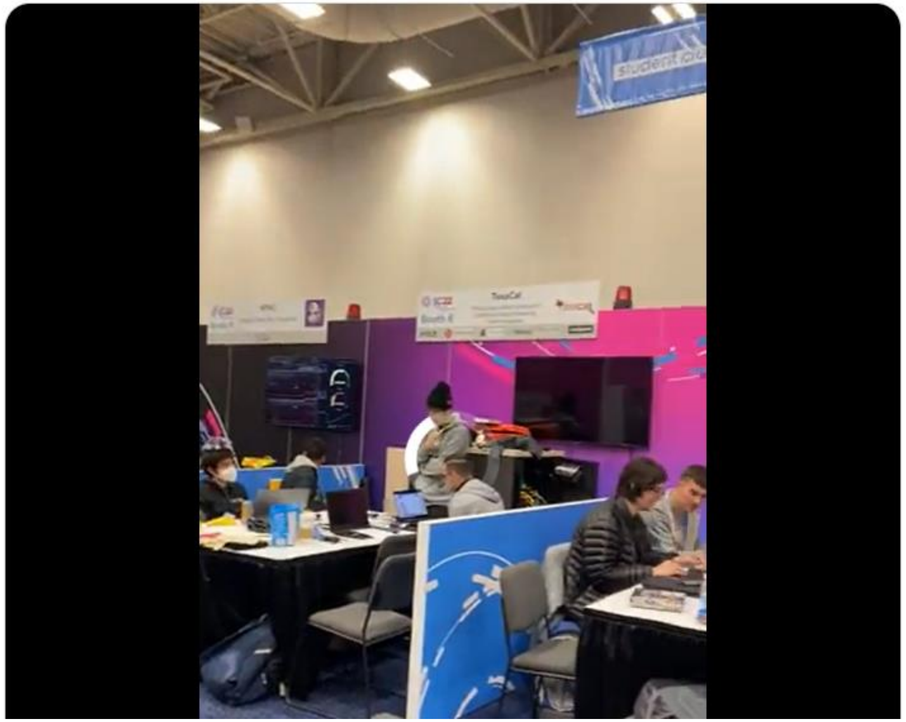
Since it is straightforward to reproduce/replicate MLPerf benchmark
using CM language, students can focus on optimizations!

Dr. Hai Ah Nam
@DrHaiAhNam

So excited at #SC22 to see which team will win the @SCCompSC.

Teams are working hard benchmarking #HPL #HPCG #IO500 #AI before the application workload kick off at the gala tonight.

Thanks @MLPerf @MLCommons @grigori_fursin @OctoML for the #AI benchmark.



Created "Collective Knowledge" platform to aggregate all results from MLPerf benchmarks

MLCommons CK Playground

Universal GUI, API and CLI (**on-prem** or **x.cKnowledge.org**)

Unified CM API, CLI and README for any model, data set, software and hardware

Automatically generated modular CM-MLPerf container

cKnowledge.org/portable-cm-scripts

Portable CM script to detect or install all deps on the fly

Portable CM script to download any model

Portable CM script to download or detect any dataset

Portable CM script to preprocess dataset

Portable CM script to prepare run-time for selected target

Portable CM script to compile any model for selected target

Run universal or vendor-specific implementation of the MLPerf benchmark

Portable CM script to process raw MLPerf output

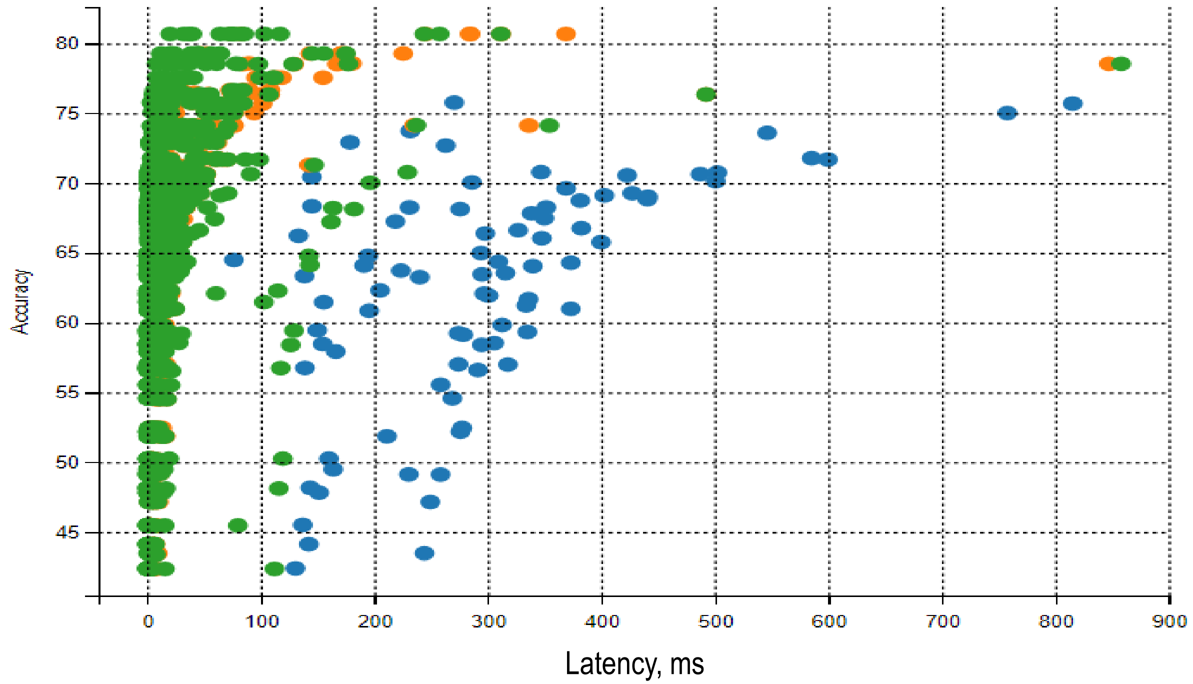
Portable CM script to prepare MLPerf submission

Output to the CM format; add derived metrics (power efficiency, usage cost …)
Visualize on-prem (private) *or from* ***github.com/mlcommons/ck_mlperf_results***

Use unified CK GUI or CM commands to compare and reproduce results

Use unified CK GUI or CM commands to optimize any ML/SW/HW stack

access.cKnowledge.org



- Perform apple-to-apple comparison of diverse ML/SW/HW stacks
- Find Pareto-optimal SW/HW stacks for ML apps based on user requirements and constraints
- Make it easier for the community to reproduce and replicate results automated by the CM language
- Connect academia and industry to optimize performance or test new models and optimization techniques from research papers

Used CM language to let the community participate in the MLPerf inference benchmark submission

Thanks to **Michael Goin, Pablo Gonzalez Mesa, Himanshu Dutta, Aditya Kumar Shaw, Sachin Mudaliyar, Thomas Zhu** and other great colleagues, we validated the CM workflow automation language and CK platform to unify, automate and reproduce MLPerf inference submissions across

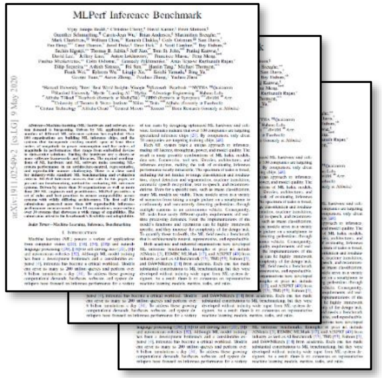
- Diverse CPUs, GPUs and DSPs with PyTorch, ONNX, QAIC, TF/TFLite, TVM and TensorRT
- Hardware from Nvidia (including 4090 workstation and Jetson AGX Orin edge device), Qualcomm, AMD, Intel and Apple
- Deep Sparse optimization from Neural Magic and models from the Hugging Face Zoo
- Cloud submissions on AWS and GCP
- 1st end-to-end student submission on Apple Metal

Reports:

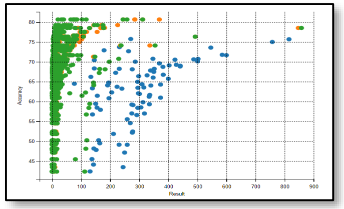
cKnowledge.org/mlperf-inf-v3.0-forbes
cKnowledge.org/mlperf-inf-v3.0-report

Conculsions: $E = mc^2$ - we need to get back to basics and revisit how we share knowledge and experience

Papers, READMEs
Artifact Appendices


MLPerf Inference Benchmark

to obtain results



Scientific and research papers describe some ideas, how they were validated,
and (hopefully) how you can validate them and build upon them using plain English:

1000 papers will have 1000 dialects to describe the same "action" to design experiments:
use Open Images dataset, get Open Images, install open-images, download raw open images from Zenodo

In the end, these actions are converted into OS commands and scripts in 1000 different ways.

OS commands and OS/python scripts

Input file | Input environment variables

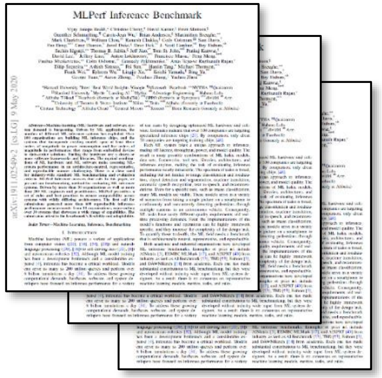
customize.py

run.sh | run.bat

Output file | New environment variables

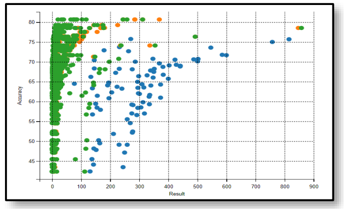
Conculsions: $E = mc^2$ - we need to get back to basics and revisit how we share knowledge and experience

Papers, READMEs
Artifact Appendices


MLPerf Inference Benchmark

Collective Mind automation and reproducibility language

to obtain results



Scientific and research papers describe some ideas, how they were validated, and (hopefully) how you can validate them and build upon them using plain English:

1000 papers will have 1000 dialects to describe the same "action" to design experiments:
use Open Images dataset, get Open Images, install open-images, download raw open images from Zenodo

We provided a simple language to convert all those tags into 1 reusable and tech. agnostic "automation action" that can work anywhere in a deterministic way using low-level OS commands and scripts:
cm run script "get open-images dataset"

or with tech. variations and env variables:
cmr "get open-images dataset _full _NCHW" --env.TYPE=int8

Our language gradually converts all artifacts into a database of interconnected components based on FAIR principles.

When reproducibility/replicability/portability issues occur, the community works together to fix reusable automation actions that automatically improves all other experiments from the community – the foundation of collaborative research and open-science.

Simple YAML or JSON file with human-readable tags "get,use,download,openimages,open-images …"

OS commands and OS/python scripts

Input file | Input environment variables

customize.py

run.sh | run.bat

Output file | New environment variables

Conclusions: we bootstrapped this project to help the community

- My cynical view from 25 years working with academia and industry: making research reproducible, replicable and reusable is considered as a huge pain and generally avoided unless it comes with 0 cost.

- My goal behind Collective Mind language (CM) and Collective Knowledge Playground (CK) is to help the community automate their tedious and time consuming tasks, make their research more deterministic and reproducible, and reuse artifacts and automations with 0 effort: https://github.com/mlcommons/ck , https://access.cKnowledge.org

- There should be no need to prepare code, artifacts, containers, Jupter notebooks and Artifact Appendices and READMEs only for artifact evaluation and external validation – it should come automatically and with 0 cost.

- My cTuning foundation and cKnowledge Ltd are funding these open-source developments to benefit everyone – we implemented the first set of portable, reusable and technology agnostic scripts and artifacts from research papers and projects and connected them with MLPerf LoadGen tool for computational and performance reproducibility:
https://github.com/mlcommons/ck/tree/master/cm-mlops/script

- CM helps our colleagues focus on research and innovation instead of reinventing the wheel and implementing numerous ad-hoc workflows and measurement tools that often die when research project is finished.

- Furthermore, CM language made it easier to transfer research to production by automatically adapting research projects to diverse and continuously changing software, hardware, models and data from the real world!

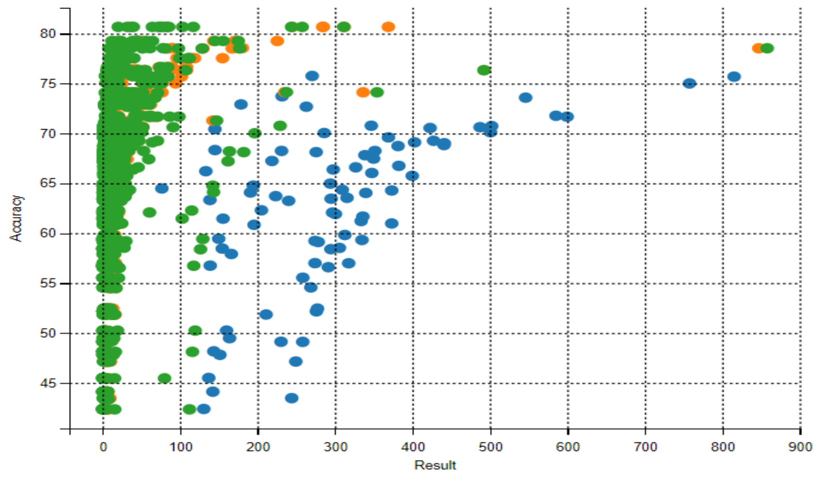
- *I converted all my R&D into CM and finally solved some of the problems that haunt me since 1996. I hope it can help you too!*

The next steps: continue reducing the friction for reproducible research and tech. transfer

1) Continue working with MLCommons, academia and industry to use and extend CM language for their R&D projects

2) Organize reproducibility and replicability challenges for MLPerf benchmarks and research papers:
   https://github.com/mlcommons/ck/tree/master/cm-mlops/challenge

   - Automate using CM language, add missing CM scripts.
   - Collaboratively analyze results and explain research techniques.
   - **Add more tasks, models and data sets from Hugging Face and paperswithcode.com**
   - Submit new MLPerf results automated by the CM language.
   - Improve benchmarking methodology and fix issues.
   - Publish reproducibility reports with the CM language **(maybe special track at ACM REP?).**



3) Use CM language to generate Pareto-optimal AI/ML applications and systems based on MLPerf results and user requirements and constraints (trading off accuracy, performance, power, costs).

4) Continue reproducing research papers and automating them using the CM language (IPOL journal, Artifact Evaluation at MICRO'23).

5) Continue improving our automation and reproducibility assistant to help researchers and engineers reuse CM scripts and convert their projects into CM format with 0 cost: https://access.cKnowledge.org/assistant

Would you like to help with this community project or provide feedback?

Collective Mind language helps me follow my passion:
helping the community focus on innovation, reproduce research projects, understand them
and bring them to the real world in the most efficient and automated way!

This project would not have happened without the help and feedback from the community!

Huge thanks to all the contributors: github.com/mlcommons/ck/blob/master/CONTRIBUTING.md

Thanks to Arjun Suresh for joining this crazy effort!

Thanks to Chloé Tessier for an illustration of a worried scientist!

Thanks to Alissa and Victor Fursin for keeping me sane and make sure that CM language is simple enough even for kids!

Join our project via Discord server: discord.gg/JjWNWXKxwT

Follow and support our project on GitHub: github.com/mlcommons/ck

Feel free to teuse/extend our portable scripts and automations to accelerate your research and experimentation:

github.com/mlcommons/ck/tree/master/cm-mlops/script
github.com/mlcommons/ck/tree/master/cm-mlops/automation