

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

Grigori Fursin

cTuning foundation and cKnowledge SAS

github.com/ctuning/ck [cKnowledge.io](https://cknowledge.io)

Accepted for Philosophical Transactions of the Royal Society A

(Submitted: 12 June 2020)

Abstract

This article provides the motivation and overview of the Collective Knowledge framework (CK or cKnowledge). The CK concept is to decompose research projects into reusable components that encapsulate research artifacts and provide unified application programming interfaces (APIs), command-line interfaces (CLIs), meta descriptions and common automation actions for related artifacts. The CK framework is used to organize and manage research projects as a database of such components.

Inspired by the USB "plug and play" approach for hardware, CK also helps to assemble portable workflows that can automatically plug in compatible components from different users and vendors (models, datasets, frameworks, compilers, tools). Such workflows can build and run algorithms on different platforms and environments in a unified way using the customizable CK program pipeline with software detection plugins and the automatic installation of missing packages.

This article presents a number of industrial projects in which the modular CK approach was successfully validated in order to automate benchmarking, auto-tuning and co-design of efficient software and hardware for machine learning (ML) and artificial intelligence (AI) in terms of speed, accuracy, energy, size and various costs. The CK framework also helped to automate the artifact evaluation process at several computer science conferences as well as to make it easier to reproduce, compare and reuse research techniques from published papers, deploy them in production, and automatically adapt them to continuously changing datasets, models and systems.

The long-term goal is to accelerate innovation by connecting researchers and practitioners to share and reuse all their knowledge, best practices, artifacts, workflows and experimental results in a common, portable and reproducible format at [cKnowledge.io](https://cknowledge.io).

Keywords: *research automation, reproducibility, reusability, portability, DevOps, AIOps, MLOps, FAIR principles, knowledge management, best practices, collaboration, optimization, portable workflow, microservices, adaptive container, machine learning, artificial intelligence, edge devices, MLPerf, API, crowd-benchmarking, crowd-tuning, auto-tuning, software/hardware co-design, efficient systems, Pareto efficiency, optimization repository*

1 Motivation

Ten years ago I developed the cTuning.org platform and released all my research code and data into the public domain in order to crowdsource the training of our machine learning-based compiler (MILEPOST GCC) [41]. My intention was to accelerate the very time-consuming auto-tuning process and help our compiler to learn the most efficient optimizations across real programs, datasets, platforms and environments

provided by volunteers.

We had a positive response from the community and, in just a few days, I collected as many optimization results as I had during the entire MILEPOST project. However, the initial excitement quickly faded when I struggled to reproduce most of the performance numbers and ML model predictions because even a tiny change in software, hardware, environment or the run-time state of the system could influence performance while I did not have a mechanism to detect such changes [51, 43].

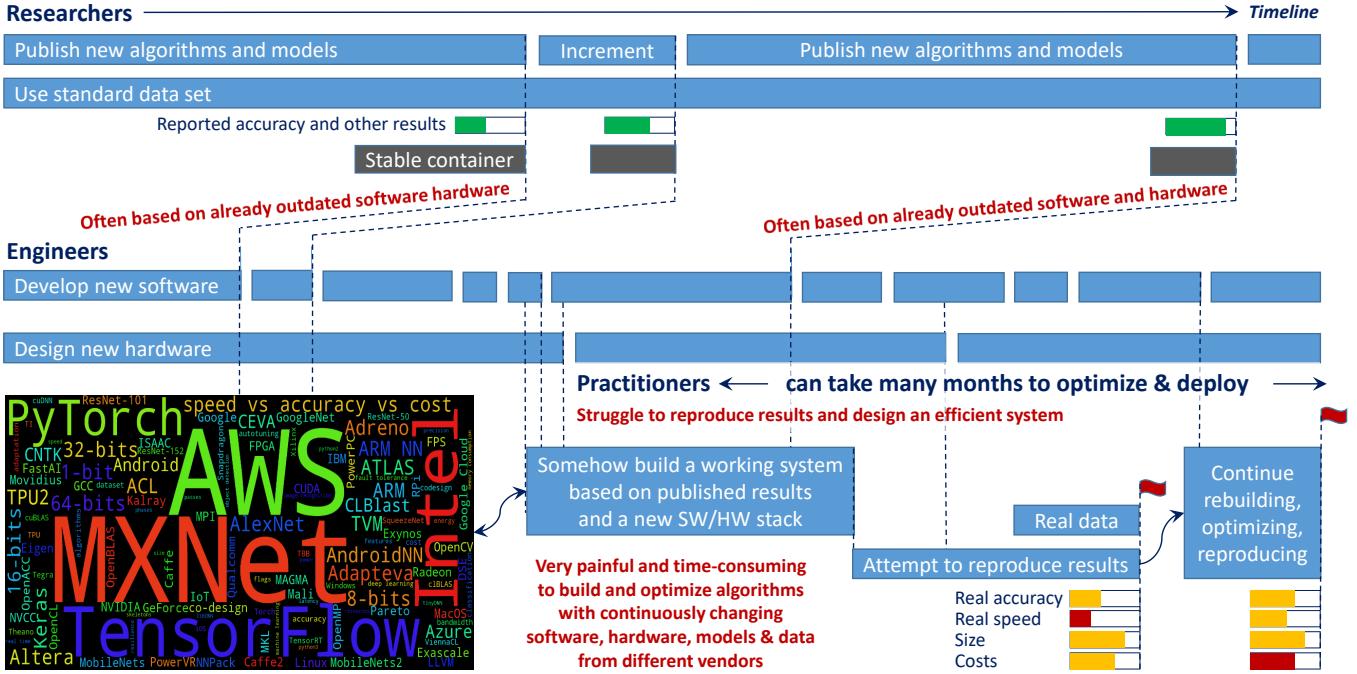


Figure 1: Reproducing research papers and adopting novel techniques in production is a tedious, repetitive and time-consuming process because of continuously changing software, hardware, models and datasets, and a lack of common formats and APIs for shared artifacts (code, data, models, experimental results, scripts and so on).

Worse still, I could not compare these empirical results with other published techniques because they rarely included the full experiment specification with the shared research code and all the necessary artifacts needed to be able to reproduce results. Furthermore, it was always very difficult to add new tools, benchmarks and datasets to any research code because it required numerous changes in different ad-hoc scripts, repetitive recompilation of the whole project when new software was released, complex updates of database tables with results and so on [42].

These problems motivated me to establish the non-profit cTuning foundation and continue working with the community on a common methodology and open-source tools to enable collaborative, reproducible, reusable and trustable R&D. We helped to initiate reproducibility initiatives and support artifact evaluation at several computer science conferences in collaboration with the ACM [5, 40]. We also promoted sharing of code, artifacts and results in a unified way along with research papers [4].

This community service gave me a unique chance to participate in reproducibility studies of more than 100 research papers at the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), International Symposium on Code Generation and Optimization (CGO), ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP), Supercomputing, International Conference on Machine Learning and

Systems (MLSys) and other computer science conferences over the past five years [27]. I also started deploying some of these techniques in production in collaboration with my industrial partners in order to better understand all the problems associated with building trustable, reproducible and production-ready computational systems.

This practical experience confirmed my previous findings: while sharing ad-hoc research code, artifacts, trained models, and Docker images along with research papers is a great step forward, it is only the tip of the reproducibility iceberg [42]. The major challenge afterwards is threefold. It is necessary to figure out how to: use research techniques outside original containers with other data, code and models; run them in a reliable and efficient way across rapidly evolving software, heterogeneous hardware and legacy platforms with continuously changing interfaces and data formats; and balance multiple characteristics including speed, latency, accuracy, memory size, power consumption, reliability and costs (Figure 1).

2 Collective Knowledge concept

When helping to organize the artifact evaluation process at systems and machine learning conferences, I decided to introduce an Artifact Appendix and a reproducibility checklist [4, 27]. My goal was to help researchers to describe how to configure, build, run, validate and compare research techniques in a unified way across different conferences and journals. It also led me to

notice that most research projects use some ad-hoc scripts, often with hardwired paths, to perform the same repetitive tasks including downloading models and datasets, detecting platform properties, installing software dependencies, building research code, running experiments, validating outputs, reproducing results, plotting graphs and generating papers [42].

This experience motivated me to search for a solution for automating such common tasks and making them reusable and customizable across different research projects, platforms and environments. First, I started looking at related tools that were introduced to automate experiments and make research more reproducible:

- Workflow frameworks such as MLFlow [57], Kedro [52], Amazon SageMaker [3], Kubeflow [34], Apache Taverna [56], popper [48], CommonWL [36] and many others help to abstract and automate data science operations. They are very useful for data scientists but do not yet provide a universal mechanism to automatically build and run algorithms across different platforms, environments, libraries, tools, models and datasets. Researchers and engineers often have to implement this functionality for each new project from scratch, which can become very complicated particularly when targeting new hardware, embedded devices, TinyML and IoT.
- Machine learning benchmarking initiatives such as MLPPerf [53], MLModelScope [39] and Deep500 [37] attempt to standardise machine learning model benchmarking and make it more reproducible. However, production deployment, integration with complex systems and adaptation to continuously changing user environments, platforms, tools, and data are currently out of their scope.
- Package managers such as Spack [46] and EasyBuild [47] are very useful for rebuilding and fixing the whole software environment. However, the integration with workflow frameworks and automatic adaptation to existing environments, native cross-compilation particularly for embedded devices, and support for non-software packages (models, datasets, scripts) are still in progress.
- Container technology such as Docker [50] is very useful for preparing and sharing stable software releases. However, it hides the software chaos rather than solving it, lacks common APIs for research projects, requires enormous amount of space, has very poor support for embedded devices, and does not yet help integrate models with existing projects, legacy systems and user data.
- PapersWithCode platform [24] helps to find relevant research code for machine learning papers and keeps track of state-of-the-art ML research using

public scoreboards with non-validated experimental results from papers. However, my experience reproducing research papers suggests that sharing ad-hoc research code is not enough to make research techniques reproducible, customizable, portable and trustable [42].

While testing all these useful tools and analyzing the Jupyter notebooks, Docker images and GitHub repositories shared alongside research papers, I started thinking that it would be possible to reorganize them as some sort of database of reusable components with a common API, command line, web interface and meta description. We could then reuse artifacts and some common automation actions across different projects while applying DevOps principles to research projects.

Furthermore, we could also gradually abstract and interconnect all existing tools rather than rewriting or substituting them. This, in turn, helped to create "plug and play" workflows that could automatically connect compatible components from different users and vendors (models, datasets, frameworks, compilers, tools, platforms) while minimising manual interventions and providing a common interface for all shared research projects.

I called my project Collective Knowledge (CK) because my goal was to connect researchers and practitioners in order to share their knowledge, best practices and research results in the form of reusable components, portable workflows, and automation actions with common APIs and meta descriptions.

3 CK framework and an open CK format

I have developed the CK framework as a small Python library with minimal dependencies to be very portable while offering the possibility of being re-implemented in other languages such as C, C++, Java and Go. The CK framework has a unified CLI, a Python API and a JSON-based web service to manage *CK repositories* and add, find, update, delete, rename and move *CK components* (sometimes called *CK entries* or *CK data*) [9].

CK repositories are human-readable databases of reusable CK components that can be created in any local directory and inside containers, pulled from GitHub and similar services, and shared as standard archive files [7]. *CK components* simply wrap (encapsulate) user artifacts and provide an extensible JSON meta description with *common automation actions* [6] for related artifacts.

Automation actions are implemented using *CK modules*, which are Python modules with functions exposed in a unified way via CK API and CLI and which use dictionaries for input and output (extensible and unified CK input/output (I/O)). The use of dictionaries

1) GitHub repo or archive file	2) User home directory	3) Jupyter/colab notebook	4) Docker image
<pre>.ckr.json /cm/alias-a-dataset /cm/alias-a-module /cm/alias-a-program /cm/alias-a-paper /dataset/.cm/alias-a-images /dataset/images/1.png /cm/meta.json /cm/info.json /program/.cm/alias-a-detect-edges /program/detect-edges/program.cpp Makefile run.sh check-output.sh autotune.sh /cm/meta.json /cm/info.json /paper/.cm/alias-a-report /paper/report/pldi.tex /cm/meta.json /cm/info.json /module/.cm/alias-a-program /module/program/module.py /cm/meta.json /cm/info.json</pre> <p>Collective Knowledge COMPATIBLE</p>	<pre>\$HOME/project/2000-forgot-everything/ .ckr.json /cm/alias-a-dataset /cm/alias-a-program /cm/alias-a-experiment /cm/alias-a-paper /dataset/images-2000/2.png /cm/meta.json /program/crazy-algorithm/source.cpp build.bat /cm/meta.json /experiment/autotuning/many.logs /lots-of-stats.sql /cm/meta.json /paper/asplos/source.tex /cm/meta.json</pre> <p>Collective Knowledge COMPATIBLE</p> <pre>\$ ck pull repo:ck-crowdtuning \$ ck add repo:2000-forgot-everything \$ ck ls dataset:image* dataset:images dataset:images-2000 \$ ck ls program program:detect-edges program:object-classification</pre>	<pre>import matplotlib.pyplot as plt import pandas import numpy Collective Knowledge COMPATIBLE ... import ck.kernel as ck # We can now access all our software projects as a database r=ck.access({'action':'search', 'module_uaa':'dataset', 'add_mta':'yes'}) if r['return']>0: ck.err(r) list_of_all_ck_entries=r['list'] for ck_entry in list_of_all_ck_entries: # CK will find all dataset entries in all CK-compatible projects, # even old ones – you don't need to remember # the project structure. Furthermore, you can continue # reusing project even if students or engineers leave! image=ck_entry['path']+ck_entry['meta'][‘image_filename’] ... # Call reusable CK automation action to extract features features=ck.access({'action':'get_features', 'module_uaa':'dataset', 'image':image}) if features['return']>0: ck.err(features)</pre>	<p>Install stable OS and packages Set environment</p> <p>Use familiar CK API/CLI to run experiments inside or outside your VM</p> <p>Move data outside VM in the CK format to continue processing it via CK!</p> <p>Collective Knowledge COMPATIBLE</p> <pre>\$ ck run program:detect-edges Searching for datasets ... Select dataset: 1) images 2) images-2000 \$ ck benchmark program:detect-edges --record --ecord_uaa=autotuning ... \$ ck reproduce experiment:autotuning ...</pre>

Figure 2: Organizing software projects as a human-readable database of reusable components that wrap artifacts and provide unified APIs, CLI, meta descriptions and common automation actions for related artifacts. The CK APIs and control files are highlighted in blue.

makes it easier to support continuous integration tools and web services and to extend the functionality while keeping backward compatibility. The unified I/O also makes it possible to reuse such actions across projects and to chain them together into unified CK pipelines and workflows.

As I wanted CK to be non-intrusive and technology neutral, I decided to use a simple 2-level directory structure to wrap user artifacts into CK components as shown in Figure 2. The root directory of the CK repository contains the `.ckr.json` file to describe the repository and specify dependencies on other CK repositories to explicitly reuse their components and automation actions.

CK uses `.cm` directories (Collective Meta) similar to `.git` to store meta information of all components as well as Unique IDs to be able to find components even if their

user-friendly names have changed over time (*CK alias*). CK modules are always stored in `module / <CK module name>` directories in the CK repository. CK components are stored in `<CK module name>/ <CK data name>` directories. Each *CK component* has a `.cm` directory with the `meta.json` file describing a given artifact and `info.json` file to keep the provenance of a given artifact including copyrights, licenses, creation date, names of all contributors, and so on.

The CK framework has an internal *default CK repository* with *stable CK modules* and the most commonly used automation actions across many research projects. When the CK framework is used for the first time, it also creates a *local CK repository* in the user space to be used as a scratch pad.

After discussing the CK CLI with colleagues, I decided to implement it in a similar way to natural language to make it easier for users to remember the commands:

```
ck <action> <CK module name> (flags) (@input.json or @input.yaml)
ck <action> <CK module name>:<CK data name> (flags)
ck <action> <CK repository name>:<CK module name>:<CK data name>
```

The next example demonstrates how to compile and run the shared automotive benchmark on any platform, and then create a copy of the CK *program component*:

```

pip install ck

ck pull repo —url=https://github.com/ctuning/ck-crowdtuning

ck search dataset —tags=jpeg

ck search program:cbench-automotive-*

ck find program:cbench-automotive-susan

ck load program:cbench-automotive-susan

ck help program

ck compile program:cbench-automotive-susan —speed
ck run program:cbench-automotive-susan —env.OMP_NUM_THREADS=4

ck run program —help

ck cp program:cbench-automotive-susan local:program:new-program-workflow
ck find program:new-program-workflow

ck benchmark program:new-program-workflow —record —record_uoa=my-test
ck replay experiment:my-test

```

The CK program module describes dependencies on simple tags and version ranges that the community has software detection plugins and meta packages using to agree on:

```

{
  "compiler": {
    "name": "C++ compiler",
    "sort": 10,
    "tags": "compiler, lang-cpp"
  },
  "library": {
    "name": "TensorFlow C++ API",
    "sort": 20,
    "version_from": [1,13,1],
    "version_to": [2,0,0],
    "no_tags": "tensorflow-lite",
    "tags": "lib, tensorflow, vstatic"
  }
}

```

I also implemented a simple *access* function in the CK simple and unified way: Python API to access all the CK functionality in a very

```

import ck.kernel as ck

# Equivalent of "ck compile program:cbench-automotive-susan —speed"
r=ck.access({'action':'compile',
            'module_uoa':'program', 'data_uoa':'cbench-automotive-susan',
            'speed':'yes'})
if r['return']>0: return r # unified error handling
print (r)

# Equivalent of "ck run program:cbench-automotive-susan —env.OMP_NUM_THREADS=4
r=ck.access({'action':'run',
            'module_uoa':'program', 'data_uoa':'cbench-automotive-susan',
            'env':{'OMP_NUM_THREADS':4}})
if r['return']>0: return r # unified error handling
print (r)

```

Such an approach allowed me to connect colleagues, students, researchers and engineers from different workgroups to implement, share and reuse automation actions and CK components rather than reimplementing them from scratch. Furthermore, the Collective Knowledge concept supported the so-called FAIR principles to make data findable, accessible, interoperable and reusable [55] while also extending it to code and other research artifacts.

4 CK components and workflows to automate machine learning and systems research

One of the biggest challenges I have faced throughout my research career automating the co-design process of efficient and self-optimizing computing systems has been how to deal with rapidly evolving hardware, models, datasets, compilers and research techniques. It is for this reason that my first use case for the CK framework was to work with colleagues to collaboratively solve these problems and thus enable trustable, reliable and efficient computational systems that can be easily deployed and used in the real world.

We started using CK as a flexible playground to decompose complex computational systems into reusable, customizable and non-virtualized CK components while agreeing on their APIs and meta descriptions. As the first step, I implemented basic actions that could automate the most common R&D tasks that I encountered during artifact evaluation and in my own research on systems and machine learning [42]. I then shared the automation actions to analyze platforms and user environments in a unified way, detect already installed code, data and ML models (*CK software detection plugins* [32]), and automatically download, install and cross-compile missing packages (*CK meta packages* [31]). At the same time, I provided initial support for different compilers, operating systems (Linux, Windows, MacOS, Android) and hardware from different vendors including Intel, Nvidia, Arm, Xilinx, AMD, Qualcomm, Apple, Samsung and Google.

Such an approach allowed my collaborators [12] to create, share and reuse different CK components with unified APIs to detect, install and use different AI and ML frameworks, libraries, tools, compilers, models and datasets. The unified automation actions, APIs and JSON meta descriptions of all CK components also helped us to connect them into platform-agnostic, portable and customizable program pipelines (workflows) with a common interface across all research projects while applying the DevOps methodology.

Such "plug&play" workflows [45] can automatically adapt to evolving environments, models, datasets and non-virtualized platforms by automatically detecting the

properties of a target platform, finding all required dependencies and artifacts (code, data and models) on a user platform with the help of CK software detection plugins [32], installing missing dependencies using portable CK meta packages [31], building and running code, and unifying and testing outputs [25]. Moreover, rather than substituting or competing with existing tools, the CK approach helped to abstract and interconnect them in a relatively simple and non-intrusive way.

CK also helps to protect user workflows whenever external files or packages are broken, disappear or move to another URL because it is possible to fix such issues in a shared CK meta package without changing existing workflows. For example, our users have already taken advantage of this functionality when the Eigen library moved from BitBucket to GitLab and when the old ImageNet dataset was no longer supported but could still be downloaded via BitTorrent and other peer-to-peer services.

Modular CK workflows can help to keep track of the information flow within such workflows, gradually expose configuration and optimization parameters as vectors via dictionary-based I/O, and combine public and private code and data. They also help to monitor, model and auto-tune system behaviour, retarget research code and machine learning models to different platforms from data centers to edge devices, integrate them with legacy systems, and reproduce results.

Furthermore, we can use CK workflows inside standard containers such as Docker while providing a unified CK API to customize, rebuild and adapt them to any platform (*Adaptive CK container*) [2] thus making research techniques truly portable, reproducible and reusable. I envision that such adaptive CK containers and portable workflows can complement existing marketplaces to deliver portable, customizable, trustable and efficient AI and ML solutions that can be continuously optimized across diverse models, datasets, frameworks, libraries, compilers and run-time systems.

In spite of some initial doubts, my collaborative CK approach has worked well to decompose complex research projects and computational systems into reusable components while agreeing on common APIs and meta descriptions: the CK functionality evolved from just a few core CK modules and abstractions to hundreds of CK modules [30] and thousands of reusable CK components and workflows [33] to automate the most repetitive research tasks particularly for AI, ML and systems R&D as shown in Figure 3.

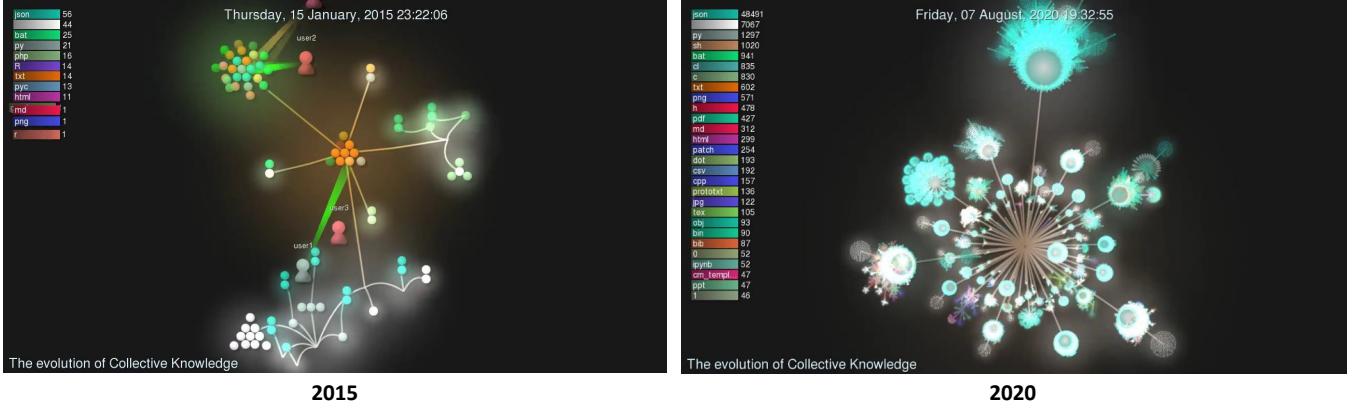


Figure 3: Collective Knowledge framework provided a flexible playground for researchers and practitioners to decompose complex computational systems into reusable components while agreeing on APIs and meta descriptions. Over the past 5 years, the Collective Knowledge project has grown to thousands of reusable CK components and automation actions.

5 CK use cases

5.1 Unifying benchmarking, auto-tuning and machine learning

The MILEPOST and cTuning projects were like the Apollo mission: on the one hand, we managed to demonstrate that it was indeed possible to crowdsource auto-tuning and machine learning across multiple users to automatically co-design efficient software and hardware [41, 44]. On the other hand, we exposed so many issues when using machine learning for system optimization in the real world that I had to stop this research and have since focused on solving a large number of related engineering problems.

For this reason, my first step was to test the CK concept by converting all artifacts and automation actions from all my past research projects related to self-optimizing computer systems into reusable CK components and workflows. I shared them with the community in the CK-compatible Git repositories [7] and started reproducing experiments from my own or related research projects [20].

I then implemented a customizable and portable program pipeline as a CK module to unify benchmarking and auto-tuning while supporting all research techniques and experiments from my PhD research and the MILEPOST project [10]. Such a pipeline could perform compiler auto-tuning and software/hardware co-design combined with machine learning in a unified way across different programs, datasets, frameworks, compilers, ML models and platforms as shown in Figure 4.

The CK program pipeline helps users to gradually expose different design choices and optimization parameters from all CK components (models, frameworks, compilers, run-time systems, hardware) via unified CK APIs and meta descriptions and thus enable the whole ML and system auto-tuning. It also

helps users to keep track of all information passed between components in complex computational systems to ensure the reproducibility of results while finding the most efficient configuration on a Pareto frontier in terms of speed, accuracy, energy and other characteristics also exposed via unified CK APIs. More importantly, it can be now reused and extended in other real-world projects [12].

5.2 Bridging the growing gap between education, research and practice

During the MILEPOST project I noticed how difficult it is to start using research techniques in the real world. New software, hardware, datasets and models are usually available at the end of such research projects making it very challenging, time-consuming and costly to make research software work with the latest systems or legacy technology.

That prompted us to organize a proof-of-concept project with the Raspberry Pi foundation to check if it was possible to use portable CK workflows and components to enable sustainable research software that can automatically adapt to rapidly evolving systems. We also wanted to provide an open-source tool for students and researchers to help them share their code, data and models as reusable, portable and customizable workflows and artifacts - something now known as FAIR principles [55].

For this purpose, we decided to reuse the CK program workflow to demonstrate that it was possible to crowdsource compiler auto-tuning across any Raspberry Pi device, with any environment and any version of any compiler (GCC or LLVM) to automatically improve the performance and code size of the most popular RPi applications. CK helped to automate experiments, collect performance numbers on live CK scoreboards and automatically plug in CK components with various

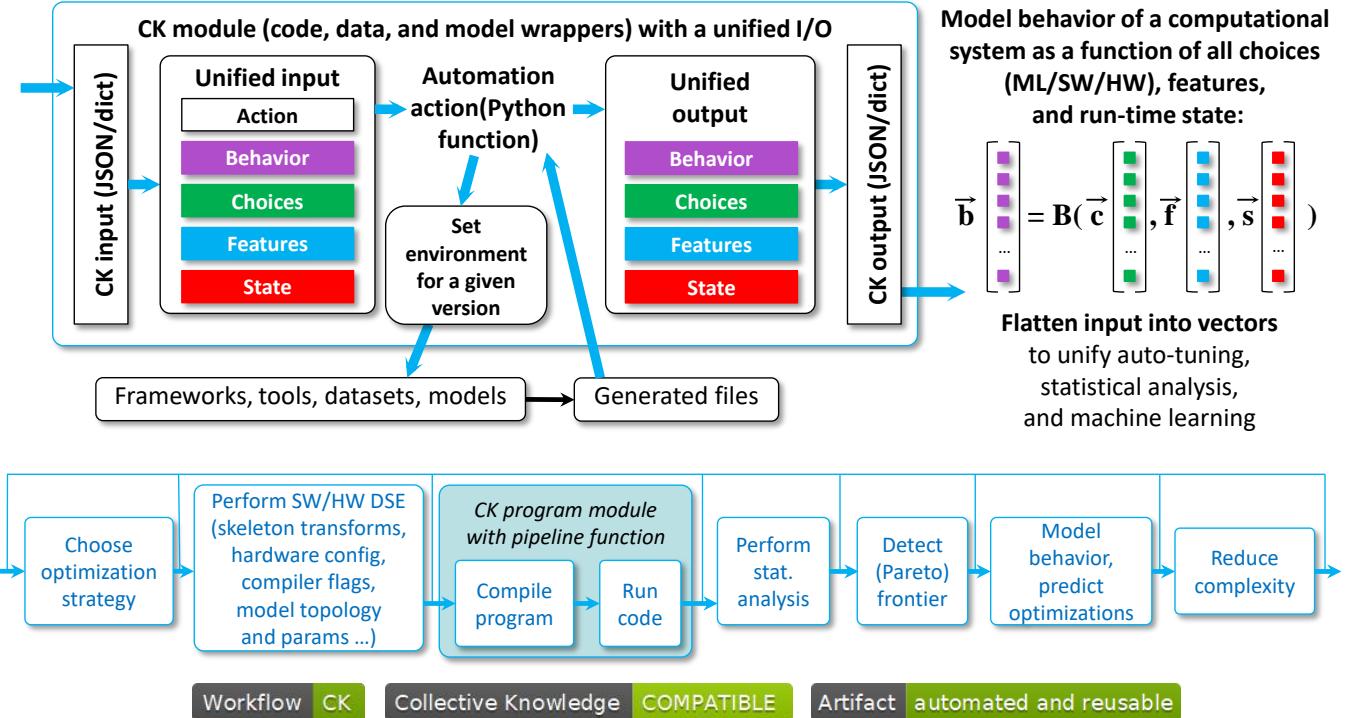


Figure 4: Portable, customizable and reusable program pipeline (workflow) assembled from the CK components to unify benchmarking, auto-tuning and machine learning. It is possible to gradually expose design search space, optimizations, features and the run-time state from all CK components to make it easier to unify performance modelling and auto-tuning.

machine learning and predictive analytics techniques including decision trees, nearest neighbour classifiers, support vector machines (SVM) and deep learning, to automatically learn the most efficient optimizations [45].

With this project, we demonstrated that it was possible to reuse portable CK workflows and let users participate in collaborative auto-tuning (crowd-tuning) on new systems while sharing best optimizations and unexpected behaviour on public CK scoreboards, even after the project.

5.3 Co-designing efficient software and hardware for AI, ML and other emerging workloads

While helping companies to assemble efficient software and hardware for image classification, object detection and other emerging AI and ML workloads, I noticed that it can easily take several months to build an efficient and reliable system before moving it to production.

This process is so long and tedious because one has to navigate a multitude of design decisions when selecting components from different vendors for different applications (image classification, object detection, natural language processing (NLP), speech recognition and many others) while trading off speed, latency, accuracy, energy and other costs: what network architecture to deploy and how to customize it (ResNet,

MobileNet, GoogleNet, SqueezeNet, SSD, GNMT); what framework to use (PyTorch vs. MXNet vs. TensorFlow vs. TF Lite vs. Caffe vs. CNTK); what compilers to use (XLA vs. nGraph vs. Glow vs. TVM); what libraries and which optimizations to employ (ArmNN vs. MKL vs. OpenBLAS vs. cuDNN). This is generally a consequence of the target hardware platform (CPU vs. GPU vs. DSP vs. FPGA vs. TPU vs. Edge TPU vs. numerous accelerators). Worse still, this semi-manual process is usually repeated from scratch for each new version of hardware, models, frameworks and datasets.

My modular CK program pipeline shown in Figure 5 helped to automate this process. We extended it slightly to plug in different AI and ML algorithms, datasets, models, frameworks and libraries for different hardware such as CPU, GPU, DSP and TPU and different target platforms from servers to Android devices and IoT [12]. We also customized this ML workflow with the new CK plugins that performed pre- and post-processing of different models and datasets to make them compatible with different frameworks, backends and hardware while unifying benchmarking results such as throughput, latency, mAP (mean Average Precision), recall and other characteristics. We also exposed different design and optimization parameters including model topology, batch sizes, hardware frequency, compiler flags and so on.

Eventually, CK allowed us to automate and

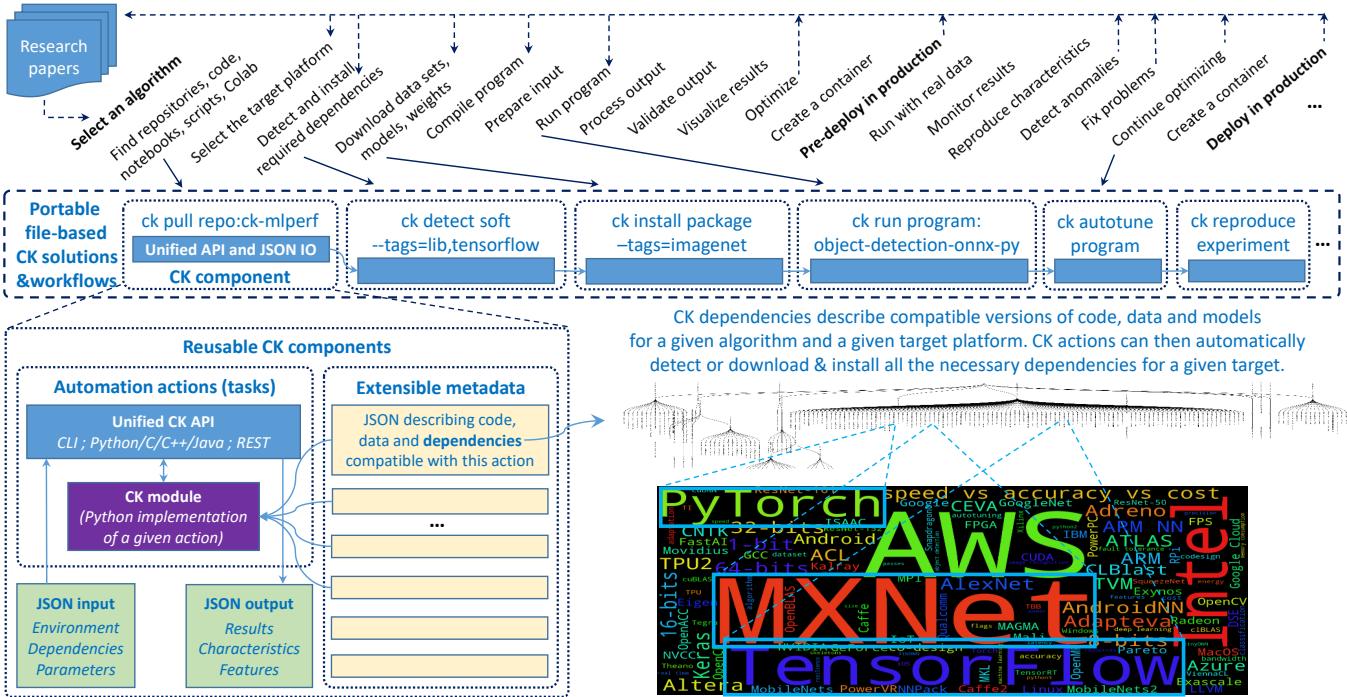


Figure 5: The use of the CK framework to automate benchmarking, optimization and co-design of efficient software and hardware for machine learning and artificial intelligence. The goal is to make it easier to reproduce, reuse, adopt and build upon ML and systems research.

systematise design space exploration (DSE) of AI/ML/software/hardware stacks and distribute it across diverse platforms and environments [8]. This is possible because CK automatically detects all necessary dependencies on any platform, installs and/or rebuilds the prerequisites, runs experiments, and records all results together with the complete experiment configuration (resolved dependencies and their versions, environment variables, optimization parameters and so on) in a unified JSON format inside CK repositories. CK also ensured the reproducibility of results while making it easier to analyze and visualize results locally using Jupyter notebooks and standard toolsets or within workgroups using universal CK dashboards also implemented as CK modules [20].

Note that it is also possible to share the entire experimental setup in the CK format inside Docker containers, thus automating all the DSE steps using the unified CK API instead of trying to figure them out from the ReadMe files. This method enables CK-powered adaptive containers that help users to start using and customizing research techniques across diverse software and hardware from servers to mobile devices in just a few simple steps while sharing experimental results within workgroups or along research papers in the CK format, reproducing and comparing experiments, and even automatically reporting unexpected behaviour such as bugs and mispredictions [35].

Eventually, I managed to substitute my original cTuning framework completely with the modular,

portable, customizable and reproducible experimental framework while addressing most of the engineering and reproducibility issues exposed by the MILEPOST and cTuning projects [41]. It also helped me to return to my original research on lifelong benchmarking, optimization and co-design of efficient software and hardware for emerging workloads, including machine learning and artificial intelligence.

5.4 Automating MLPerf and enabling portable MLOps

The modularity of my portable CK program workflow helped to enable portable MLOps when combined with AI and ML components, FAIR principles and the DevOps methodology. For example, my CK workflows and components were reused and extended by General Motors and dividiti to collaboratively benchmark and optimize deep learning implementations across diverse devices from servers to the edge [11]. They were also used by Amazon to enable scaling of deep learning on AWS using C5 instances with MXNet, TensorFlow and BigDL [49]. Finally, the CK framework made it easier to prepare, submit and reproduce MLPerf inference benchmark results ("fair and useful benchmark for measuring training and inference performance of ML hardware, software, and services") [53, 23]. These results are aggregated in the public optimization repository [20] to help users find and compare different AI/ML/software/hardware stacks in terms of speed,

accuracy, power consumption, costs and other collected metrics.

5.5 Enabling reproducible papers with portable workflows and reusable artifacts

Ever since my very first research project, I wanted to be able to easily find all artifacts (code, data, models) from research papers, reproduce and compare results in just a few clicks, and immediately test research techniques in the real world with different platforms, environments and data. It is for that reason that one of my main goals when designing the CK framework was to use portable CK workflows for these purposes.

I had the opportunity to test my CK approach when co-organizing the first reproducible tournament at the ACM ASPLOS’18 conference (the International Conference on Architectural Support for Programming Languages and Operating Systems). The tournament required participants to co-design Pareto-efficient systems for deep learning in terms of speed, accuracy, energy, costs and other metrics [29]. We wanted to extend existing optimization competitions, tournaments and hackathons including Kaggle [19], ImageNet [18], the Low-Power Image Recognition Challenge (LPIRC) [21], DAWN Bench (an end-to-end deep learning benchmark and competition) [17], and MLPerf [53, 22] with a customizable experimental framework for collaborative and reproducible optimization of Pareto-efficient software and hardware stack for deep learning and other emerging workloads.

This tournament helped to validate my CK approach for reproducible papers. The community submitted five complete implementations (code, data, scripts, etc.) for the popular ImageNet object classification challenge. We then collaborated with the authors to convert their artifacts into the CK format, evaluate the converted artifacts on the original or similar platforms, and reproduce the results based on the rigorous artifact evaluation methodology [5]. The evaluation metrics included accuracy on the ImageNet validation set (50,000 images), latency (seconds per image), throughput (images per second), platform price (dollars) and peak power consumption (Watts). Since collapsing all metrics into one to select a single winner often results in over-engineered solutions, we decided to aggregate all reproduced results on a universal CK scoreboard shown in Figure 6 and let users select multiple implementations from a Pareto frontier, based on their requirements and constraints.

We then published all five papers with our unified artifact appendix [4] and a set of ACM reproducibility badges in the ACM Digital Library [38], accompanied by adaptive CK containers (CK-powered Docker) and portable CK workflows covering a very diverse model/software/hardware stack:

- **Models:** MobileNets, ResNet-18, ResNet-50, Inception-v3, VGG16, AlexNet, SSD.
- **Data types:** 8-bit integer, 16-bit floating-point (half), 32-bit floating-point (float).
- **AI frameworks and libraries:** MXNet, TensorFlow, Caffe, Keras, Arm Compute Library, cuDNN, TVM, NNVM.
- **Platforms:** Xilinx Pynq-Z1 FPGA, Arm Cortex CPUs and Arm Mali GPGPUs (Linaro HiKey960 and T-Firefly RK3399), a farm of Raspberry Pi devices, NVIDIA Jetson TX1 and TX2, and Intel Xeon servers in Amazon Web Services, Google Cloud and Microsoft Azure.

The reproduced results also exhibited great diversity:

- **Latency:** 4 .. 500 milliseconds per image
- **Throughput:** 2 .. 465 images per second
- **Top 1 accuracy:** 41 .. 75 percent
- **Top 5 accuracy:** 65 .. 93 percent
- **Model size (pre-trained weights):** 2 .. 130 megabytes
- **Peak power consumption:** 2.5 .. 180 Watts
- **Device frequency:** 100 .. 2600 megahertz
- **Device cost:** 40 .. 1200 dollars
- **Cloud usage cost:** 2.6E-6 .. 9.5E-6 dollars per inference

The community can now access all the above CK workflows under permissive licenses and continue collaborating on them via dedicated GitHub projects with CK repositories. These workflows can be automatically adapted to new platforms and environments by either detecting already installed dependencies (frameworks, libraries, datasets) or rebuilding dependencies using CK meta packages supporting Linux, Windows, MacOS and Android. They can be also extended to expose new design and optimization choices such as quantization, as well as evaluation metrics such as power or memory consumption. We also used these CK workflows to crowdsource the design space exploration across devices provided by volunteers such as mobile phones, laptops and servers with the best solutions aggregated on live CK scoreboards [20].

After validating that my portable CK program workflow can support reproducible papers for deep learning systems, I decided to conduct one more test to check if CK could also support quantum computing R&D. Quantum computers have the potential to solve certain problems dramatically faster than conventional computers, with applications in areas such as machine

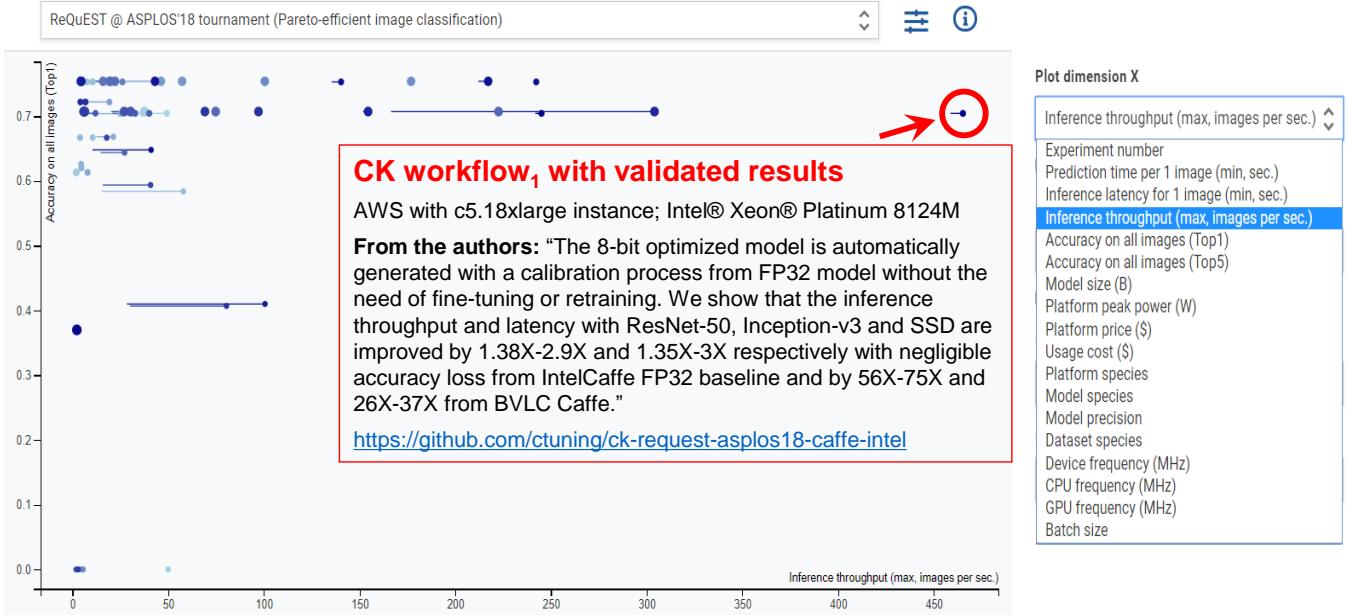


Figure 6: The CK dashboard with reproduced results from the ACM ASPLOS-REQUEST’18 tournament to co-design Pareto-efficient image classification in terms of speed, accuracy, energy and other costs. Each paper submission was accompanied by the portable CK workflow to be able to reproduce and compare results in a unified way.

learning, drug discovery, materials, optimization, finance and cryptography. However it is not yet known when the first demonstration of quantum advantage will be achieved, or what shape it will take.

This led me to co-organize several Quantum hackathons similar to the REQUEST tournament [26] with IBM, Rigetti, Riverlane and dividiti. My main goal was to check if we could aggregate and share multidisciplinary knowledge about the state-of-the-art in quantum computing using portable CK workflows that can run on classical hardware and quantum platforms from IBM, Rigetti and other companies, can be connected to a public dashboard to simplify reproducibility and comparison of different algorithms across different platforms, and can be extended by the community even after hackathons.

Figure 7 shows the results from one of such Quantum hackathons where over 80 participants, from undergraduate and graduate students to startup founders and experienced professionals from IBM and CERN, worked together to solve a quantum machine learning problem designed by Riverlane. All participants were given some labelled quantum data and asked to develop algorithms for solving a classification problem.

We also taught participants how to perform these experiments in a collaborative, reproducible and automated way using the CK framework so that the results could be transferred to industry. For example, we introduced the CK repository with workflows and components for the Quantum Information Science Kit (QISKit) - an open source software development kit (SDK) for working IBM Q quantum processors [13]. Using the CK program workflow from this repository,

the participants were able to start running quantum experiments with a standard CK command:

```
ck pull repo --url=https://github.com/ctuning/ck-qiskit
ck run program:qiskit-demo --cmd_key=quantum_coin_flip
```

Whenever ready, the participants could submit their solutions to the public CK dashboards to let other users validate and reuse their results [20, 26].

Following the successful validation of portable CK workflows for reproducible papers, I continued collaborating with ACM [1] and ML and systems conferences to automate the tedious artifact evaluation process [5, 42]. For example, we developed several CK workflows to support the Student Cluster Competition Reproducibility Challenge (SCC) at the Supercomputing conference [15]. We demonstrated that it was possible to reuse the CK program workflow to automate the installation, execution, customization and validation of the SeisSol application (Extreme Scale Multi-Physics Simulations of the Tsunamigenic 2004 Sumatra Megathrust Earthquake) [54] from the SC18 Student Cluster Competition Reproducibility Challenge across several supercomputers and HPC clusters [14]. We also showed that it was possible to abstract HPC job managers including Slurm and Flux and connect them with our portable CK workflows.

Some authors already started using CK to share their research artifacts and workflows at different ML and systems conferences during artifact evaluation [28]. My current goal is to make the CK onboarding as simple as possible and help researchers to automatically convert their ad-hoc artifacts and scripts into CK workflows, reusable artifacts, adaptive containers and live



Figure 7: The CK dashboard connected with portable CK workflows to visualize and compare public results from reproducible Quantum Hackathons. Over 80 participants worked together to solve a quantum machine learning problem and minimise time to solution.

dashboards.

5.6 Connecting researchers and practitioners to co-design efficient computational systems

CK use cases demonstrated that it was possible to develop and use a common research infrastructure with different levels of abstraction to bridge the gap between researchers and practitioners and help them to collaboratively co-design efficient computational systems. Scientists could then work with a higher-level abstraction while allowing engineers to continue improving the lower-level abstractions for continuously evolving software and hardware in deploying new techniques in production without waiting for each other, as shown in Figure 8. Furthermore, the unified interfaces and meta descriptions of all CK components and workflows made it possible to

explain what was happening inside complex and "black box" computational systems, integrate them with legacy systems, use them inside "adaptive" Docker, and share them along with published papers while applying the DevOps methodology and agile principles in scientific research.

6 CK platform

The practical use of CK as a portable and customizable workflow framework in multiple academic and industrial projects exposed several limitations:

- The distributed nature of the CK technology, the lack of a centralized place to keep all CK components, automation actions and workflows, and the lack of a convenient GUI made it very challenging to keep track of all contributions from the community.

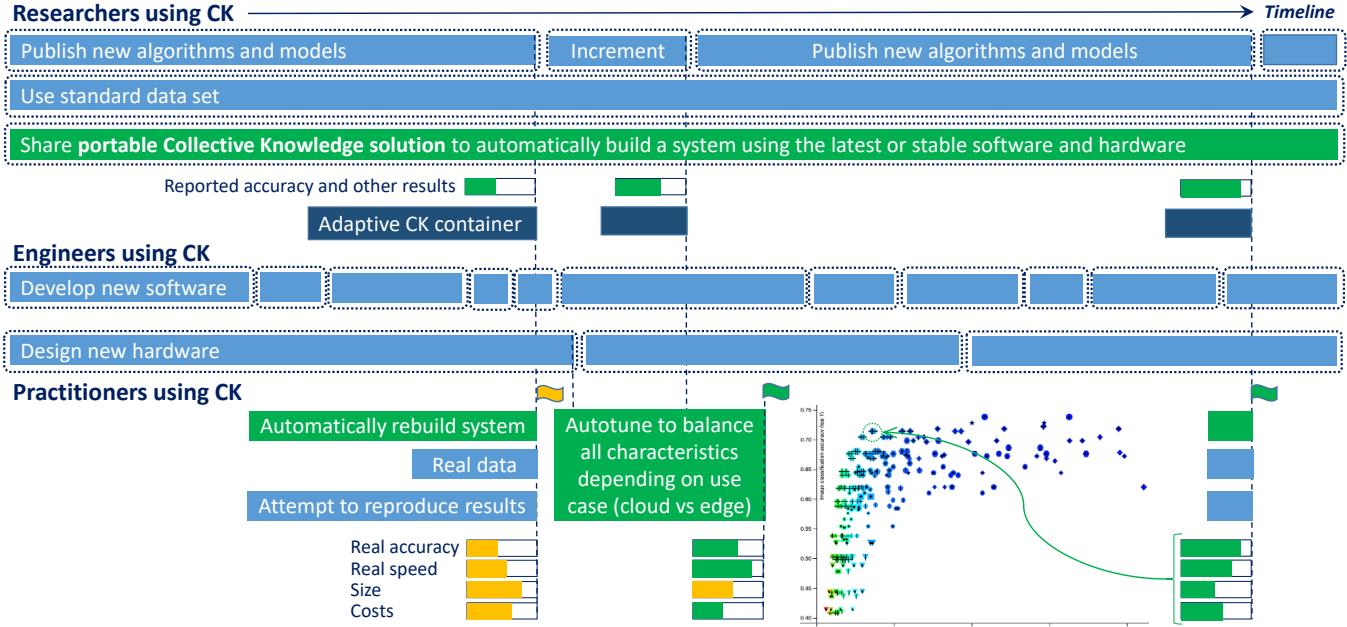


Figure 8: The CK concept helps to connect researchers and practitioners to co-design complex computational systems using DevOps principles while automatically adapting to continuously evolving software, hardware, models and datasets. The CK framework and public dashboards also help to unify, automate and crowdsource the benchmarking and auto-tuning process across diverse components from different vendors to automatically find the most efficient systems on the Pareto frontier.

As a result, it is not easy to discuss and test APIs, add new components and assemble workflows, automatically validate them across diverse platforms and environments, and connect them with legacy systems.

- The concept of backward compatibility of CK APIs and the lack of versioning similar to Java made it challenging to keep stable and bug-free workflows in the real world: any bug in a reusable CK component from one GitHub project could easily break dependent workflows in another GitHub project.
- The CK command-line interface with the access to all automation actions with numerous parameters was too low-level for researchers. This is similar to the situation with Git, a powerful but quite complex and CLI-based tool that requires extra web services such as GitHub and GitLab to make it more user-friendly.

This feedback from CK users motivated me to start developing cKnowledge.io (Figure 9), an open web-based platform with a GUI to aggregate, version and test all CK components and portable workflows. I also wanted to replace aging optimization repository at cTuning.org with an extensible and modular platform to crowdsource and reproduce tedious experiments such as benchmarking and co-design of efficient systems for AI and ML across diverse platforms and data provided by volunteers.

The CK platform is inspired by GitHub and PyPI: I see it as a collaborative platform to share reusable automation actions for repetitive research tasks and assemble portable workflows. It also includes the open-source CK client [16] that provides a common API to initialise, build, run and validate different research projects based on a simple JSON or YAML manifest. This client is connected with live scoreboards on the CK platform to collaboratively reproduce and compare the state-of-the-art research results during artifact evaluation that we helped to organize at ML and Systems conferences [20].

My intention is to use the CK platform to complement and enhance MLPerf, the ACM Digital Library, PapersWithCode.com, and existing reproducibility initiatives and artifact evaluation at ACM, IEEE and NeurIPS conferences with the help of CK-powered adaptive containers, portable workflows, reusable components, "live" papers and reproducible results validated by the community using realistic data across diverse models, software and hardware.

7 CK demo: automating and customizing AI/ML benchmarking

I prepared a live and interactive demonstration of the CK solution that automates the MLPerf inference

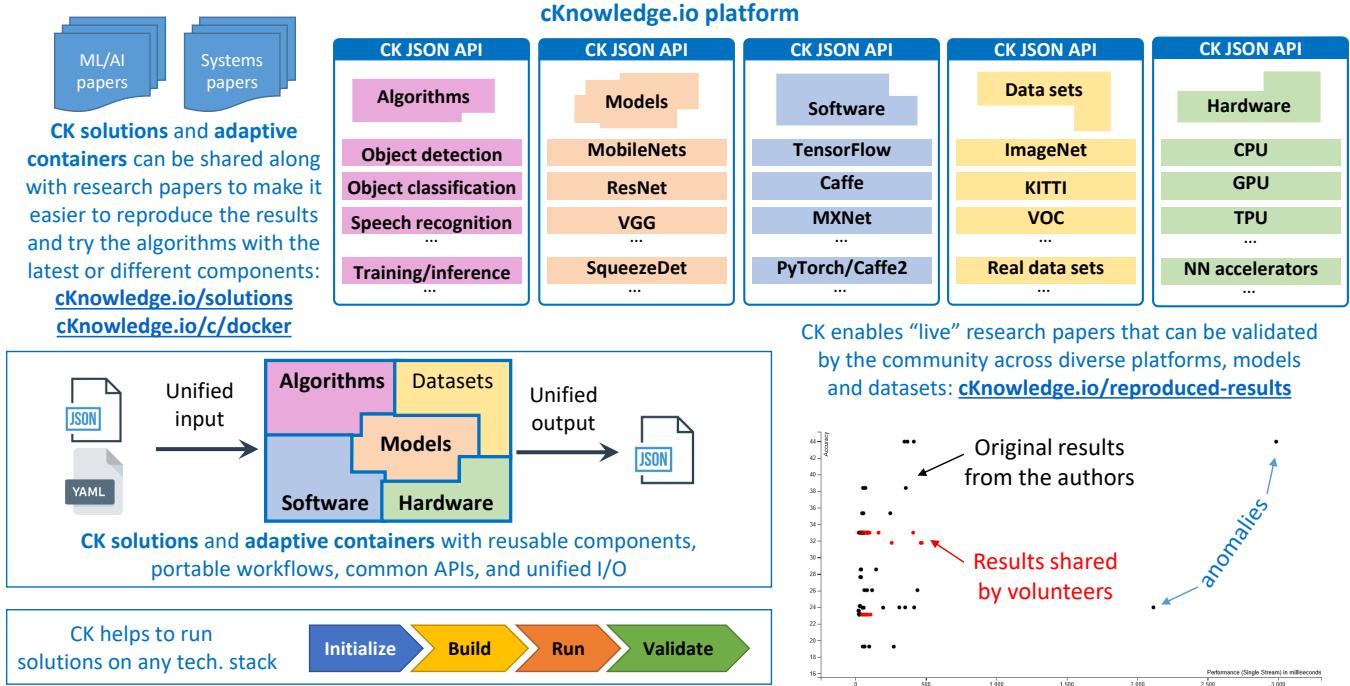


Figure 9: cKnowledge.io: an open platform to organize scientific knowledge in the form of portable workflows, reusable components and reproducible research results. It already contains many automation actions and components needed to co-design efficient and self-optimizing computational systems, enable reproducible and live papers validated by the community, and keep track of the state-of-the-art research techniques that can be deployed in production.

benchmark [53], connects it with the live CK dashboard (public optimization repository) [20], and helps volunteers to crowdsource benchmarking and design space exploration across diverse platforms similar to the Collective Tuning Initiative [41] and the SETI@home project: cKnowledge.io/test.

This demonstration shows how to use a unified CK API to automatically build, run and validate object detection based on SSD-Mobilenet, TensorFlow and COCO dataset across Raspberry Pi computers, Android phones, laptops, desktops and data centers. This solution is based on a simple JSON file describing the following tasks and their dependencies on CK components:

- prepare a Python virtual environment (can be skipped for the native installation)
- download and install the Coco dataset (50 or 5000 images)

- detect C++ compilers or Python interpreters needed for object detection
- install Tensorflow framework with a specified version for a given target machine
- download and install the SSD-MobileNet model compatible with selected Tensorflow
- manage installation of all other dependencies and libraries
- compile object detection for a given machine and prepare pre/post-processing scripts

This solution was published on the cKnowledge.io platform using the open-source CK client [16] to help users participate in crowd-benchmarking on their own machines as follows:

```
# Install the CK client from PyPi using:
pip install cbench

# Download and build the solution on a given machine (example for Linux):
cb init demo-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows

# Run the solution on a given machine:
cb benchmark demo-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows
```

The users can then see the benchmarking results (speed, latency, accuracy and other exposed characteristics through the CK workflow) on the live CK dashboard associated with this solution and compare them against the official MLPerf results or with the results shared by other users: cKnowledge.io/demo-result.

After validating this solution on a given platform, the users can also clone it and update the JSON description to retarget this benchmark to other devices and operating

systems such as MacOS, Windows, Android phones, servers with CUDA-enabled GPUs and so on.

The users have the possibility to integrate such ML solutions with production systems with the help of unified CK APIs as demonstrated by connecting the above CK solution for object detection with the webcam in any browser: cKnowledge.io/demo-solution.

Finally, it is possible to use containers with CK repositories, workflows and common APIs as follows:

```
docker run ctuning/cbrain-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows \
/bin/bash -c "cb benchmark demo-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows"

docker run ctuning/cbench-mlperf-inference-v0.5-detection-openvino-ssd-mobilenet-coco-500-linux \
/bin/bash -c "cb benchmark mlperf-inference-v0.5-detection-openvino-ssd-mobilenet-coco-500-linux"
```

Combining Docker and portable CK workflows enables "adaptive" CK containers for MLPerf that can be easily customized, rebuilt with different ML models, datasets, compilers, frameworks and tools encapsulated inside CK components, and deployed in production [2].

8 Conclusions and future work

My very first research project to prototype an analog neural network stalled in the late 1990s because it took me far too long to build from scratch all the infrastructure to model and train Hopfield neural networks, generate diverse datasets, co-design and optimize software and hardware, run and reproduce all experiments, compare them with other techniques from published papers, and use this technology in practice in a completely different environment.

In this article, I explain why I have developed the Collective Knowledge framework and how it can help to address the issues above by organizing all research projects as a database of reusable components, portable workflows and reproducible experiments based on FAIR principles (findable, accessible, interoperable, and reusable). I also describe how the CK framework attempts to bring DevOps and "Software 2.0" principles to scientific research and to help users share and reuse best practices, automation actions and research artifacts in a unified way alongside reproducible papers. Finally, I demonstrate how the CK concept helps to complement, unify and interconnect existing tools, platforms, and reproducibility initiatives with common APIs and extensible meta descriptions rather than rewriting them or competing with them.

I present several use cases of how CK helps to connect researchers and practitioners to collaboratively design more reliable, reproducible and efficient computational systems for machine learning, artificial intelligence and other emerging workloads that can automatically adapt

to continuously evolving software, hardware, models and datasets. I also describe the <https://cKnowledge.io> platform that I have developed to organize knowledge particularly about AI, ML, systems and other innovative technology in the form of portable CK workflows, automation actions, reusable artifacts and reproducible results from research papers. My goal is to help the community to find useful methods from research papers, quickly build them on any tech stack, integrate them with new or legacy systems, start using them in the real world with real data, and combine Collective Knowledge and AI to build better AI systems.

Finally, I demonstrate the concept of "live" research papers connected with portable CK workflows and online CK dashboards to let the community automatically validate and update experimental results even after the project, detect and share unexpected behaviour, and fix problems collaboratively [45]. I believe that such a collaborative approach can make computational research more reproducible, portable, sustainable, explainable and trustable.

However, CK is still a proof-of-concept and there remains a lot to simplify and improve. Future work will intend to make CK more user-friendly and to simplify the onboarding process, as well as to standardise all APIs and JSON meta descriptions. It will also focus on development of a simple GUI to create and share automation actions and CK components, assemble portable workflows, run experiments, compare research techniques, generate adaptive containers, and participate in lifelong AI, ML and systems optimization.

My long-term goal is to use CK to develop a virtual playground, an optimization repository and a marketplace where researchers and practitioners assemble AI, ML and other novel applications similar to live species that continue to evolve, self-optimize and compete with each other across diverse tech stacks from different vendors and users. The winning solutions with the best

trade-offs between speed, latency, accuracy, energy, size, costs and other metrics can then be selected at any time from the Pareto frontier based on user requirements and constraints. Such solutions can be immediately deployed in production on any platform from data centers to the edge in the most efficient way thus accelerating AI, ML and systems innovation and digital transformation.

Acknowledgements

I thank Sam Ainsworth, Erik Altman, Lorena Barba, Victor Bittorf, Unmesh D. Bordoloi, Steve Brierley, Luis Ceze, Milind Chabbi, Bruce Childers, Nikolay Chunosov, Marco Cianfriglia, Albert Cohen, Cody Coleman, Chris Cummins, Jack Davidson, Alastair Donaldson, Achi Dosanjh, Thibaut Dumontet, Debojyoti Dutta, Daniil Efremov, Nicolas Essayan, Todd Gamblin, Leo Gordon, Wayne Graves, Christophe Guillon, Herve Guillou, Stephen Herbein, Michael Heroux, Patrick Hesse, James Hetherington, Kenneth Hoste, Robert Hundt, Ivo Jimenez, Tom St. John, Timothy M. Jones, David Kanter, Yuriy Kashnikov, Gaurav Kaul, Sergey Kolesnikov, Shriram Krishnamurthi, Dan Laney, Andrei Lascu, Hugh Leather, Wei Li, Anton Lokhmotov, Peter Mattson, Thierry Moreau, Dewey Murdick, Luigi Nardi, Cedric Nugteren, Michael O’Boyle, Ivan Ospiov, Bhavesh Patel, Gennady Pekhimenko, Massimiliano Picone, Ed Plowman, Ramesh Radhakrishnan, Ilya Rahkovsky, Vijay Janapa Reddi, Vincent Rehm, Alka Roy, Shubhadeep Roychowdhury, Dmitry Savenko, Aaron Smith, Jim Spohrer, Michel Steuwer, Victoria Stodden, Robert Stojnic, Michela Taufer, Stuart Taylor, Olivier Temam, Eben Upton, Nicolas Vasilache, Flavio Vella, Davide Del Vento, Boris Veysman, Alex Wade, Pete Warden, Dave Wilkinson, Matei Zaharia, Alex Zhigarev and other great colleagues for interesting discussions, practical use cases and useful feedback.

References

- [1] ACM Pilot demo ”Collective Knowledge: packaging and sharing”. <https://youtu.be/DIkZxraTmGM>.
- [2] Adaptive CK containers with a unified CK API to customize, rebuild, and adapt them to any platform and environment as well as to integrate them with external tools and data. <https://cKnowledge.io/c/docker>.
- [3] Amazon SageMaker: fully managed service to build, train, and deploy machine learning models quickly. <https://aws.amazon.com/sagemaker>.
- [4] Artifact Appendix and reproducibility checklist to unify and automate the validation of results at systems and machine learning conferences. <https://cTuning.org/ae/checklist.html>.
- [5] Artifact Evaluation: reproducing results from published papers at machine learning and systems conferences. <https://cTuning.org/ae>.
- [6] Automation actions with a unified API and JSON IO to share best practices and introduce DevOps principles to scientific research. <https://cKnowledge.io/actions>.
- [7] CK-compatible GitHub, GitLab, and BitBucket repositories with reusable components, automation actions, and workflows. <https://cKnowledge.io/repos>.
- [8] CK dashboard with results from CK-powered, automated and multi-objective design space exploration of image classification across different models, software and hardware. <https://cKnowledge.io/ml-optimization-repository>.
- [9] CK GitHub: an open-source and cross-platform framework to organize software projects as a database of reusable components with common automation actions, unified APIs, and extensible meta descriptions based on FAIR principles (findability, accessibility, interoperability, and reusability). <https://github.com/ctuning/ck>.
- [10] CK repository with CK workflows and components to reproduce the MILEPOST project. <https://github.com/ctuning/reproduce-milepost-project>.
- [11] CK use case from General Motors: Collaboratively Benchmarking and Optimizing Deep Learning Implementations. <https://youtu.be/1ldgVZ64hEI>.
- [12] Collective Knowledge real-world use cases. <https://cKnowledge.org/partners>.
- [13] Collective Knowledge Repository for the Quantum Information Software Kit (QISKit). <https://github.com/ctuning/ck-qiskit>.
- [14] Collective Knowledge Repository to automate the installation, execution, customization and validation of the SeisSol application from the SC18 Student Cluster Competition Reproducibility Challenge across different platforms, environments and datasets. <https://github.com/ctuning/ck-scc18/wiki>.
- [15] Collective Knowledge workflow to prepare digital artifacts for the Student Cluster Competition Reproducibility Challenge. <https://github.com/ctuning/ck-scc>.
- [16] Cross-platform CK client to unify preparation, execution, and validation of research techniques shared along with research papers. <https://github.com/ctuning/cbench>.

- [17] DAWN Bench: an end-to-end deep learning benchmark and competition. <https://dawn.cs.stanford.edu/benchmark/>.
- [18] Imagenet challenge (ILSVRC): Imagenet large scale visual recognition challenge where software programs compete to correctly classify and detect objects and scenes. <http://www.image-net.org>.
- [19] Kaggle: platform for predictive modelling and analytics competitions. <https://www.kaggle.com>.
- [20] Live scoreboards with reproduced results from research papers at machine learning and systems conferences. <https://cKnowledge.io/reproduced-results>.
- [21] LPIRC: low-power image recognition challenge. <https://rebootingcomputing.ieee.org/lpirc>.
- [22] MLPerf: a broad ML benchmark suite for measuring performance of ML software frameworks, ML hardware accelerators, and ML cloud platforms. <https://mlperf.org>.
- [23] MLPerf Inference Benchmark results. <https://mlperf.org/inference-results>.
- [24] PapersWithCode.com provides a free and open resource with Machine Learning papers, code and evaluation tables. <https://paperswithcode.com>.
- [25] Portable and reusable CK workflows. <https://cKnowledge.io/programs>.
- [26] Quantum Collective Knowledge and reproducible quantum hackathons: keeping track of the state-of-the-art in quantum computing using portable CK workflows, live CK dashboards, and reproducible results. <https://cKnowledge.io/c/event/reproducible-quantum-hackathons/>.
- [27] Reproduced papers with ACM badges based on the cTuning evaluation methodology. <https://cKnowledge.io/reproduced-papers>.
- [28] Reproduced papers with CK workflows during artifact evaluation at ML and Systems conferences. <https://cKnowledge.io/?q=reproduced-papers AND portable-workflow-ck>.
- [29] ReQuEST: open tournaments on collaborative, reproducible and Pareto-efficient software/hardware co-design of deep learning and other emerging workloads. <https://cKnowledge.io/c/event/request-reproducible-benchmarking-tournament/>.
- [30] Shared and versioned CK modules. <https://cKnowledge.io/modules>.
- [31] Shared CK meta packages (code, data, models) with automated installation across diverse platforms. <https://cKnowledge.io/packages>.
- [32] Shared CK plugins to detect software (models, frameworks, data sets, scripts) on a user machine. <https://cKnowledge.io/soft>.
- [33] The Collective Knowledge platform: organizing knowledge about artificial intelligence, machine learning, computational systems, and other innovative technology in the form of portable workflows, automation actions, and reusable artifacts from reproduced papers. <https://cKnowledge.io>.
- [34] The Machine Learning Toolkit for Kubernetes. <https://www.kubeflow.org>.
- [35] The public database of image misclassifications during collaborative benchmarking of ML models across Android devices using the CK framework. http://cKnowledge.org/repo/web.php?wcid=model.image.classification:collective_training_set.
- [36] Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. Common Workflow Language, v1.0. <https://doi.org/10.6084/m9.figshare.3115156.v2>, 7 2016.
- [37] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefer. A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning. IEEE, May 2019. The 33rd IEEE International Parallel & Distributed Processing Symposium (IPDPS'19).
- [38] Luis Ceze Ceze, Natalie Enright Jerger, Babak Falsafi, Grigori Fursin, Anton Lokhmotov, Thierry Moreau, Adrian Sampson, and Phillip Stanley Marbell. ACM ReQuEST'18: Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-Designing Pareto-Efficient Deep Learning. 2018.
- [39] Cheng Li and Abdul Dakkak and Jinjun Xiong and Wei Wei and Lingjie Xu and Wen-mei Hwu. Across-Stack Profiling and Characterization of Machine Learning Models on GPUs. <https://arxiv.org/abs/1908.06869>, 2019.
- [40] Bruce R. Childers, Grigori Fursin, Shriram Krishnamurthi, and Andreas Zeller. Artifact Evaluation for Publications (Dagstuhl Perspectives Workshop 15452). *Dagstuhl Reports*, 5(11):29–35, 2016.
- [41] Grigori Fursin. Collective Tuning Initiative: automating and accelerating development and optimization of computing systems. <https://hal.inria.fr/inria-00436029v2>, June 2009.

- [42] Grigori Fursin. Enabling reproducible ML and Systems research: the good, the bad, and the ugly. <https://doi.org/10.5281/zenodo.4005773>, 2020.
- [43] Grigori Fursin and Christophe Dubach. Community-driven reviewing and validation of publications. In *Proceedings of the 1st ACM SIGPLAN Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering*, TRUST ’14, New York, NY, USA, 2014. Association for Computing Machinery.
- [44] Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, Francois Bodin, Phil Barnard, Elton Ashton, Edwin Bonilla, John Thomson, Christopher Williams, and Michael F. P. O’Boyle. Milepost gcc: Machine learning enabled self-tuning compiler. *International Journal of Parallel Programming*, 39:296–327, 2011. 10.1007/s10766-010-0161-2.
- [45] Grigori Fursin, Anton Lokhmotov, Dmitry Savenko, and Eben Upton. A Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques. <https://cknowledge.io/report/rpi3-crowd-tuning-2017-interactive>, January 2018.
- [46] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futrat. The spack package manager: bringing order to hpc software chaos. In *SC ’15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2015.
- [47] Kenneth Hoste, Jens Timmerman, Andy Georges, and Stijn Weirdt. Easybuild: Building software with ease. pages 572–582, 11 2012.
- [48] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. The popper convention: Making reproducible systems evaluation practical. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1561–1570, 2017.
- [49] Gaurav Kaul, Suneel Marthi, and Grigori Fursin. Scaling deep learning on AWS using C5 instances with MXNet, TensorFlow, and BigDL: From the edge to the cloud. <https://conferences.oreilly.com/artificial-intelligence/ai-eu-2018/public/schedule/detail/71549>, 2018.
- [50] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.
- [51] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 265–276, 2009.
- [52] QuantumBlack. Kedro: the open source library for production-ready machine learning code. <https://github.com/quantumblacklabs/kedro>, 2019.
- [53] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. MLPerf Inference Benchmark. <https://arxiv.org/abs/1911.02549>, 2019.
- [54] Carsten Uphoff, Sebastian Rettenberger, Michael Bader, Elizabeth H. Madden, Thomas Ulrich, Stephanie Wollherr, and Alice-Agnes Gabriel. Extreme scale multi-physics simulations of the tsunamigenic 2004 sumatra megathrust earthquake. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’17, New York, NY, USA, 2017. Association for Computing Machinery.
- [55] 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, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.
- [56] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 05 2013.

- [57] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41:39–45, 2018.