The Collective Knowledge project: making ML models more portable and reproducible with open APIs, reusable best practices and MLOps

Grigori Fursin

cTuning foundation and cKnowledge SAS

cKnowledge.io

June 18, 2020

Abstract

This article provides an overview of the Collective Knowledge technology (CK or cKnowledge). CK attempts to make it easier to reproduce ML&systems research, deploy ML models in production, and adapt them to continuously changing data sets, models, research techniques, software, and hardware. The CK concept is to decompose complex systems and ad-hoc research projects into reusable sub-components with unified APIs, CLI, and JSON meta description. Such components can be connected into portable workflows using DevOps principles combined with reusable automation actions, software detection plugins, meta packages, and exposed optimization parameters. CK workflows can automatically plug in different models, data and tools from different vendors while building, running and benchmarking research code in a unified way across diverse platforms and environments. Such workflows also help to perform whole system optimization, reproduce results, and compare them using public or private scoreboards on the cKnowledge.io platform. For example, the modular CK approach was successfully validated with industrial partners to automatically co-design and optimize software, hardware, and machine learning models for reproducible and efficient object detection in terms of speed, accuracy, energy, size, and other characteristics. The long-term goal is to simplify and accelerate the development and deployment of ML models and systems by helping researchers and practitioners to share and reuse their knowledge, experience, best practices, artifacts, and techniques using open CK APIs.

Keywords: *machine learning, systems, portability, reproducibility, reusability, automation, reusable best practices, portable MLOps, MLSysOps, DevOps, portable workflows, collaborative benchmarking, optimization, software/harware/model co-design, collective knowledge, open API*

1 Motivation

10 years ago I developed the cTuning.org platform and released all my research code and data to the public to crowdsource the training of our machine learning based compiler (MILEPOST GCC) [19]. I intended to accelerate this very time consuming autotuning process and help our compiler to learn the most efficient optimizations across real programs, data sets, platforms, and environments provided by volunteers.

We had a great response from the community and it took me just a few days to collect as many optimization results as during my entire PhD research. 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 and the run-time state of the system could influence performance while I did not have a mechanism to detect such changes [29, 22]. Even worse, I could not compare these empirical results with other published techniques because they rarely included the full experiment specification and all the necessary artifacts along with shared research code to be able to reproduce results. Furthermore, it was always a nightmare to add new tools, benchmarks and data sets 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.

These problems motivated me to establish the non-profit cTuning foundation and work on a common methodology and open-source tools to enable collaborative, reproducible, reusable, and trustable R&D. My foundation has supported multiple reproducibility initiatives at systems and machine learning conferences in col-

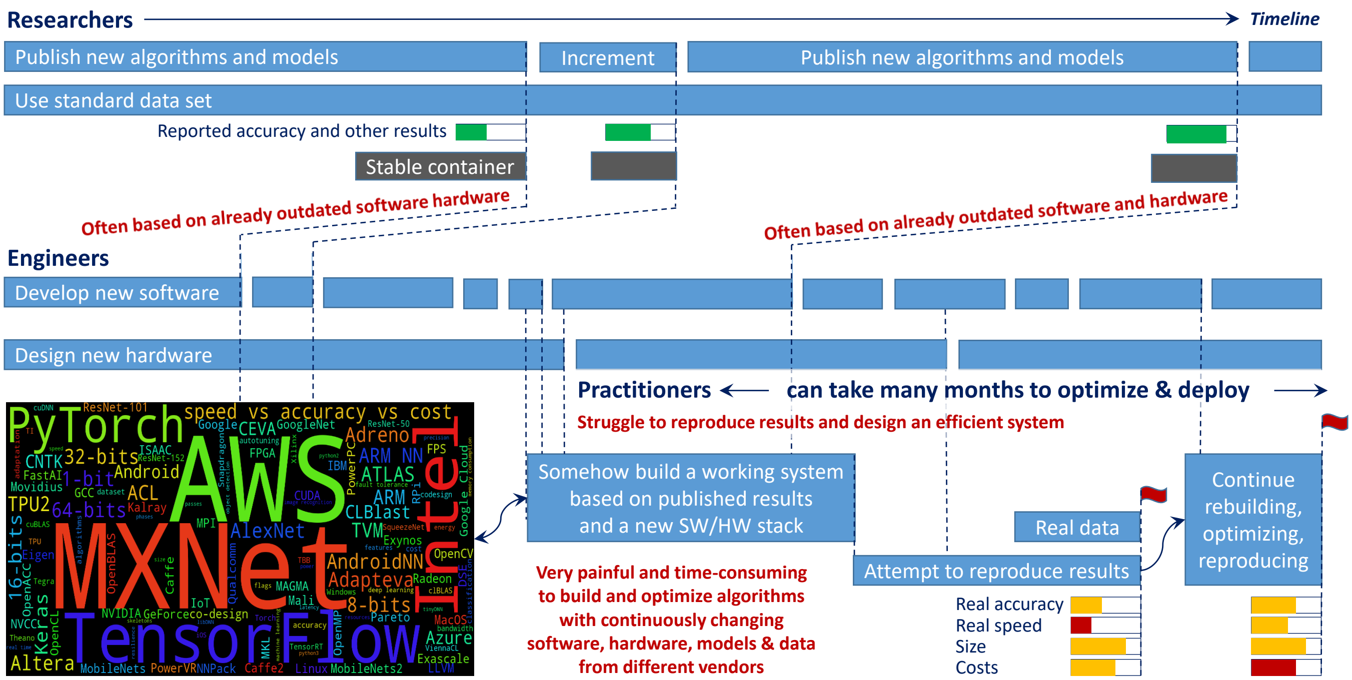

Figure 1: Reproducing research papers and adopting novel techniques in production is a tedious, repetitive and time consuming process because of continuously changing software, hardware, models and datasets, and a lack of common formats and APIs for shared code, models, and artifacts.

laboration with ACM. We also promoted sharing of code, artifacts and results in a unified way along with research papers [3, 18]. It gave me a unique chance to participate in reproducibility studies of more than 100 research papers at MLSys, ASPLOS, CGO, PPoPP, Supercomputing, and other computer science conferences during the past 5 years [12]. I also started deploying some of these techniques in production in collaboration with my industrial partners to better understand all the problems when building trustable, reproducible, and production-ready computational systems.

This practical experience confirmed my previous findings: while sharing ad-hoc research code, artifacts, and trained models along with research papers is a great step forward, it is only a tip of the reproducibility iceberg [21]. The major challenge afterwards is to figure out how to integrate such code and models with complex production systems and run them in a reliable and efficient way across rapidly evolving software, heterogeneous hardware and legacy platforms with continuously changing interfaces and data formats while balancing multiple characteristics including speed, latency, accuracy, memory size, power consumption, reliability, and costs (Figure 1).

2 Collective Knowledge framework

As the first step to deal with this chaos, I introduced an Artifact Appendix and a reproducibility checklist. My goal was to help researchers describe how to reproduce their research techniques in a unified way across different conferences and journals [2, 12]. It was striking to notice that most of the research projects used some ad-hoc scripts often with hardwired paths to perform the same repetitive tasks including downloading models and data sets, detecting required software, building and testing research code, preparing target platforms, running experiments, validating outputs, reproducing results, plotting graphs, and generating papers. This motivated me to search for a solution to automate such common tasks and make them reusable and customizable across different research projects.

First, I started looking at related tools that were introduced to automate experiments, make research more reproducible and make it easier to deploy machine learning in production:

- ML workflow frameworks such as MLFlow [32], Kedro [30] and Amazon SageMaker [1] help to abstract and automate high-level ML operations. However, unless used inside AWS or DataBricks cloud they still have limited support for the complex system integration and optimization particularly when targeting embedded devices and IoT - the last mile of MLOps.

- ML benchmarking initiatives such as MLPerf [31], MLModelScope [28] and Deep500 [17] attempt to standardize benchmarking and co-design of models and systems. However, production deployment, integration with complex systems and adaptation to

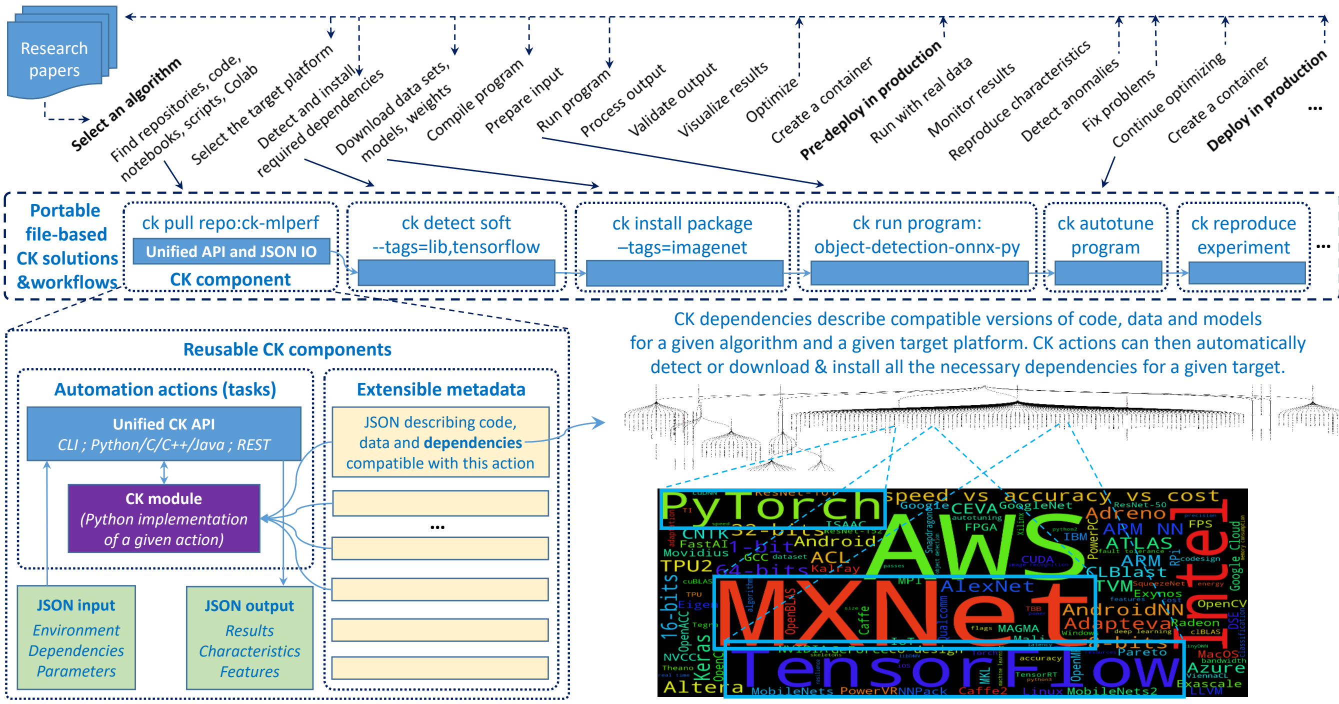

Figure 2: Collective Knowledge framework helps to convert ad-hoc research artifacts (code, data, models, results) into reusable components, automation actions and portable workflows with a unified CLI, Python API and JSON meta description shared along with papers. The goal is to make it easier to reproduce, reuse, adopt and build upon ML&systems research.

continuously changing tools, user environments and data are out of their scope.

- Package managers such as Spack [25] and EasyBuild [26] are very useful to rebuild the whole environment with fixed software versions. However adaptation to existing environments, native cross-compilation and support for non-software packages (models, data sets) is still in progress.

- Docker, Kubernetes and other container-based technology is very useful to prepare and share stable software releases. However, it hides all the software chaos rather than solving it, has some performance overheads, requires an enormous amount of space, have very poor support for embedded devices and do not help to integrate models to existing projects and user data.

- PapersWithCode.com platform helps to find relevant research code for published machine learning papers and keep track of the state-of-the-art machine learning research using public scoreboards. However, my experience suggests that sharing ad-hoc research code is not enough to make research techniques reproducible, customizable, portable and trustable.

While working with these useful tools and platforms I realized that a higher-level API can help to connect them into portable workflows with reusable artifacts that can adapt to never-ending changes in systems and environments. That's why I decided to develop the Collective Knowledge framework (CK or cKnowledge) [23, 20] - a small and cross-platform Python framework that helps to convert ad-hoc research projects into a file-based database of reusable CK components [13] (code, data, models, pre-/post-processing scripts, experimental results, R&D automation actions [4], best research practices to reproduce results, and live papers) with unified Python and REST APIs, common command-line interface, JSON meta information and JSON input/output (Figure 2). I also provided reusable API to automatically detect different software, models and datasets on a user machine or install/cross-compile the missing ones while supporting different operating systems (Linux, Windows, MacOS, Android) and hardware (Nvidia, Arm, Intel, AMD ...).

Such an approach allows researchers to create, share and reuse flexible APIs with JSON input/output for different AI/ML frameworks, libraries, compilers, models and datasets, connect them into unified workflows instead of hardwired scripts, and make them portable [11] using automatic software detection plugins [15] and meta-packages [14]. It also helps to make research more reliable and reproducible by decomposing complex computational systems into reusable, portable, customizable, and non-virtualized CK components. Finally, the CK concept is to be non-intrusive and complement, abstract and interconnect all existing tools including MLFlow, SageMaker, Kedro, Spack, EasyBuild, MLPerf, Docker, and Kuber-

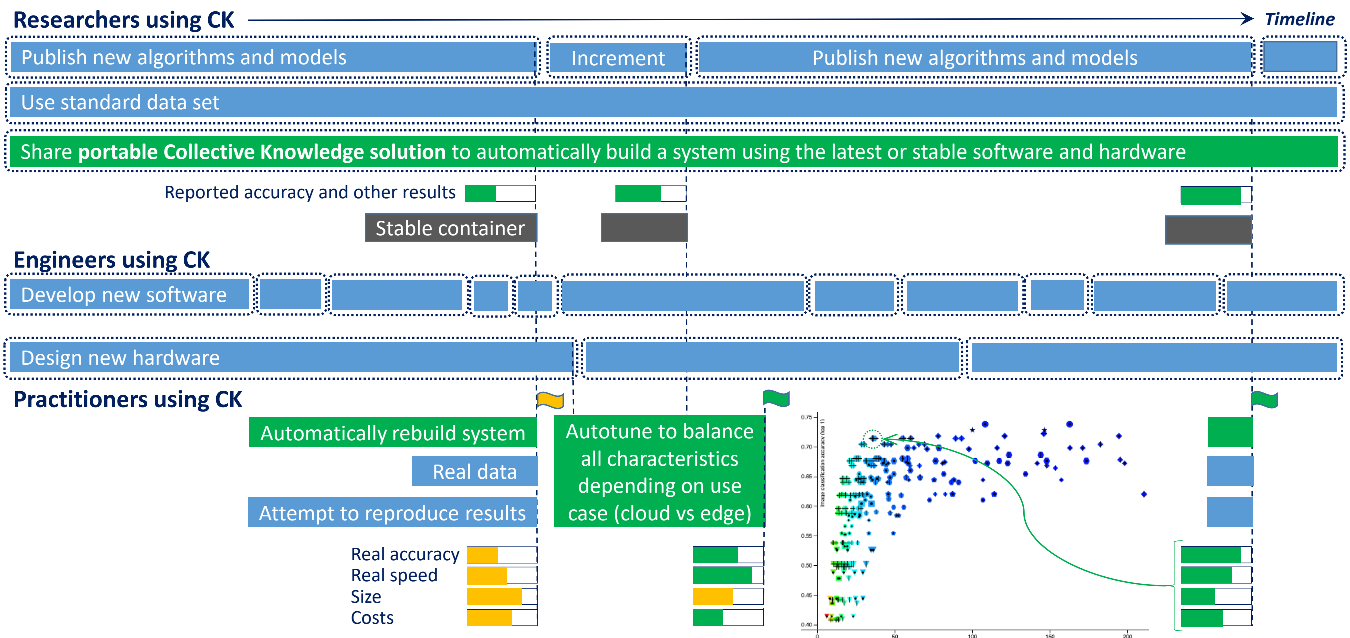

Figure 3: Portable CK workflows with reusable components can connect researchers and practitioners to co-design complex computational systems using DevOps principles while adapting to continuously changing software, hardware, models and data sets. CK framework also helps to unify, automate and crowdsource the benchmarking and autotuning process across diverse components from different vendors to automatically find the most efficient systems on the Pareto frontier.

netes while making them more adaptive and system aware rather than replacing or rewriting them.

My long-term objective is to provide a common research infrastructure with different levels of abstraction that can bridge the gap between researchers and practitioners and help them to collaboratively co-design complex computational systems that can be immediately used in production as shown in Figure 3. Scientists can then work with a higher-level abstraction of such a system while allowing engineers to continue improving the lower-level abstractions for evolving software and hardware without breaking the system.

Furthermore, the unified interfaces and meta descriptions of all CK components and workflows make it possible to better understand what is happening inside complex and "black box" computational systems, integrate them with production and legacy systems, use them inside Docker and Kubernetes, share them along with published papers, and apply the DevOps methodology and agile principles in scientific research.

3 Collective Knowledge platform

During the past 4 years, CK has been validated in different academic and industrial projects as a portable and modular workflow framework. CK helped to enable reproducible experiments, optimize software and hardware stacks for emerging AI, ML and quantum workloads, bridge the gap between high-level ML operations and systems, and support MLOps [7]. The authors of 18 research papers used CK to share their research artifacts and workflows at different ML&systems conferences [16].

While CK helped to automate benchmarking, optimization and co-design of complex computational systems and make it easier to deploy them in production [6] I also noticed three major limitations:

- The distributed nature of the CK technology, the lack of a centralized place to keep all CK APIs, workflows and components, and the lack of convenient GUIs made it very challenging to keep track of all contributions from the community, add new components, assemble workflows, automatically test them across diverse platforms, 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 real life - any bug in a reusable CK component from one GitHub project could easily break dependent workflows in another GitHub project.

- CK command-line interface was too low-level and not very user friendly.

The feedback from CK users motivated me to start developing cKnowledge.io (Figure 4) - an open web-based platform to aggregate, version and test all CK components, APIs, and portable CK workflows. This

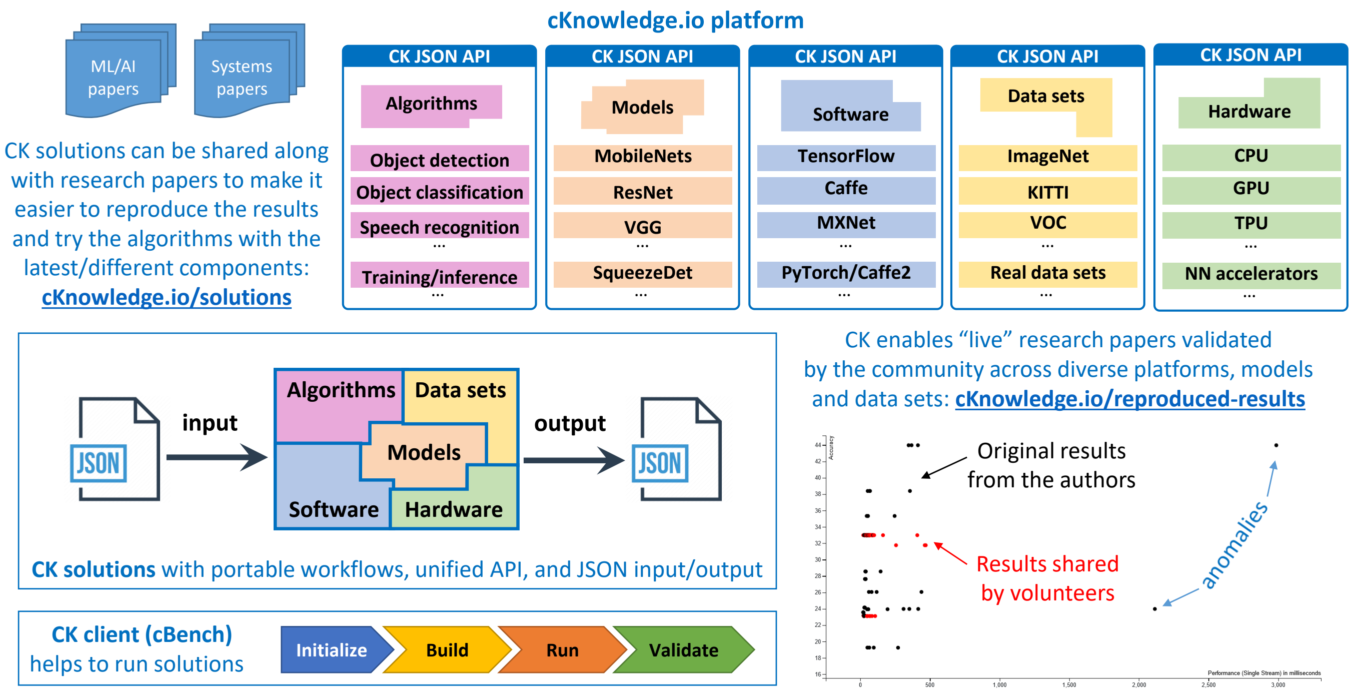

Figure 4: cKnowledge.io: a prototype of an open platform to share and reuse all the basic blocks and APIs needed to co-design efficient and self-optimizing computational systems, enable live papers validated by the community, and keep track of the state-of-the-art machine learning and systems research with the help of portable workflows and reproducible crowd-benchmarking.

is necessary to support collaborative and reproducible ML&systems research and deploy ML models in production across diverse systems, data sets and environments from IoT to data centers. The CK platform is inspired by GitHub and PyPI: I see it as a collaborative platform to share reusable research APIs, portable workflows, reproducible solutions, and associated reproducible results. It also includes the open-source CK client [8] to provide a common API to initialize, build, run and validate different research projects based on a simple JSON manifest describing all CK dependencies and installation/execution/validation recipes for different tasks and target platforms. Such a platform can be used to keep track of reproducible and reusable state-of-the-art AI, ML and systems research by connecting portable workflows and reusable artifacts with live scoreboards to validate and compare experimental results during Artifact Evaluation at different conferences [9].

I believe that the combination of the CK framework and the CK platform can make it easier to implement and share portable workflows for research code assembled from stable and versioned CK components with unified APIs. Such modular workflows can help to keep track of all the information flow within such workflows, expose and modify all configuration and optimization parameters via simple JSON input files, combine public and private code and data, monitor system behavior, retarget research code and machine learning models to different platforms from IoT to cloud, use them inside containers, integrate them with legacy systems, reproduce results, and generate reproducible papers with live scoreboards.

4 Collective Knowledge use cases

As the first practical use case, I decided to convert all artifacts, workflows and automation tasks from my past research related to self-learning and self-optimizing computer systems into reusable CK components. I shared them with the community in CK-compatible Git repositories [5] and started reproducing my past research results with new software, hardware, data sets and deep learning models [9]. I also implemented a customizable and portable benchmarking and autotuning pipeline (workflow) that could perform software/hardware co-design in a unified way across different programs, data sets, frameworks, compilers and platforms as shown in Figure 5.

Such a pipeline helped to gradually expose different design choices and optimization parameters from all subcomponents (models, frameworks, compilers, run-time systems, hardware) via unified CK APIs and enable the whole system autotuning. It also helped to keep track of all information passed between sub-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.

I then decided to validate the CK concept of reusability by using the same pipeline in another collaborative

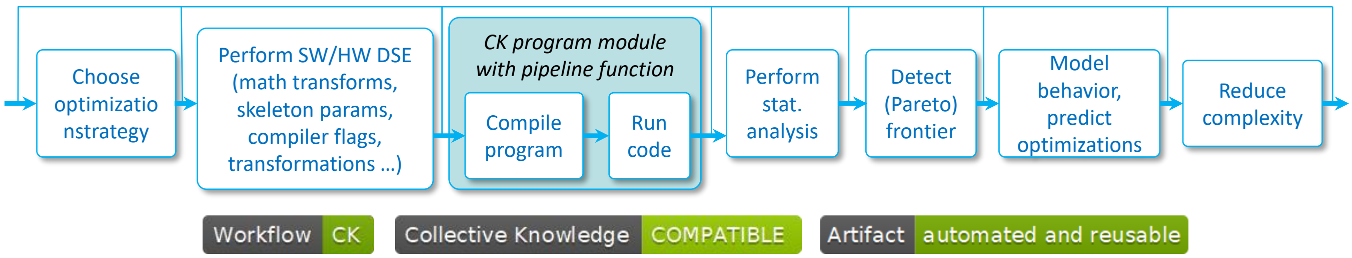

Figure 5: Reusable benchmarking and autotuning pipeline assembled from portable CK components with unified APIs.

project with the Raspberry Pi foundation. The practical task was to crowdsource compiler autotuning across multiple Raspberry Pi devices to improve their performance. CK helped to automate experiments, collect performance numbers on live CK scoreboards, and plug in CK components with various machine learning and predictive analytics techniques including decision trees, nearest neighbor classifiers, support vector machines (SVM) and deep learning to automatically learn the most efficient optimizations [24]. It also demonstrated the possibility to reduce the growing technology transfer gap between academia and industry by reusing portable workflows that can adapt to evolving systems and can be integrated with existing and legacy projects. For example, the same pipeline was successfully reused by General Motors to collaboratively benchmark and optimizing deep learning implementations [6] and by Amazon to enable scaling of deep learning on AWS using C5 instances with MXNet, TensorFlow, and BigDL from the edge to the cloud [27]. Finally, my CK autotuning pipeline was reused and extended by dividiti to make it easier to prepare, submit and reproduce MLPerf inference benchmark results [10].

5 Collective Knowledge demo: automating, sharing and reproducing MLPerf inference benchmarks

I prepared a live and interactive demo of the CK solution that automates the MLPerf inference benchmark, connects it with the live CK dashboard and crowdsource benchmarking across diverse platforms provided by volunteers similar to SETI@home: cKnowledge.io/demo. This demo shows how to use CK APIs 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 the 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 [8] to help users to participate in crowd-benchmarking using their own machines as follows:

1. install CK client from PyPi using:

   pip install cbench

2. download and install the solution on a given machine (example for Linux):

   cb init demo-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows

3. run the solution on a given machine:

   cb benchmark demo-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows

The users can then see their measurements (speed, latency, accuracy and other exposed characteristics) and compare them against the official MLPerf results or with the results shared by other users with the help of the live CK dashboard associated with this solution: cKnowledge.io/result/sota-mlperf-object-detection-v0.5-crowd-benchmarking.

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 with Docker, Android phones, servers with CUDA-enabled GPUs, and so on.

Finally, the users can integrate such ML solutions with production systems with the help of unified CK APIs as demonstrated by connecting above CK solution for object detection with the webcam in the browser: cKnowledge.io/solution/demo-obj-detection-coco-tf-cpu-webcam-linux-azure.

6 Conclusions and future work

My very first research project to prototype semiconductor neural network stalled in the late 90s because it took me way too long to build all the infrastructure from scratch to generate data sets, train neural networks, prepare and optimize hardware, run all experiments and optimize the NN implementation. Since then, I have always been looking for solutions to enable more efficient computer systems and accelerate ML&systems research.

I have developed the CK framework and cKnowledge.io platform to bring DevOps, MLOps, reusability and agile principles to ML&systems research, and connect researchers and practitioners to co-design more reliable, reproducible and efficient computational systems that can adapt to continuously changing software, hardware, models, and data sets. I hope that CK will help to share and reuse best practices and pack research techniques and artifacts along with research papers to make it easier to reproduce results and deploy them in production. I also want to enable "live" research papers by connecting CK workflows with live dashboards to let the community reproduce results, detect unexpected behavior, and collaboratively fix problems in shared workflows and components [24].

However, CK is still a proof-of-concept and there is a lot to be improved. For example, I would like to make it more user friendly, standardize APIs and JSON meta descriptions of all CK components and workflows, and develop a simple GUI to share CK components, assemble workflows, run experiments, and compare research techniques similar to LEGO. My dream is to use CK to build a virtual world (playground) where researchers and practitioners assemble AI, ML and other novel algorithms similar to live species that can continue to evolve, self-optimize and compete with each other across devices and data provided by volunteers. At the same time, the winning solutions with the best trade-off in terms of speed, latency, accuracy, energy, size, and costs can be simplify picked from the Pareto frontier at any time and immediately deployed in production thus accelerating AI, ML and systems research and making AI practical.

Software and Data

All code and data can be found at cKnowledge.io under permissive license.

Acknowledgements

I would like to thank Unmesh D. Bordoloi, Nikolay Chunosov, Marco Cianfriglia, Cody Coleman, Achi Dosanjh, Debojyoti Dutta, Nicolas Essayan, Todd Gamblin, Leo Gordon, Wayne Graves, Herve Guillou, Michael Heroux, James Hetherignton, Ivo Jimenez, Gaurav Kaul, Sergey Kolesnikov, Dan Laney, Anton Lokhmotov, Peter Mattson, Thierry Moreau, Dewey Murdick, Cedric Nugteren, Bhavesh Patel, Gennady Pekhimenko, Massimiliano Picone, Ed Plowman, Vijay Janapa Reddi, Alka Roy, Michel Steuwer, Victoria Stodden, Robert Stojnic, Stuart Taylor, Eben Upton, Flavio Vella, Boris Veytsman, Alex Wade, Matei Zaharia, Alex Zhigarev, and many other ACM, IEEE, NeurIPS and MLPerf colleagues for interesting discussions, practical use cases and useful feedback.

References

[1] Amazon sagemaker: fully managed service to build, train, and deploy machine learning models quickly. `https://aws.amazon.com/sagemaker`.

[2] Artifact Appendix and reproducibility checklist to unify and automate the validation of results at systems and machine learning conferences. `https://ctuning.org/ae/submission_extra.html`.

[3] Artifact Evaluation: reproducing experimental results from published research papers at systems and machine learning conferences (the common methodology, automation tools, artifact appendix and reproducibility checklist). `https://cTuning.org/ae`.

[4] Automation actions with a unified API and JSON IO to share best practices and introduce DevOps principles to scientific research. `https://cKnowledge.io/actions`.

[5] CK-compatible GitHub, GitLab and BitBucket repositories with CK APIs, components and workflows. `https://cKnowledge.io/repos`.

[6] CK use case from General Motors: collaboratively Benchmarking and Optimizing Deep Learning Implementations. `https://youtu.be/1ldgVZ64hEI`.

[7] Collective Knowledge real-world use cases. `https://cKnowledge.org/partners`.

[8] Cross-platform CK client to unify preparation, execution and validation of research techniques shared along with research papers. `https://github.com/ctuning/cbench`.

[9] Live scoreboards with reproduced results from research papers at machine learning and systems conferences. `https://cKnowledge.io/reproduced-results`.

[10] MLperf inference benchmark results. https://mlperf.org/inference-results.

[11] Portable and reusable CK workflows. https://cKnowledge.io/programs.

[12] Reproduced papers with ACM badges based on the cTuning evaluation methodology. https://cKnowledge.io/reproduced-papers.

[13] Shared and versioned CK modules. https://cKnowledge.io/modules.

[14] Shared CK meta packages (code, data, models) with automated installation across diverse platforms. https://cKnowledge.io/packages.

[15] Shared CK plugins to detect software (models, frameworks, data sets, scripts) on a user machine. https://cKnowledge.io/soft.

[16] Artifact Evaluation: reproducing experimental results from systems and machine learning conferences. https://cTuning.org/ae, 2015–current.

[17] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler. A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning. IEEE, May 2019. The 33rd IEEE International Parallel & Distributed Processing Symposium (IPDPS'19).

[18] 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.

[19] Grigori Fursin. Collective Tuning Initiative: automating and accelerating development and optimization of computing systems. In *Proceedings of the GCC Developers' Summit*, June 2009.

[20] Grigori Fursin. CK: an open-source and cross-platform framework to decompose complex computational systems and research projects into portable, customizable and reusable components and workflows with Python APIs, JSON input/output files and an integrated meta package manager. https://github.com/ctuning/ck, 2015–present.

[21] Grigori Fursin. CNRS presentation: enabling open and reproducible research at computer systems conferences: the good, the bad and the ugly. https://doi.org/10.5281/zenodo.2544204, March 2017.

[22] Grigori Fursin and Christophe Dubach. Community-driven reviewing and validation of publications. *CoRR*, abs/1406.4020, 2014.

[23] Grigori Fursin, Anton Lokhmotov, and Ed Plowman. Collective Knowledge: towards R&D sustainability. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'16)*, March 2016.

[24] Grigori Fursin, Anton Lokhmotov, Dmitry Savenko, and Eben Upton. A Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques. January 2018.

[25] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. 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.

[26] Kenneth Hoste, Jens Timmerman, Andy Georges, and Stijn Weirdt. Easybuild: Building software with ease. pages 572–582, 11 2012.

[27] 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.

[28] Cheng Li, Abdul Dakkak, Jinjun Xiong, Wei Wei, Lingjie Xu, and Wen mei Hwu. Across-stack profiling and characterization of machine learning models on gpus. https://arxiv.org/abs/1908.06869, 2019.

[29] 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*, page 265276, 2009.

[30] QuantumBlack. Kedro: the open source library for production-ready machine learning code. https://github.com/quantumblacklabs/kedro, 2019.

[31] 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.

[32] 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.