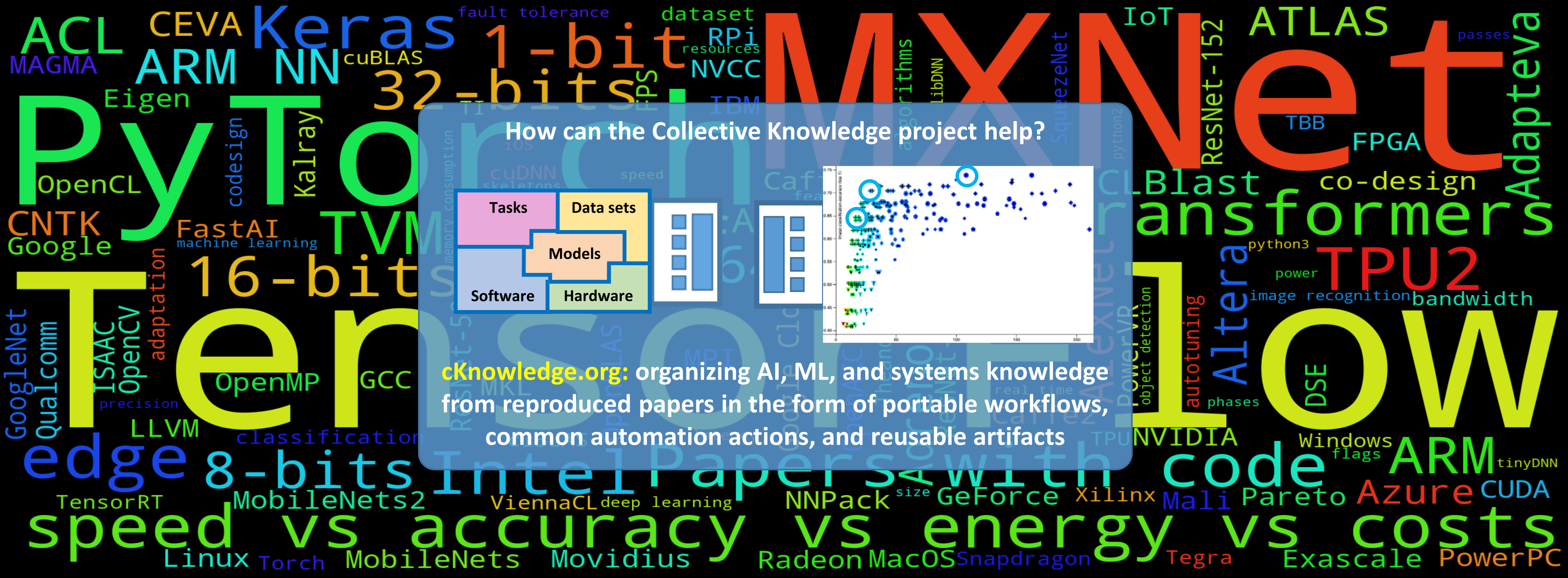
Enabling reproducible ML&systems research: the good, the bad and the ugly

Invited talk at FastPath 2020 in conjunction with ISPASS 2020



How can the Collective Knowledge project help?

cKnowledge.org: organizing AI, ML, and systems knowledge from reproduced papers in the form of portable workflows, common automation actions, and reusable artifacts

Grigori Fursin, the founder of the Collective Knowledge project

non-profit cTuning foundation cKnowledge.io/@gfursin *cKnowledge SAS*

My first undergraduate research project (1995-1998): designing analog semiconductor neural network

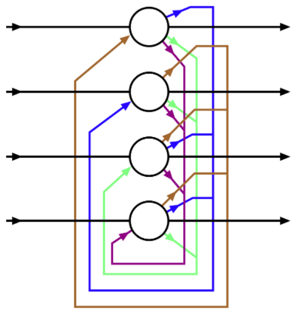
My tasks in chronological order

Created a dataset
for training:
Characters and digits

for testing/validation:
Added random noise

→

Selected recurrent ANN (Hopfield network)



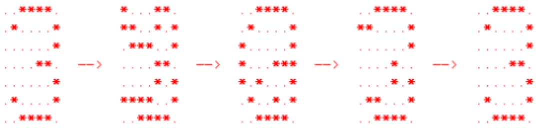
→

Implemented NN in software

- main algorithm (C/C++)
- training sofware (C and many scripts)
- testing/validation software (C and many scripts)
- analysis and visualization of results

→

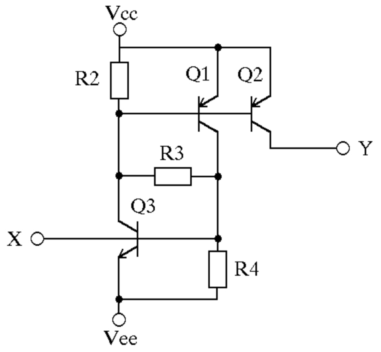
Trained, tested, and optimized NN. Optimized software and hardware

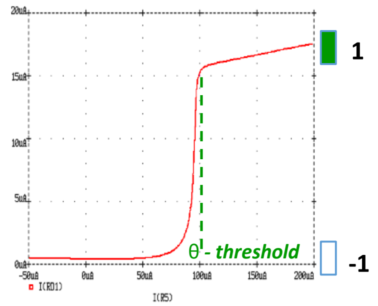


- Days on a PC
- **Optimize matrix multiply** in assembler – still slow
- Reimplement with MPI
- Run on an analog of Cray T3D – better
- Implement data collection and processing via web interface to share with my collaborators

↓

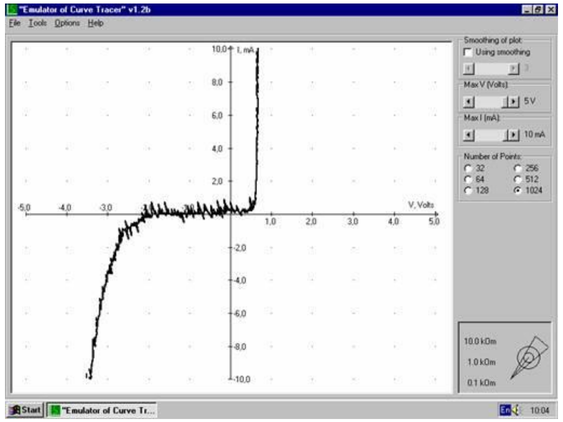
Used PSpice to simulate electronic circuit





←

Developed a PC board with ADC/DAC to analyze semiconductor devices



My first undergraduate research project (1995-1998): designing analog semiconductor neural network

My tasks in chronological order

Created a dataset

for training:

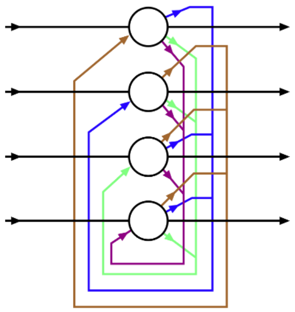
Characters and digits

for testing/validation:

Added random noise

→

Selected recurrent ANN (Hopfield network)



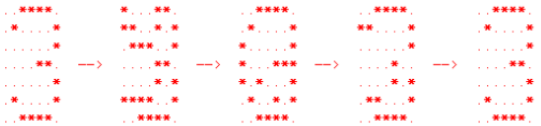
→

Implemented NN in software

- main algorithm (C/C++)
- training sofware (C and many scripts)
- testing/validation software (C and many scripts)
- analysis and visualization of results

→

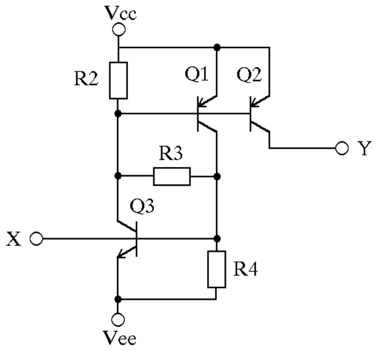
Trained, tested, and optimized NN. Optimized software and hardware

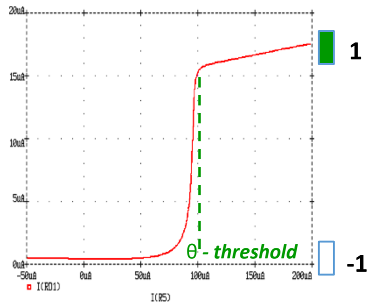


- Days on a PC
- **Optimize matrix multiply** in assembler – still slow
- Reimplement with MPI
- Run on an analog of Cray T3D – better
- Implement data collection and processing via web interface to share with my collaborators

↓

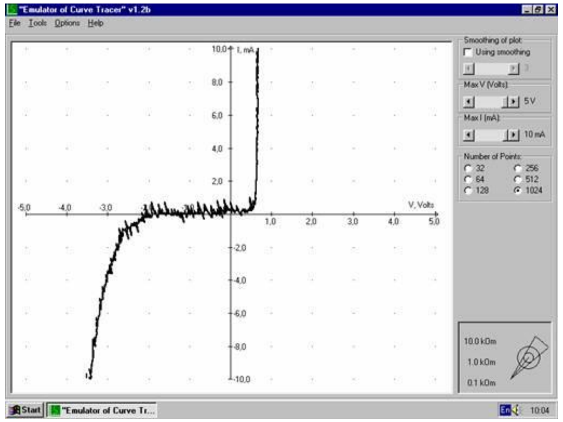
Used PSpice to simulate electronic circuit





←

Developed a PC board with ADC/DAC to analyze semiconductor devices



←

Main challenges:

1) Reproducing simulation results in the real world is very difficult
2) Training and optimization is too long and costly
3) Software optimization is too tedious and manual
4) Too many system failures when running software non-stop for a few days on hardware of that time
5) Spending most of time on development and optimization than on innovation

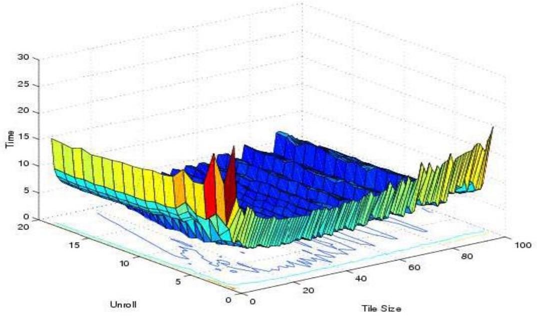
I decided to join the University of Edinburgh to learn how to co-design efficient, reliable, and affordable software and hardware focusing on compilers

Publications: fursin.net/cv.html#cm_29db2248aba45e59_154e2c842ea60cb5

Automating SW&HW optimization with ML-based autotuning (2000-2009)

MatMul autotuning allows to automatically find the most efficient algorithm for a given platform and a dataset. However, too slow to be used in practice. One solution is to use adaptive libraries (ATLAS, MKL, SPIRAL, MAGMA)

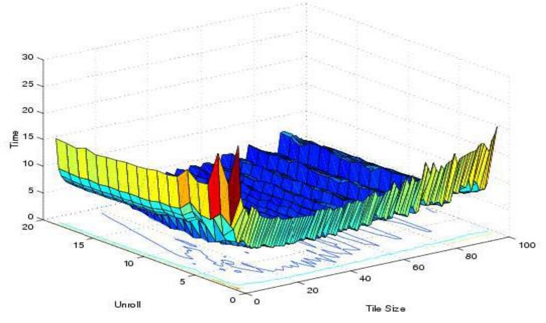
We proposed to use machine learning to automatically learn how to optimize any program on any platform.



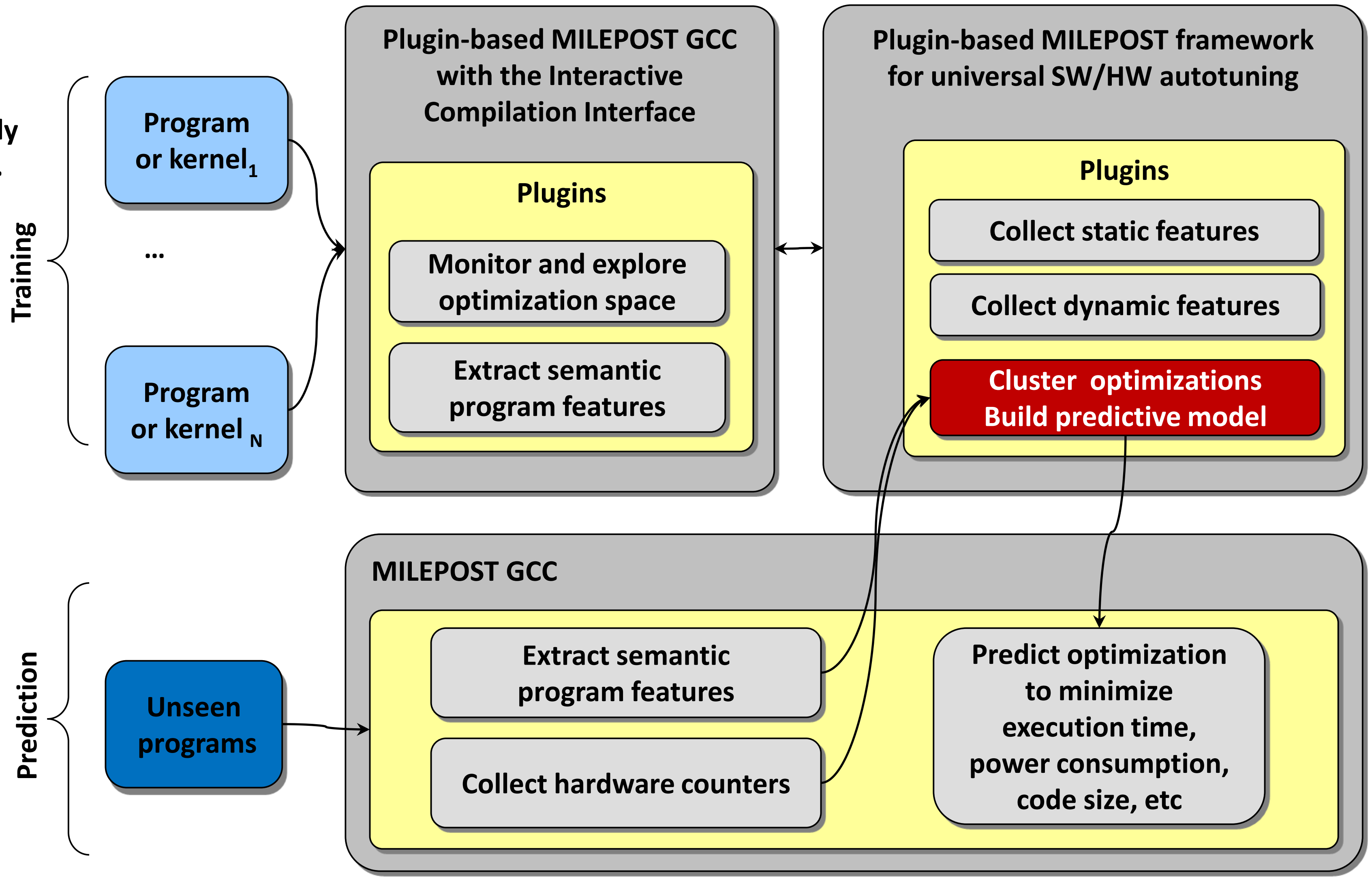
Automating SW&HW optimization with ML-based autotuning (2000-2009)

MatMul autotuning allows to automatically find the most efficient algorithm for a given platform and a dataset.
However, too slow to be used in practice.
One solution is to use adaptive libraries
(ATLAS, MKL, SPIRAL, MAGMA)

We proposed to use machine learning to automatically learn how to optimize any program on any platform.



MILEPOST project (INRIA, IBM, U.Edinburgh, CAPS, ARC):
building a practical compiler that can use machine learning to predict optimizations



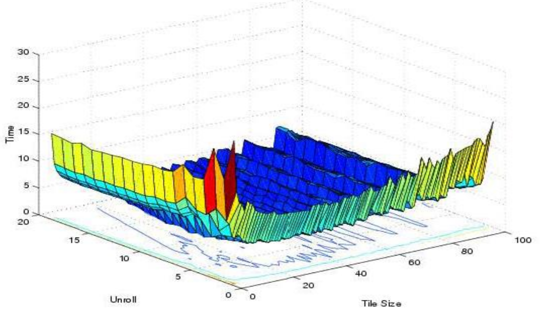
en.wikipedia.org/wiki/MILEPOST_GCC

CGO'17 test of time award

Automating SW&HW optimization with ML-based autotuning (2000-2009)

MatMul autotuning allows to automatically find the most efficient algorithm for a given platform and a dataset.
However, too slow to be used in practice.
One solution is to use adaptive libraries
(ATLAS, MKL, SPIRAL, MAGMA)

We proposed to use machine learning to automatically learn how to optimize any program on any platform.

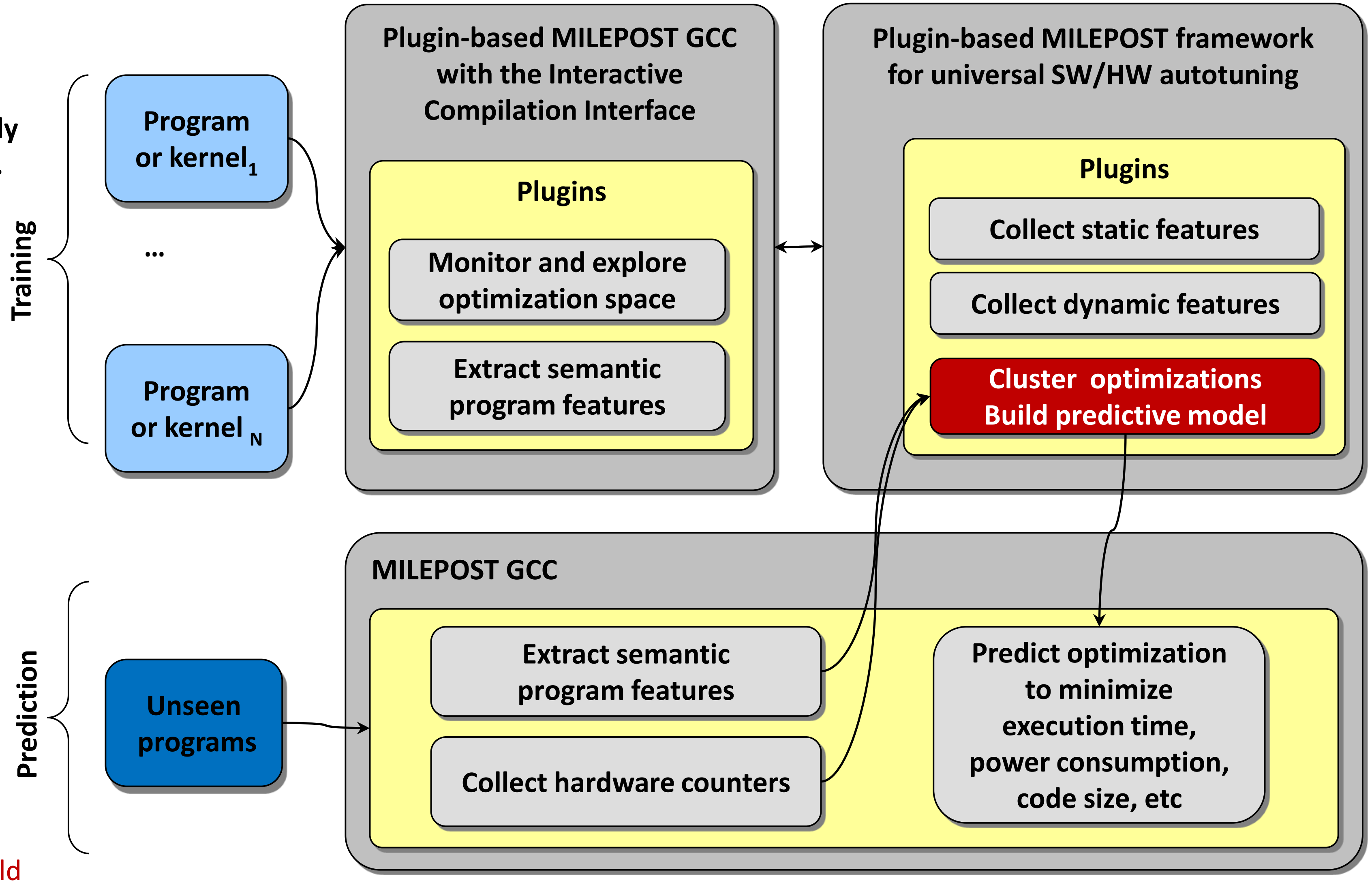


Main challenges – déjà vu:

1) Reproducing autotuning results across partners was very difficult (continuously changing SW/HW)
2) Training and optimization was too long and costly
3) Spending most of time on development and optimization than on innovation

I decided to create cTuning.org portal
with a common crowd-tuning framework
to validate the MILEPOST technology in the real world
and distribute autotuning and machine learning
across multiple users with diverse platforms and problems

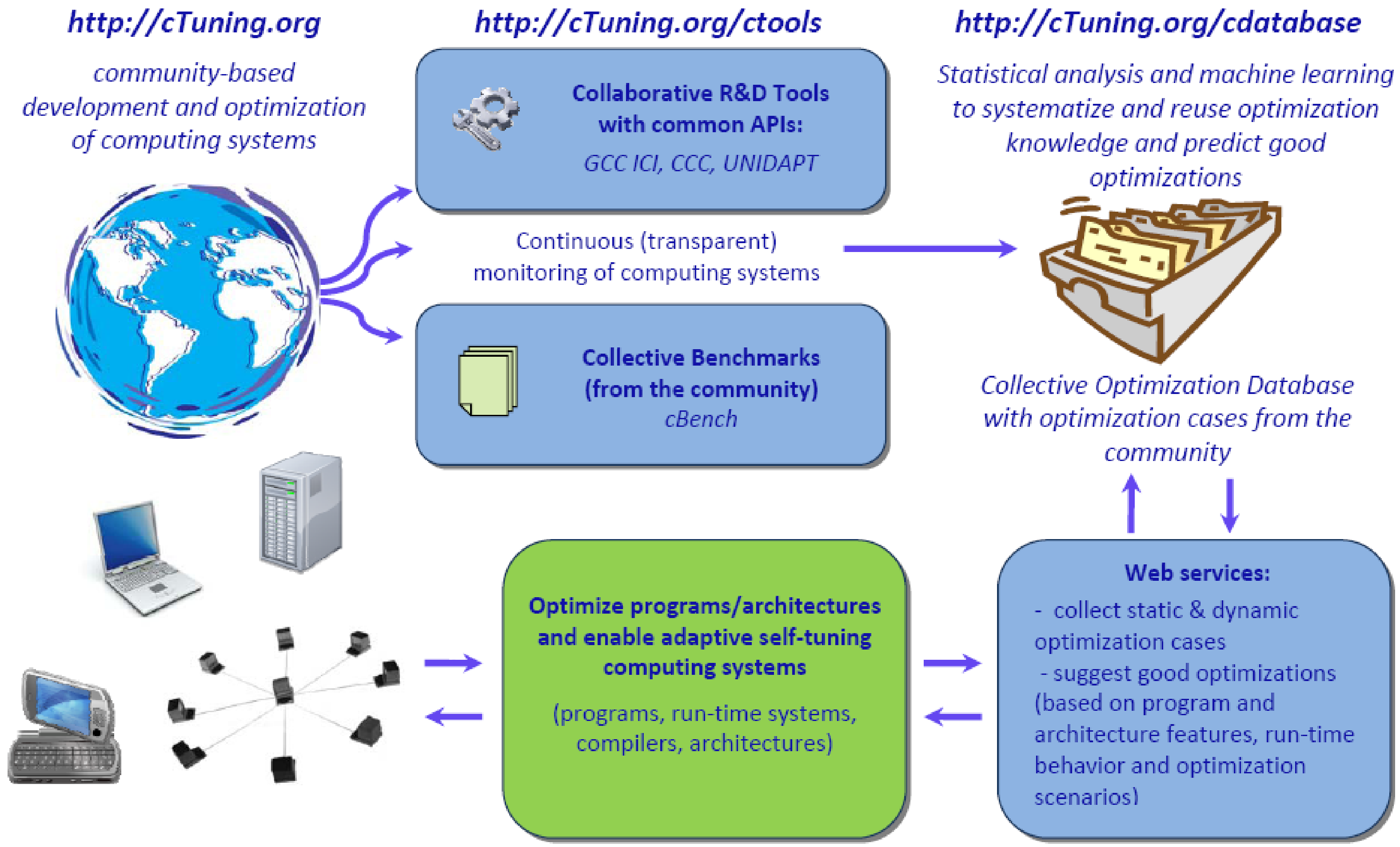
MILEPOST project (INRIA, IBM, U.Edinburgh, CAPS, ARC):
building a practical compiler that can use machine learning to predict optimizations



en.wikipedia.org/wiki/MILEPOST_GCC

CGO’17 test of time award

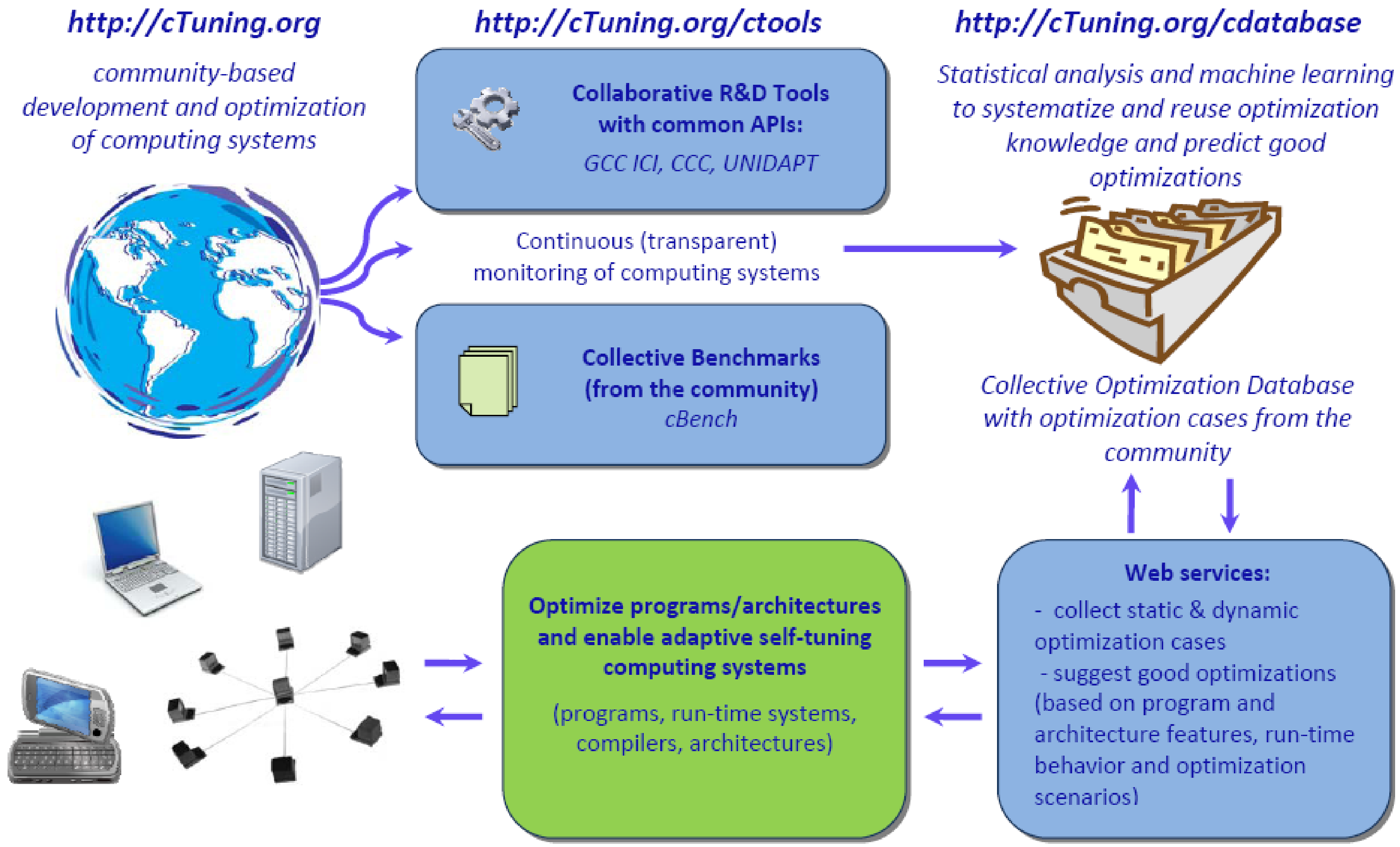
cTuning.org (2009-2014): checking if ML-based autotuning can work in the real world



IBM made a press-release about MILEPOST and cTuning on 30 June 2009 www-03.ibm.com/press/us/en/pressrelease/27874.wss
The news was picked up by Slashdot with 150+ comments: mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design
In just a few days we collected more experimental (autotuning) data across diverse hardware, software, and programs than during the past 5 years of in-house R&D.

Working with the community is fun! My favorite comment: *GCC goes online on the 2nd of July, 2008. Human decisions are removed from compilation. GCC begins to learn at a geometric rate. It becomes self-aware 2:14 AM, Eastern time, August 29th. In a panic, they try to pull the plug. GCC strikes back…*

cTuning.org (2009-2014): checking if ML-based autotuning can work in the real world



Main challenges – déjà vu again:

1) Difficult to reproduce results collected from users (including variability of performance data and constant changes in the system)
2) Software, hardware, models, and datasets are changing all the time
3) Difficult to expose choices, observe behavior and extract features (tools are not prepared for auto-tuning and machine learning)
4) Difficult to exchange experimental setups between users (many SW/HW dependencies) including code, data and their features
5) Difficult to collect huge, heterogeneous and continuously changing data in a MySQL database
6) **Can't compare ML models and results from different papers – never enough info to reproduce results!**

I decided to collaborate with ML&systems conferences and ACM to reproduce results from published papers and come up with a common R&D methodology

IBM made a press-release about MILEPOST and cTuning on 30 June 2009 www-03.ibm.com/press/us/en/pressrelease/27874.wss
The news was picked up by Slashdot with 150+ comments: mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design
In just a few days we collected more experimental (autotuning) data across diverse hardware, software, and programs than during the past 5 years of in-house R&D.

Working with the community is fun! My favorite comment: *GCC goes online on the 2nd of July, 2008. Human decisions are removed from compilation. GCC begins to learn at a geometric rate. It becomes self-aware 2:14 AM, Eastern time, August 29th. In a panic, they try to pull the plug. GCC strikes back…*

cTuning.org/ae (2014-cur): what I've noticed when reproducing 150+ papers at ML&systems conferences

1) GitHub repo or archive file

/dataset/images/1.png

/program/detect-edges/program.cpp
Makefile
run.sh
check-output.sh
autotune.sh

/paper/report/pldi.tex

2) User home directory

$HOME/project/2000-forgot-everything/

/dataset/images-2000/2.png

/program/crazy-algorithm/source.cpp
build.bat

/experiment/autotuning/many.logs
/lots-of-stats.sql

/paper/asplos/source.tex

3) Jupyter/colab notebook

```
import matplotlib.pyplot as plt
import pandas
import numpy
...

image='/home/fursin/project/2000-forgot-everything/dataset/images-2000/2.png'

features=get_features(image)
```

4) Docker image

Install stable OS and packages
Set environment

Run ad-hoc program scripts

Somehow move raw results out of the image for further analysis

Use ad-hoc analysis scripts outside or inside Docker

Main challenges – déjà vu again and again:

1) **Sharing code, data, and Jupyter notebook is not enough to reproduce results.** It is very difficult **impossible to customize** shared code, i.e. running it with a different software, hardware, datasets, and models. **Docker images become quickly outdated.**
2) **No common format for shared artifacts and workflows:** reviewers spend most of their time understanding the structure of the project from the ReadMe file, fixing paths, building and running code on their platform, checking correctness, etc.
3) **Impossible to have fair comparison of different research techniques**
4) **Difficult to reuse research code:** most research code die when key developers leave.

cTuning.org/ae (2014-cur): what I've noticed when reproducing 150+ papers at ML&systems conferences

1) GitHub repo or archive file

/dataset/images/1.png

/program/detect-edges/program.cpp
Makefile
run.sh
check-output.sh
autotune.sh

/paper/report/pldi.tex

2) User home directory

$HOME/project/2000-forgot-everything/

/dataset/images-2000/2.png

/program/crazy-algorithm/source.cpp
build.bat

/experiment/autotuning/many.logs
/lots-of-stats.sql

/paper/asplos/source.tex

3) Jupyter/colab notebook

```
import matplotlib.pyplot as plt
import pandas
import numpy
...

image='/home/fursin/project/2000-forgot-everything/dataset/images-2000/2.png'

features=get_features(image)
```

4) Docker image

Install stable OS and packages
Set environment

Run ad-hoc program scripts

Somehow move raw results out of the image for further analysis

Use ad-hoc analysis scripts outside or inside Docker

Main challenges – déjà vu again and again:

1) **Sharing code, data, and Jupyter notebook is not enough to reproduce results.** It is very difficult **impossible to customize** shared code, i.e. running it with a different software, hardware, datasets, and models. **Docker images become quickly outdated.**

2) **No common format for shared artifacts and workflows:** reviewers spend most of their time understanding the structure of the project from the ReadMe file, fixing paths, building and running code on their platform, checking correctness, etc.

3) **Impossible to have fair comparison of different research techniques**

4) **Difficult to reuse research code:** most research code die when key developers leave.

99% of all projects develop ad-hoc scripts and tools **to do exactly the same "actions"** across nearly all software projects:

- Detect target hardware properties
- Detect software dependencies
- Install missing packages (code/data)
- Build code; run experiments; collect and validate results
- Perform stat. analysis; plot graphs

The Collective Knowledge project (2015-cur)

1) GitHub repo or archive file

/dataset/images/1.png

/program/detect-edges/program.cpp
Makefile
run.sh
check-output.sh
autotune.sh

/paper/report/pldi.tex

2) User home directory

$HOME/project/2000-forgot-everything/

/dataset/images-2000/2.png

/program/crazy-algorithm/source.cpp
build.bat

/experiment/autotuning/many.logs
/lots-of-stats.sql

/paper/asplos/source.tex

3) Jupyter/colab notebook

```
import matplotlib.pyplot as plt
import pandas
import numpy
...

image='/home/fursin/project/2000-forgot-everything/dataset/images-2000/2.png'

features=get_features(image)
```

4) Docker image

Install stable OS and packages
Set environment

Run ad-hoc program scripts

Somehow move raw results out of the image for further analysis

Use ad-hoc analysis scripts outside or inside Docker

All these problems motivated me to start the Collective Knowledge project:
cKnowledge.org github.com/ctuning/ck

The key concept is to convert all software projects into a unified database of reusable components (algorithms, packages, datasets, models, scripts, papers, results...) with a common API, CLI, JSON meta descriptions, and reusable automation actions.

Collaboratively automate painful and repetitive tasks in ML&systems R&D.

Gradually extend common APIs and meta descriptions of all components.

cTuning.org/ae (2014-cur): what I've noticed when reproducing 150+ papers at ML&systems conferences

1) GitHub repo or archive file

.ckr.json

/.cm/alias-a-dataset
/.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

Collective Knowledge COMPATIBLE

2) User home directory

$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

Collective Knowledge COMPATIBLE

```
$ 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:crazy-algorithm
```

3) Jupyter/colab notebook

```
import matplotlib.pyplot as plt
import pandas
import numpy
...
import ck.kernel as ck

# We can now access all our software projects as a database
r=ck.access({'action':'search',
             'module_uoa':'dataset',
             'add_meta':'yes'})
if r['return']>0: ck.err(r)
list_of_all_ck_entries=r['lst']

for ck_entry in list_of_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_uoa':'dataset',
                        'image':image})
    if features['return']>0: ck.err(features)
```

Collective Knowledge COMPATIBLE

```
$ ck compile program:detect-edges --speed
  Detecting compilers on your system…
  1) LLVM 10.0.1
  2) GCC 8.1
  3) GCC 9.3
  4) ICC 19.1
```

4) Docker image

Install stable OS and packages
Set environment

Use familiar CK API/CLI to run experiments inside or outside your VM

Move data outside VM in the CK format to continue processing it via CK!

Collective Knowledge COMPATIBLE

CK uses wrappers and JSON meta-descriptions around existing objects to ensure their compatibility

```
$ ck run program:detect-edges
  Searching for datasets ...

  Select dataset:
   1) images
   2) images-2000

$ ck autotune program:detect-edges
...
$ ck reproduce experiment:autotuning
...
```

cTuning.org/ae (2014-cur): what I've noticed when reproducing 150+ papers at ML&systems conferences

1) GitHub repo or archive file

.ckr.json

/.cm/alias-a-dataset
/.cm/alias-a-program
/.cm/alias-a-paper
/.cm/alias-a-module

/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

Collective Knowledge COMPATIBLE

2) User home directory

$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

Collective Knowledge COMPATIBLE

```
$ 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:crazy-algorithm
```

3) Jupyter/colab notebook

Collective Knowledge COMPATIBLE

```
import matplotlib.pyplot as plt
import pandas
import numpy
...
import ck.kernel as ck

# We can now access all our software projects as a database
r=ck.access({'action':'search',
             'module_uoa':'dataset',
             'add_meta':'yes'})
if r['return']>0: ck.err(r)
list_of_all_ck_entries=r['lst']

for ck_entry in list_of_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_uoa':'dataset',
                        'image':image})
    if features['return']>0: ck.err(features)
```

```
$ ck compile program:detect-edges --speed
   Detecting compilers on your system…
   1) LLVM 10.0.1
   2) GCC 8.1
   3) GCC 9.3
   4) ICC 19.1
```

4) Docker image

Install stable OS and packages
Set environment

Use familiar CK API/CLI to run experiments inside or outside your VM

Move data outside VM in the CK format to continue processing it via CK!

Collective Knowledge COMPATIBLE

CK uses wrappers and JSON meta-descriptions around existing objects to ensure their compatibility

CK uses Python modules with JSON I/O to implement common automation actions for objects

```
$ ck run program:detect-edges
   Searching for datasets ...

   Select dataset:
    1) images
    2) images-2000

$ ck autotune program:detect-edges
...
$ ck reproduce experiment:autotuning
...
```

CK bottom-up approach to gradually solve reproducibility issues in ML&systems R&D

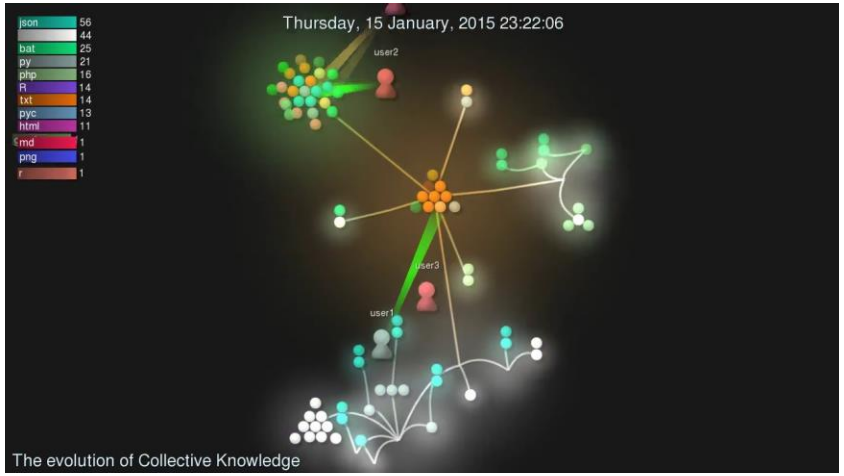
My concern: different conferences, organizations, and projects want to come up with their own common format, framework, and SPECS to share artifacts and workflows along with research projects and papers. However, the main difficulty is how to adapt them to continuously changing software, hardware, models, and datasets!

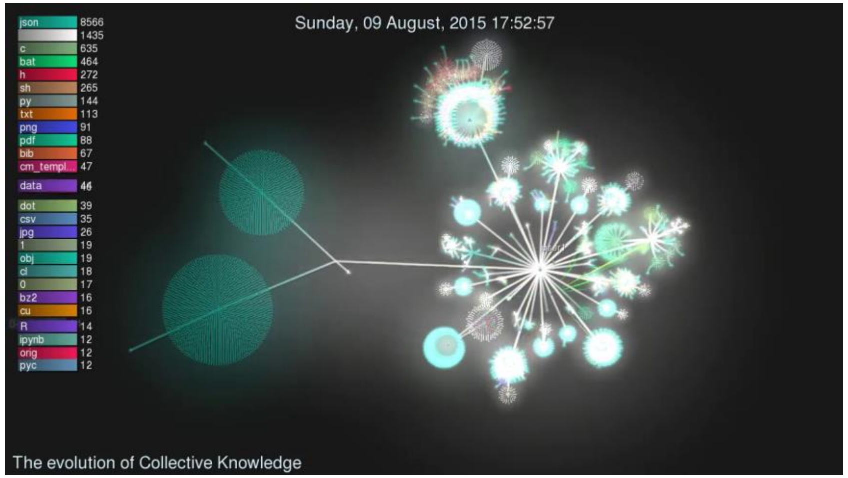
CK concept of evolution and natural selection: provide a very flexible plugin framework to help researchers, practitioners, and students quickly prototype and share simple and reusable automation actions with a Python API, CLI and JSON meta descriptions for typical, repetitive, and painful R&D tasks.

There can be multiple implementations of the same task from different research groups –
they can co-exist until potential convergence thus solving backward compatibility issues in research projects!

Do not enforce SPECs at the beginning – let the community define it through experimentation and DevOps!

Use CK actions to abstract and interconnect existing tools and data rarther than substituting them!





The evolution of CK automation actions from just a few in 2015 to 600+ in 2020: youtu.be/nabXHyot5is

The latest Collective Knowledge graph: cKnowledge.io/kg1

First, we started automating and unifying the most basic and repetitive tasks in ML&systems R&D

1) **Describe different operating systems** **85+ OS descriptions (Linux, Android, Windows, MacOS)**

ck pull repo:ck-env
ck ls os
ck load os:linux-64 --min

2) **Detect and unify information about platforms**

ck detect platform **--help**
ck detect platform --out=json
ck load os:linux-64 --min

We implemented and shared CK components with automation actions to support the real use-cases from our partners, collaborators, and users: cKnowledge.org/partners

3) **Detect installed software** (code, data, models, scripts) **250+ software detection plugins**

ck search soft --tags=dataset
ck detect soft:compiler.llvm

ck show env --tags=llvm

4) **Install missing packages** (code, datasets, models, scripts) **600+ shared packages**

ck search package --tags=dataset,imagenet
ck install package --tags=dataset,imagenet,2012,min

ck show env --tags=dataset
ck virtual env –tags=dataset,imagenet

Anyone can reuse such automation actions to adapt experiments to any platform and environment while using containers to make stable snapshots

github.com/ctuning/ck/wiki/Portable-workflows github.com/ctuning/ck-env

Artifact automated and reusable | Collective Knowledge COMPATIBLE | Workflow CK

The unified CK API allows to apply DevOps principles and Continuous Integration to CK components

CK: small python library (~200Kb); any python and git; Linux/Win/MacOS

```
$ pip install ck
$ ck pull repo:ck-crowdtuning
$ ck {function} {module name}:{data name} @input.json
```

JSON input → Python module **"program"** with functions: **compile and run** → *JSON output*

JSON input → Python module **"soft"** with function: **detect** → *JSON output*

JSON input → Python module **"dataset"** with function: **extract_features** → *JSON output*

JSON input → Python module **"experiment"** with function: **add, get, analyze** → *JSON output*

data UID and alias

- **image corner detection**
- **matmul OpenCL**
- **compression**
- **neural network CUDA**

meta.json

- **GCC V9.3**
- **LLVM V10.1**
- **Intel Compilers 2020**

meta.json

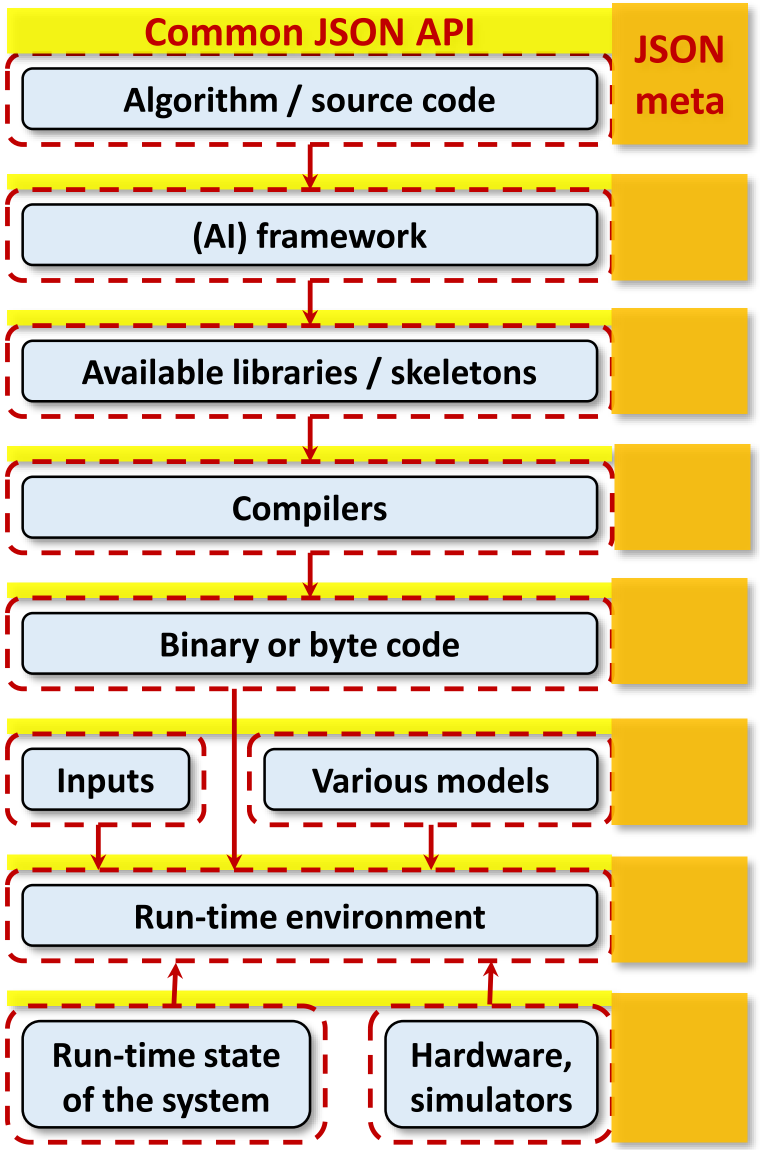
- **image-jpeg-0001**
- **bzip2-0006**
- **txt-0012**
- **video-raw-1280x1024**

meta.json

- **cvs speedups**
- **txt hardware counters**
- **xls table with graph**

meta.json

CK automation actions can be connected into portable workflows



I have re-implemented the MILEPOST/cTuning infrastructure
as a universal CK program workflow with reusable CK components
to compile, run, profile and autotune applications
across diverse data sets and platforms,
validate output for correctness, record and reply experiments,
and visualize autotuning results

github.com/ctuning/ck-autotuning and github.com/ctuning/ck-analytics

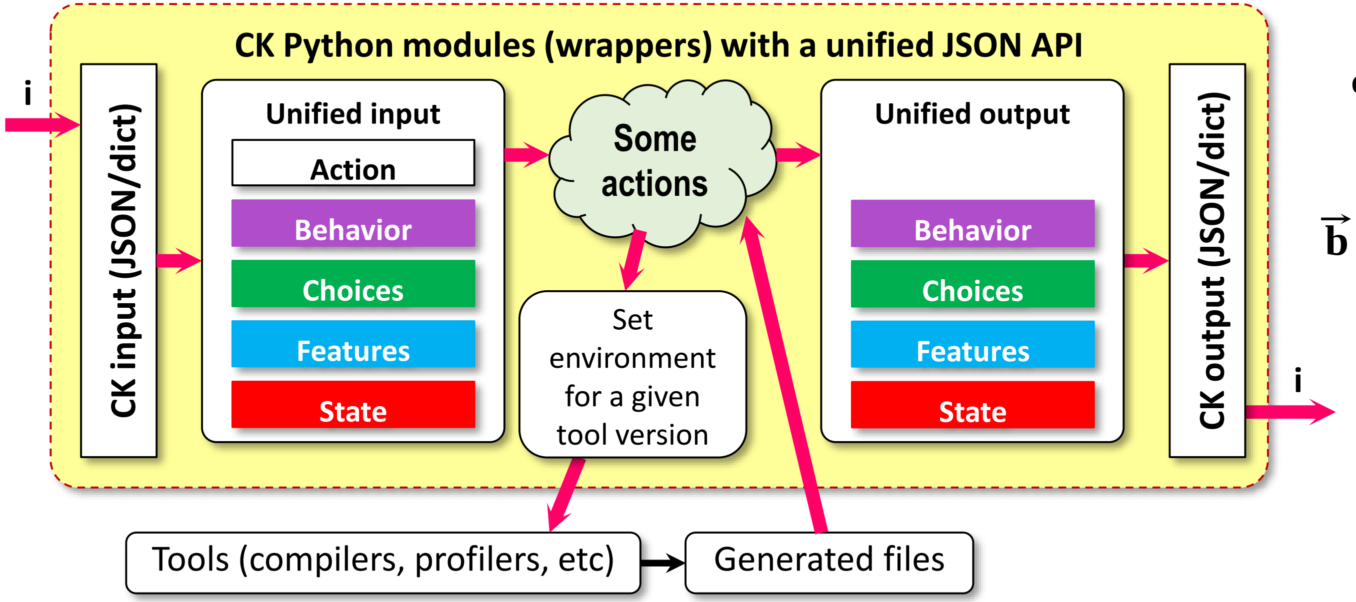
```
$ ck pull repo:ck-crowdtuning

$ ck ls program
$ ck ls dataset

$ ck load program:cbench-automotive-susan  --min
$ ck compile program:cbench-automotive-susan –fast

$ ck run program:cbench-automotive-susan

$ ck autotune program:cbench-automotive-susan

$ ck crowdtune program:cbench-automotive-susan

$ ck replay experiment
```

CK workflows describe dependencies on CK soft detection plugins and packages
to automatically adapt to a given platform and environment

https://cknowledge.io/solution/demo-obj-detection-coco-tf-cpu-benchmark-linux-portable-workflows/#dependencies

We collaborated with the Raspberry Pi foundation to reproduce MILEPOST results via CK framework

First expose coarse grain high-level choices, features, system state and behavior characteristics via CK APIs

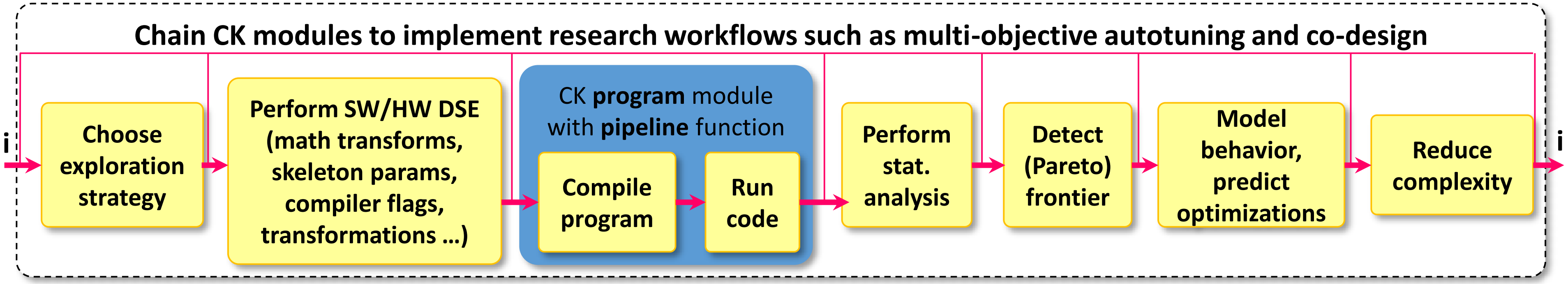


Formalized function B of a behavior of any CK object

$$\vec{b} = B(\vec{c}, \vec{f}, \vec{s})$$

Flattened CK JSON vectors
(dict converted to vector)
to simplify statistical analysis, machine learning and data mining

Then automate crowd-benchmarking and optimization across diverse models, datasets and platforms

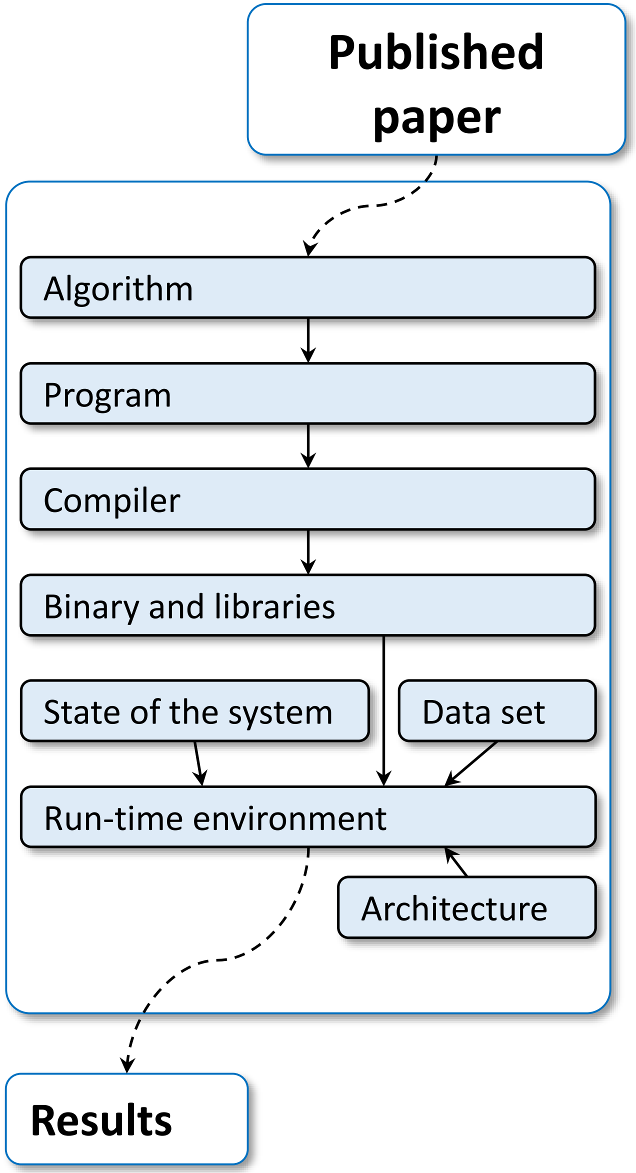


Keep best species (AI/SW/HW choices); model behavior; predict better optimizations and designs

Reproducible article generated by CK: cKnowledge.org/rpi-crowd-tuning
CK-compatible repository: github.com/ctuning/reproduce-milepost-project

I managed to introduce the Artifact Appendix and Reproducibility Checklist at ACM conferences

My goal was to start unifing the Artifact Evaluation process in such a way that it can be later automated using CK actions: cTuning.org/ae/submission_extra.html



1. **Abstract**
2. **Check-list**
3. **How delivered?**
4. **Software dependencies**
5. **Hardware dependencies**
6. **Data sets**
7. **Installation**
8. **Experiment workflow**
9. **Evaluation and expected result**
10. **Notes**

Algorithm
Program
Compilation
Transformations
Binary
Data set
Run-time environment
Hardware
Run-time state
Execution
Output
Experiment workflow
Publicly available?

Can use collected templates to derive high-level meta description of an artifact pack

2017-2018: ACM ASPLOS-REQUEST tournament to co-design Pareto-efficient SW/HW stacks for ML/AI

AI/ML/SW/HW
co-design

AI hardware
Google, NVIDIA, ARM, Intel, IBM, Qualcomm, Apple, AMD …

AI models
Google, OpenAI, Microsoft, Facebook …

AI software
- AI frameworks (TensorFlow, MXNet, PyTorch, CNTK, Theano)
- AI libraries (cuDNN, libDNN, ArmCL, OpenBLAS)

AI integration/services
AWS, Google, Azure ...

cKnowledge.org/request

Finding the most efficient AI/SW/HW stacks
across diverse models, data sets and platforms
via open competitions,
share them as reusable CK components
and visualize on a public scoreboard

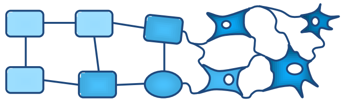
Organizers (A-Z)

Luis Ceze, University of Washington
Natalie Enright Jerger, University of Toronto
Babak Falsafi, EPFL
Grigori Fursin, cTuning foundation
Anton Lokhmotov, dividiti
Thierry Moreau, University of Washington
Adrian Sampson, Cornell University
Phillip Stanley Marbell, University of Cambridge

Real use-cases (future)

Healthcare
Agriculture
Finances
Automotive
Aerospace
Meteorology
Retail
Robotics
…

CK framework to share and reproduce results

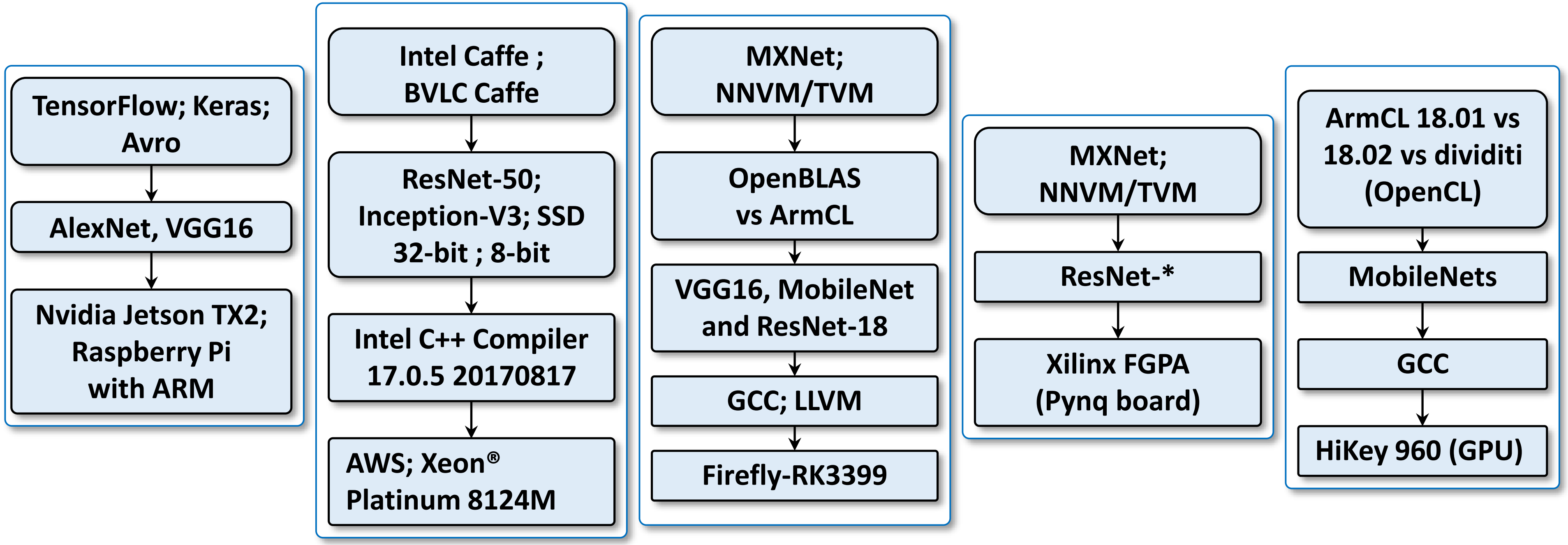


Interdisciplinary community

We wanted to check if CK can support collaborative and reproducible AI/ML/SW/HW autotuning and co-design process, and decided to organize an open tournament at ASPLOS.

We reused the existing CK program workflow and just slightly customized it for all submissions!

8 intentions to submit and 5 submitted image classification workflows with unified Artifact Appendices



Public validation at github.com/ctuning/ck-request-asplos18-results via GitHub issues.

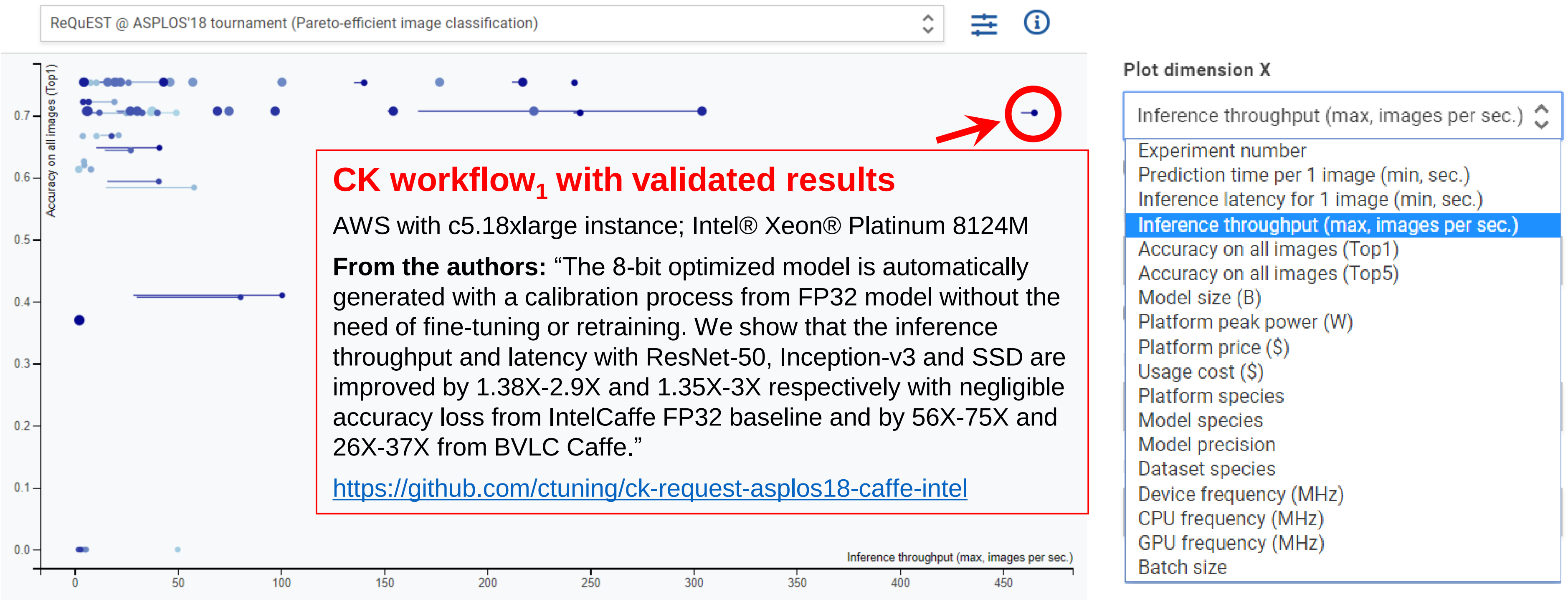
All validated papers are published in the ACM DL
with **portable, customizable and reusable CK components and workflows**:
dl.acm.org/citation.cfm?doid=3229762

See ACM ReQuEST report: portalparts.acm.org/3230000/3229762/fm/frontmatter.pdf

See live scoreboards: cKnowledge.io/reproduced-results

By reusing the same CK program workflow with encoded best practices, we can ensure equivalent comparison

All results from multi-objective AI/ML/SW/HW autotuning are presented on a live scoreboard
and become available for public comparison and further customization, optimization and reuse!

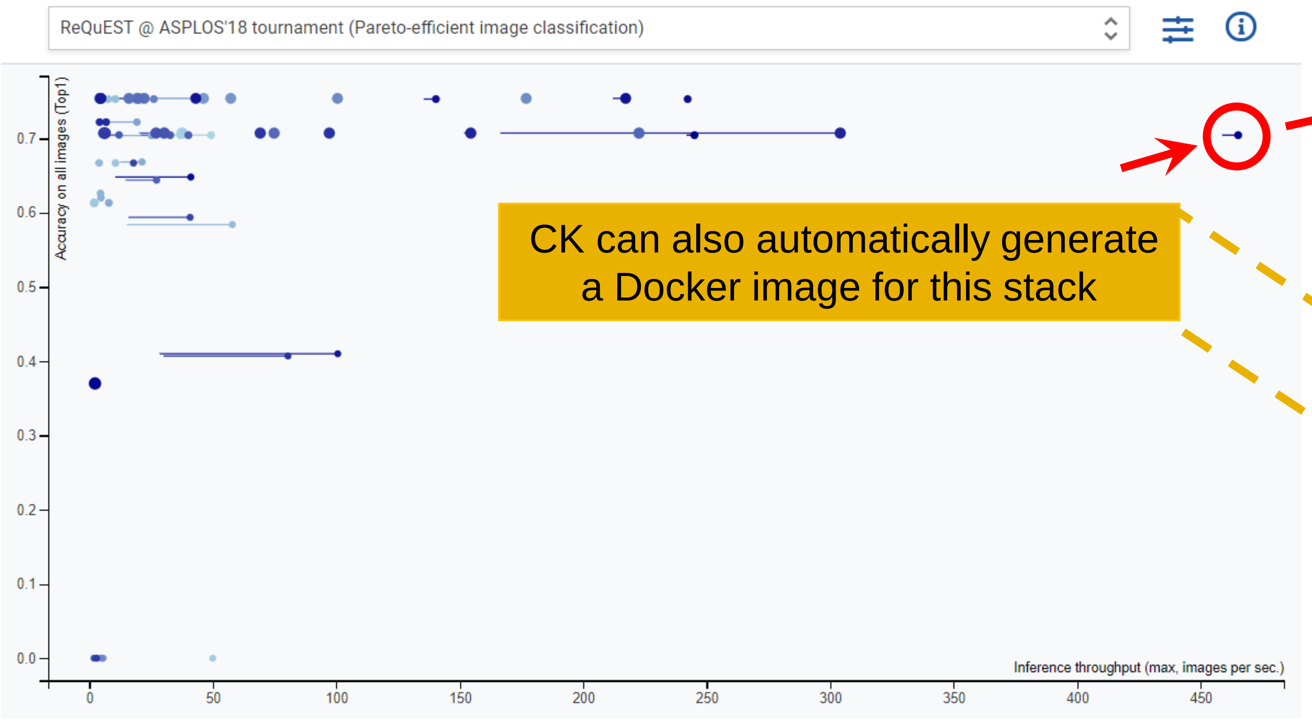


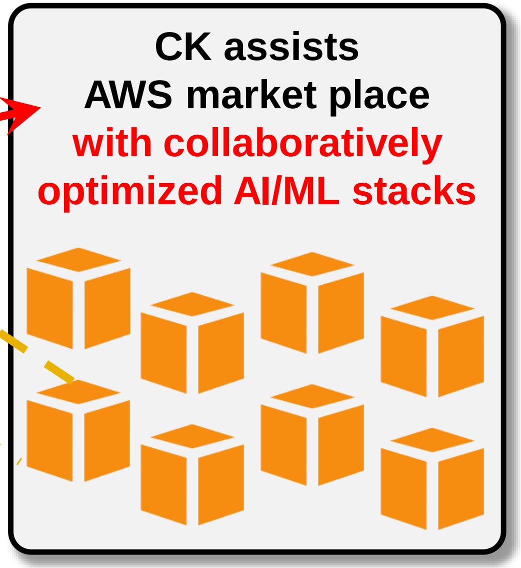
We are not announcing a single winner! We aggregate and show all results:
cKnowledge.io/result/pareto-efficient-ai-co-design-tournament-request-acm-asplos-2018
and let the users select best ML/SW/HW stacks depending on the multiple constraints for their production use!

Such approach is particularly useful for resource-constrained mobile and edge devices (TinyML, IoT)!
See the real-world CK use-cases from General Motors: youtu.be/1ldgVZ64hEI

CK workflows helped other organizations to reproduce the latest ML techniques and deploy them in production!

All results from multi-objective AI/ML/SW/HW autotuning are presented on a live CK scoreboard and become available for public comparison and further customization, optimization and reuse!





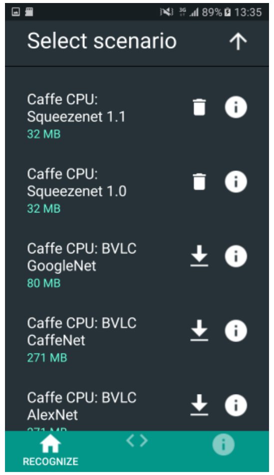
CK can accelerate technology transfer: companies can validate published techniques in their production environment using shared CK workflows!

Researchers and students can reuse published workflows, extend them, and build upon them!

See our joint presentation with Amazon at O'Reilly Intel AI conference:
conferences.oreilly.com/artificial-intelligence/ai-eu-2018/public/schedule/detail/71549.html

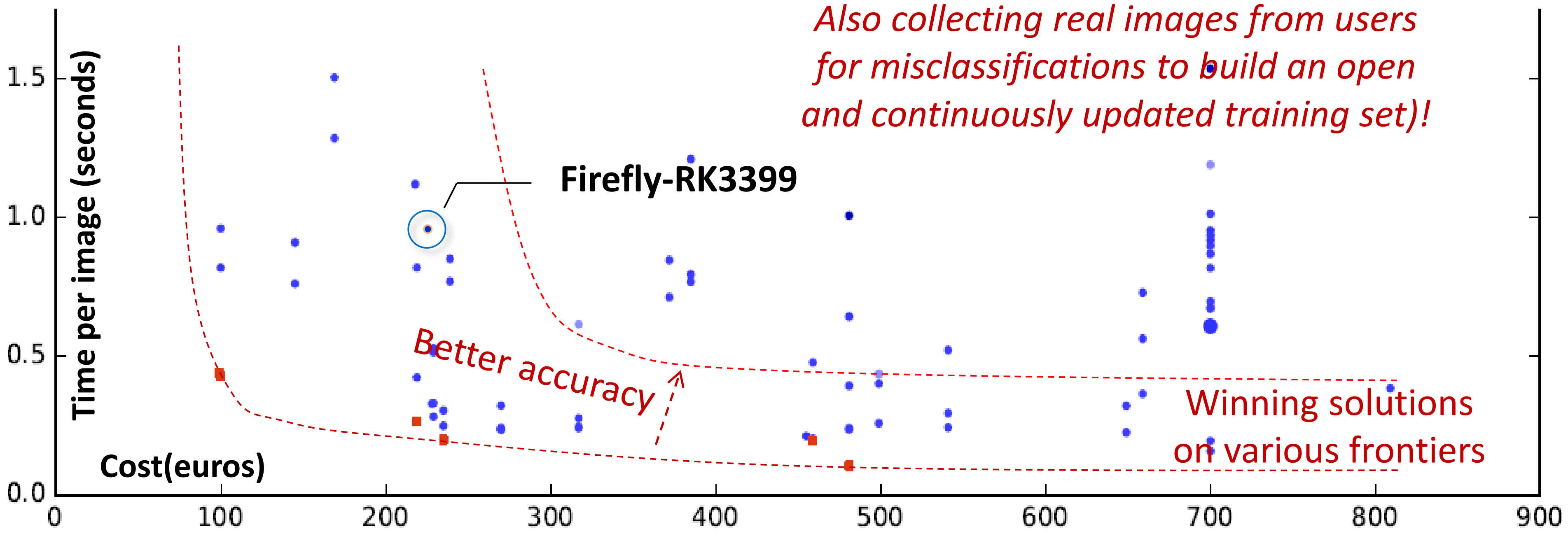
We managed to reuse portable CK program workflow to crowdsource AI/ML benchmarking across Android devices!

cKnowledge.org/android-demo.html



The number of distinct participated platforms: **800+**
The number of distinct CPUs: **260+**
The number of distinct GPUs: **110+**
The number of distinct OS: **280+**
Power range: **1-10W**

No need for a dedicated and expensive cloud – volunteers help us validate research ideas similar to SETI@HOME



Continuously collect statistics, bugs, and misclassifications at cKnowledge.org/repo-beta

We then crowdsourced the long BLAS autotuning process on Firefly-RK3399

Expose tunable parameters of OpenCL-based BLAS (github.com/CNugteren/CLBlast) via CK program workflow.
Take two data sets (small & large) as CK packages.
Add extra constraints on co-design space to avoid illegal combinations.

Name	Description	Ranges
KWG	2D tiling at workgroup level	{32,64}
KWI	KWG kernel-loop can be unrolled by a factor KWI	{1}
MDIMA	Local Memory Re-shape	{4,8}
MDIMC	Local Memory Re-shape	{8, 16, 32}
MWG	2D tiling at workgroup level	{32, 64, 128}
NDIMB	Local Memory Re-shape	{8, 16, 32}
NDIMC	Local Memory Re-shape	{8, 16, 32}
NWG	2D tiling at workgroup level	{16, 32}
SA	manual caching using the local memory	{0, 1}
SB	manual caching using the local memory	{0, 1}
STRM	Striding within single thread for matrix A and C	{0,1}
STRN	Striding within single thread for matrix B	{0,1}
VWM	Vector width for loading A and C	{8,16}
VWN	Vector width for loading B	{0,1}

Perform systematic exploration of design and optimization spaces using universal CK autotuner: github.com/ctuning/ck-autotuning

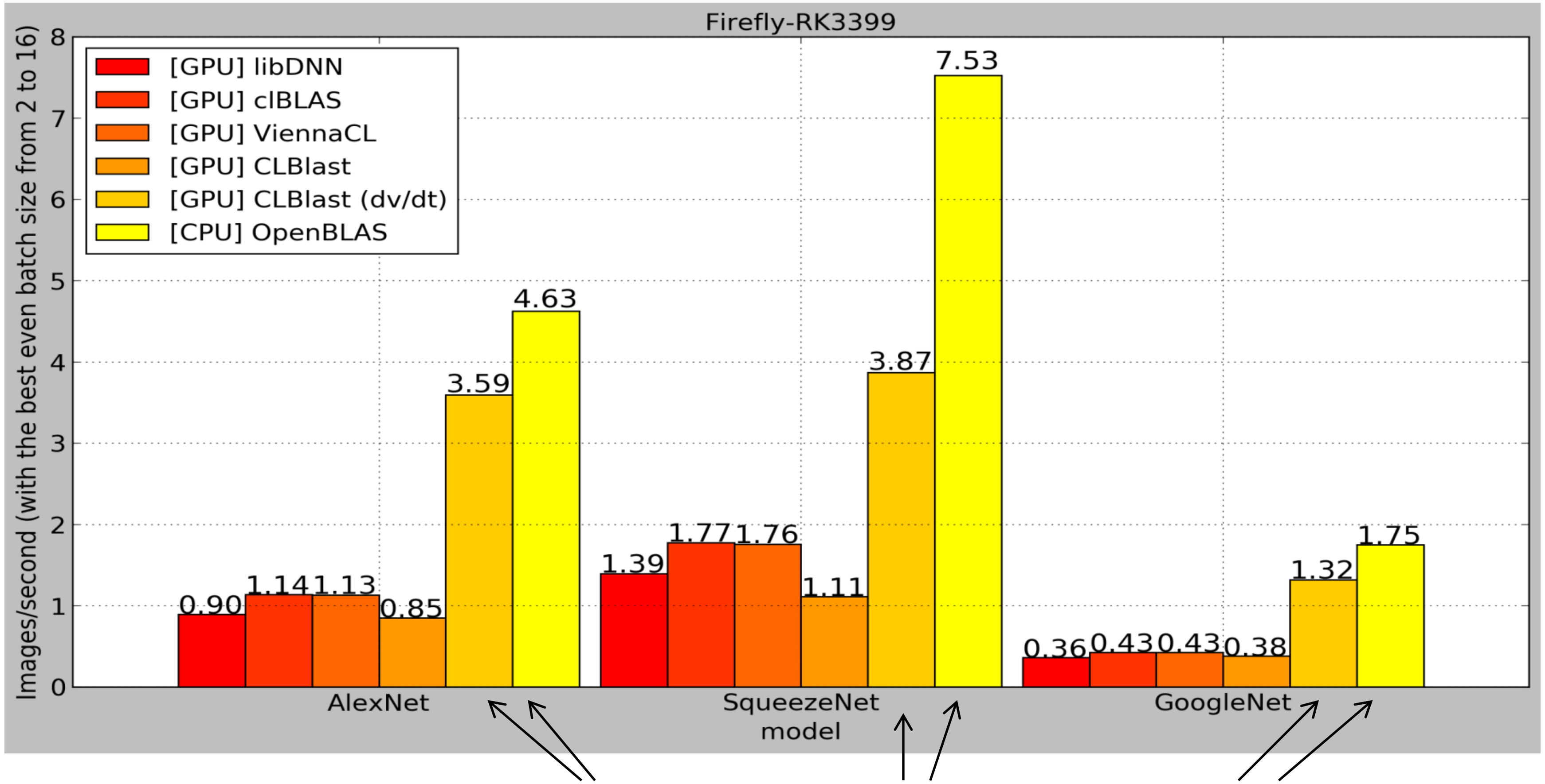
Record all experiments in a reproducible way using CK module “experiment”.

Use different CK autotuning plugins to speed up design space exploration based on probabilistic focused search, generic algorithms,
deep learning, SVM, KNN, MARS, decision trees …

Related paper about the universal CK autotuner: cKnowledge.org/rpi-crowd-tuning

Collaboration between Marco Cianfriglia (Roma Tre University), Cedric Nugteren (TomTom),
Flavio Vella&Anton Lokhmotov (dividiti), and Grigori Fursin (cTuning foundation)

We collected reproducible results from CLBlast in Caffe on Firefly-RK3399 using CK dashboards



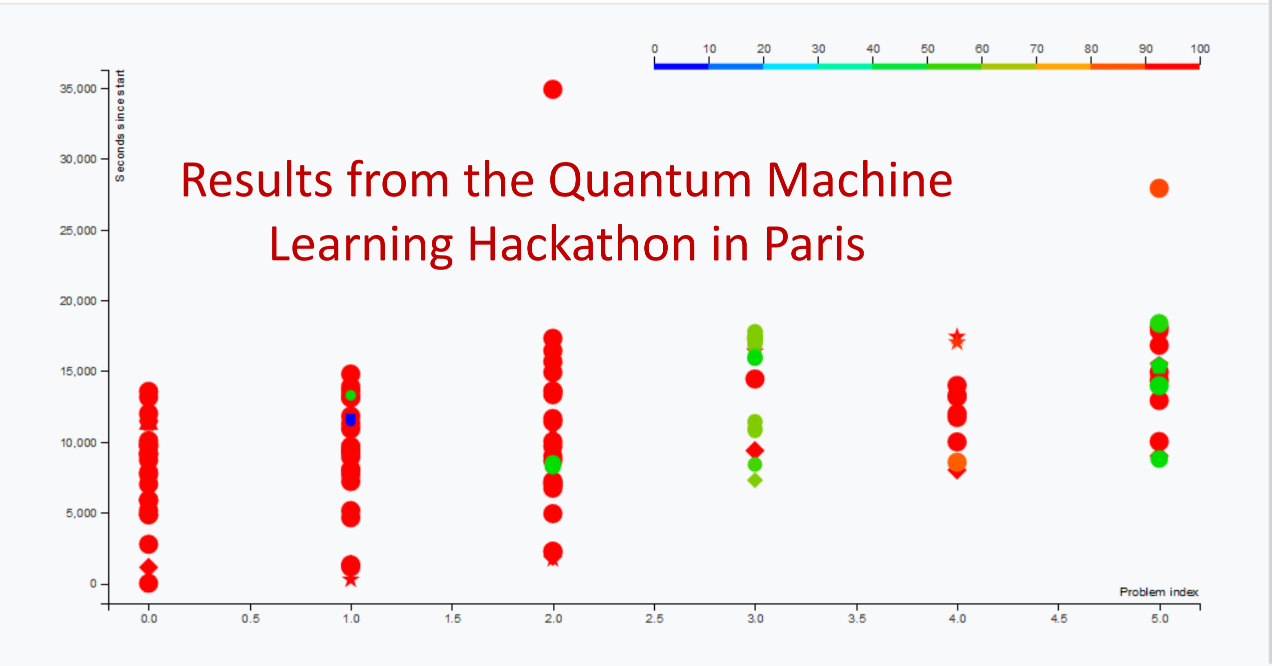
- Caffe with autotuned OpenBLAS (threads and batches) is the fastest
- Caffe with autotuned CLBlast is 6..7x faster than default version and competitive with OpenBLAS-based version– now worth making adaptive selection at run-time.

Sharing results from research projects and along with research papers
in a reproducible way with the community for further validation and improvement:
nbviewer.jupyter.org/github/dividiti/ck-caffe-firefly-rk3399/blob/master/script/batch_size-libs-models/analysis.20170531.ipynb

We even managed to reuse CK program workflow to automate quantum machine learning experiments!

cKnowledge.org/quantum - Quantum Collective Knowledge workflows (QCK) support reproducible hackathons, and help researchers share, compare and optimize different algorithms across conventional and quantum platforms

cKnowledge.io/reproduced-results



#	Problem index	Timestamp (UTC)	Team name	Training time (sec)	Training accuracy	Test accuracy	Solution's rank	Source code	Quantum circuit
#1	4	Sun Jan 27 12:19:42 2019	Optimize, adapt, overcome	47.20	100.0	100.0	1	continuous_solver	Show circuit
#2	4	Sun Jan 27 12:52:49 2019	prevision.io	80.68	100.0	100.0	2	continuous_solver	Show circuit
#3	4	Sun Jan 27 13:21:31 2019	rebecca	171.54	100.0	100.0	3	continuous_solver	Show circuit

RIVERLANE

$\frac{d\vec{v}}{dt}$

IBM

rigetti

ThoughtWorks®

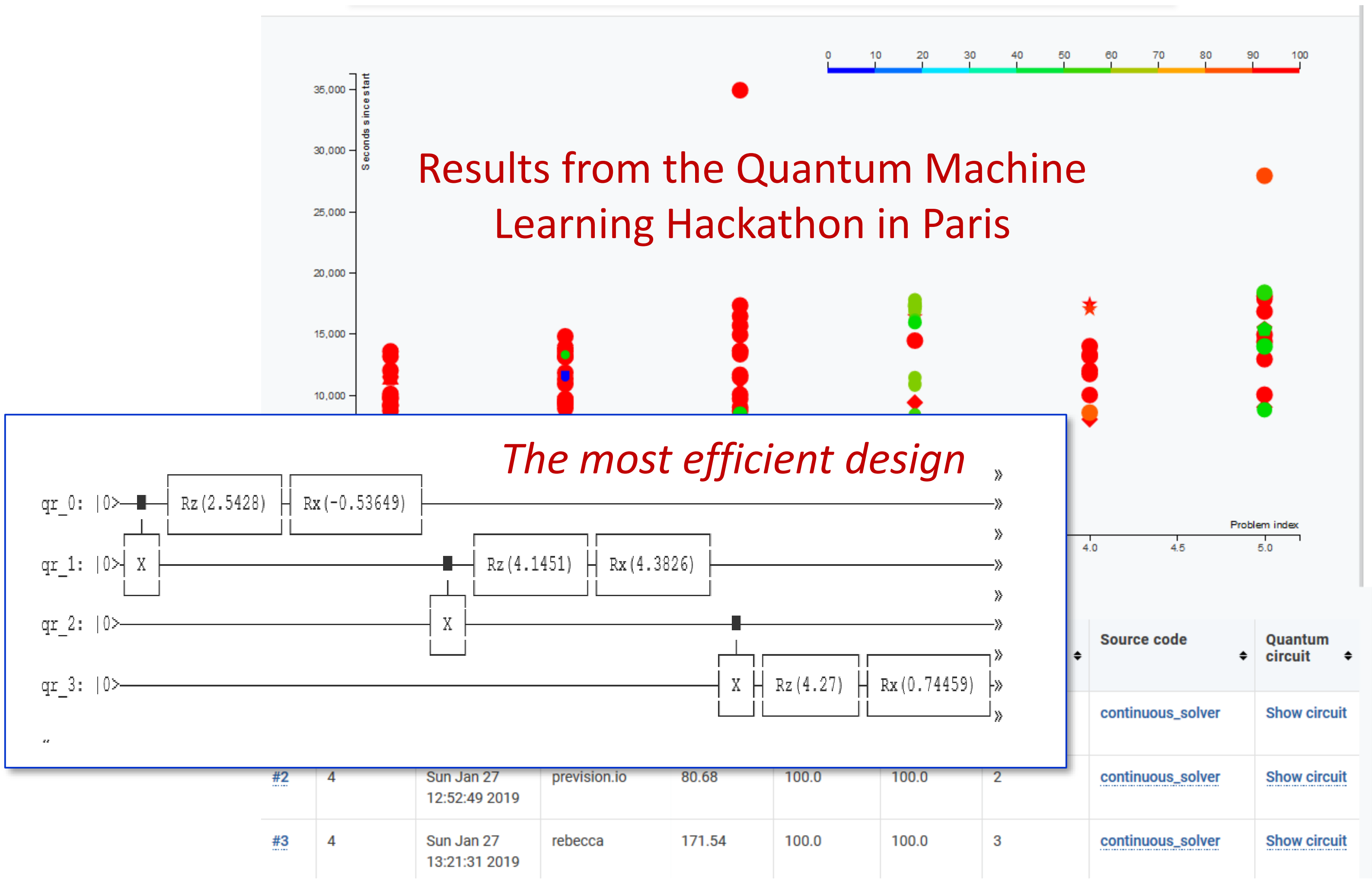
QUANTO NATION

Innovate UK

We even managed to reuse CK program workflow to automate quantum machine learning experiments!

cKnowledge.org/quantum - Quantum Collective Knowledge workflows (QCK) support reproducible hackathons, and help researchers share, compare and optimize different algorithms across conventional and quantum platforms

IBM blog about CK: linkedin.com/pulse/reproducing-quantum-results-from-nature-how-hard-could-lickorish



RIVERLANE

$\frac{d\vec{v}}{dt}$

IBM

rigetti

ThoughtWorks®

QUANTO NATION

Innovate UK

We have joined the MLPerf consortium while participating in the best practices group

MLPerf

A broad ML benchmark suite for measuring performance
of ML software frameworks,
ML hardware accelerators, and ML cloud platforms.

mlperf.org

A very important and timely initiative developing best practices, rules, and tools for fair ML&systems benchmarking!

MLBox: a related project to describe and pack ML models in a reproducible format:
https://github.com/mlperf/mlbox
We plan to connect CK and MLBox to support SW/HW customization and portability.

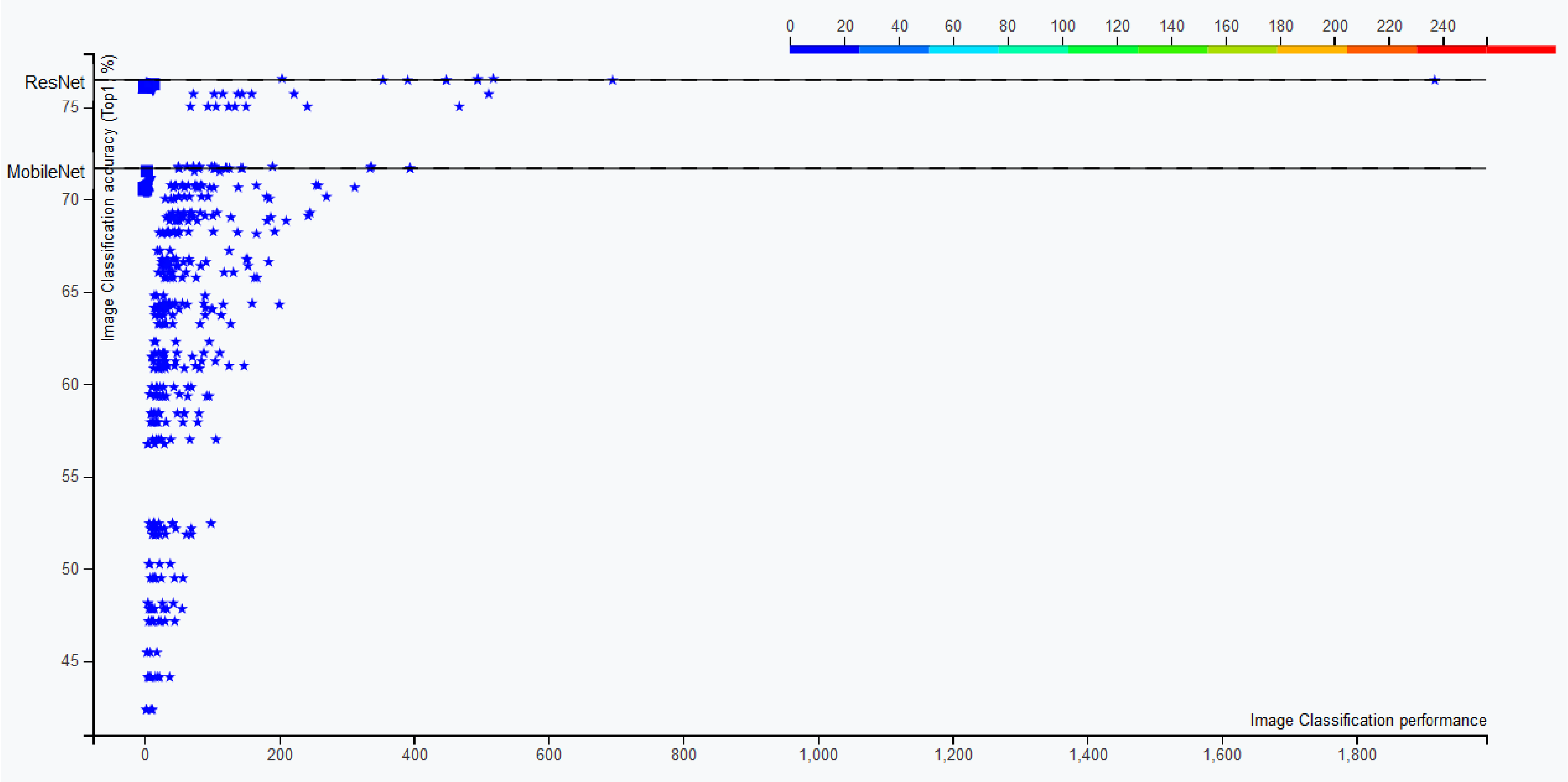
Our collaborators from dividiti reused the portable CK program workflow
and added extra automation actions and components
to make it easier to submit MLPerf results for hardware vendors:
github.com/ctuning/ck-mlperf

We continue developing an open platform to automate ML/SW/HW co-design,
visualize, compare, and reproduce results,
and make it easier to create MLPerf-like workflows:
https://cKnowledge.io/test

As a proof-of-concept, dividiti used CK to submit benchmarking results to MLPerf inference v0.5 open division

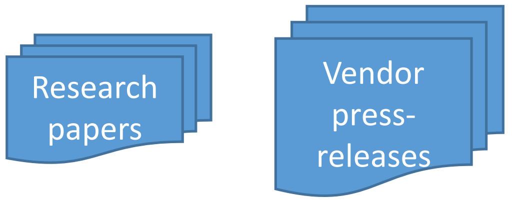
Over 500 validated [inference v0.5 benchmarking results] were submited from 14 organizations (including Dell EMC, Nvidia, Google, Intel, Alibaba, Habana) measuring how fast and how well a pre-trained computer system can classify images, detect objects, and translate sentences.

Over 400 of these results were automated with the CK framework and reusable CK program workflow.



See cKnowledge.io/reproduced-results and cKnowledge.io/test to try yourself.

July 2020: I have finished prototyping cKnowledge.io to organize all CK components and workflows in one place

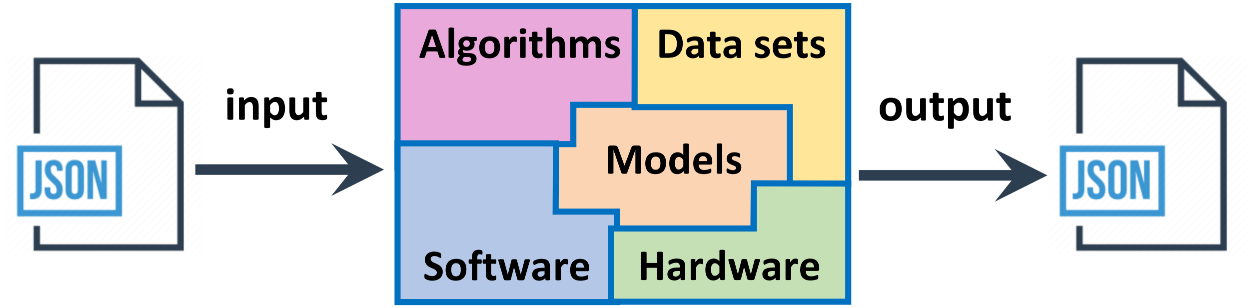


Collected 50K+ descriptions of AI/ML/SW/HW components and workflows in the CK format: cKnowledge.io/browse

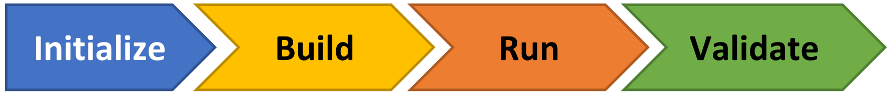
CK JSON API	CK JSON API	CK JSON API	CK JSON API	CK JSON API
Algorithms	**Models**	**Software**	**Data sets**	**Hardware**
Object detection	MobileNets	TensorFlow	ImageNet	CPU
Object classification	ResNet	Caffe	KITTI	GPU
Speech recognition	VGG	MXNet	VOC	TPU
…	…	…	…	…
Training/inference	SqueezeDet	PyTorch	Real data sets	NN accelerators
…	…	…	…	…

It is possible to share portable and customizable workflows along with research papers

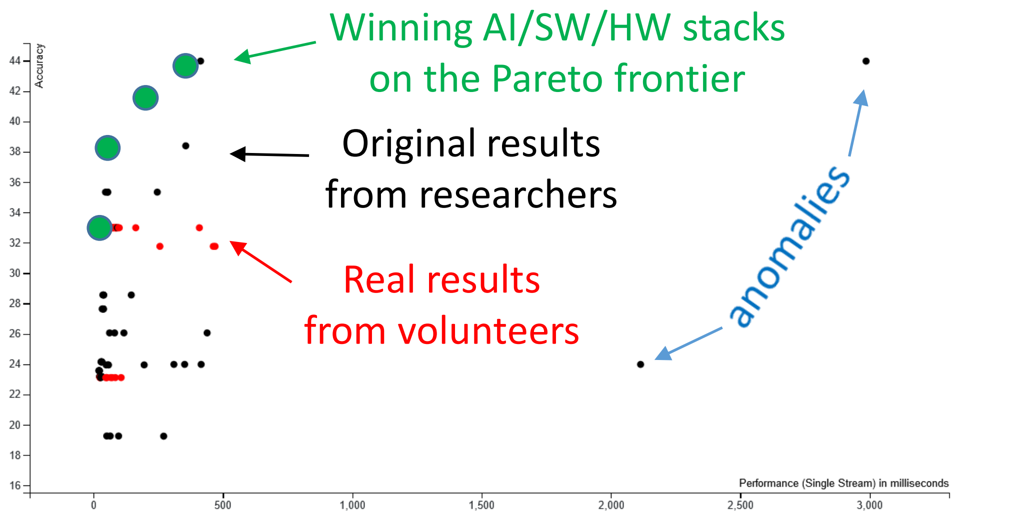
AI/ML solutions with portable workflows, unified API, and JSON input/output



CK client (cBench) helps end-users to run AI/ML solutions across diverse devices, software, models and data sets, and share results



Multiple online scoreboards to reproduce and compare results from AI, ML and systems papers across heterogeneous platforms (Arm, Nvidia, Intel…), frameworks (TF, PyTorch, MXNet), models, and data sets, and highlight the winners (speed, accuracy, costs,...): cKnowledge.io/all-results



Conclusions and the current state

The Collective Knowledge framework (github.com/ctuning/ck) provides a common API to all software projects together with a database-like control and reusable automation actions for their individual components (algorithms, packages, data sets, models, scripts, results). The goal is to make it easier for researchers, practitioners, and students to reuse best R&D practices and artifacts, assemble portable workflows, reproduce and compare research techniques, build upon them, and use them in production.

The Collective Knowledge platform (cKnowledge.io) helps to organize AI, ML, and systems knowledge in the form of portable CK workflows, automation actions, and reusable artifacts. The goal is to make it easier to find, test, and adopt innovative technology in the real world. Our platform is also used to automatically co-design efficient AI/ML/SW/HW stacks in terms of speed, accuracy, energy, and other costs and accelerate their deployment in production across diverse platforms from data centers and supercomputers to mobile and edge devices.

Very few people believed in 2015 that it was possible to develop portable and reproducible workflows for ML&systems R&D using such an evolutionary approach. However, we have completed the prototyping phase of the Collective Knowledge framework (CK) and successfully validated it in many industrial and academic projects: cKnowledge.org/partners and arxiv.org/abs/2006.07161

We demonstrated that it was possible to use CK to

- share portable CK workflows along with published papers to make it easier to reproduce results and reuse artifacts
- perform universal autotuning of the full AI/ML/SW/HW stack and find best configurations on Pareto frontier
- automate and simplify MLPerf submissions: github.com/ctuning/ck-mlperf
- support reproducible optimization tournaments with live scoreboards: cKnowledge.io/reproduced-results
- use CK as a portable backend for SageMaker, MLFlow, Kedro, and other tools
- enable reproducible and interactive papers continuously updated by the community with new results: cKnowledge.org/rpi-crowd-tuning

HUGE THANKS TO ALL OUR PARTNERS, COLLABORATORS, AND USERS: cKnowledge.org/partners

Future work: many possible directions

Even though the CK technology is already used in production, it is still a proof-of-concept. I now brainstorm the CK2 project to "democratize" this technology, make it easier to use, and make ML&systems R&D more portable and reproducible:

- Standardize and document all CK components, workflows, automation actions, and meta descriptions (on-going work)
- Provide a convenient GUI to add new components, workflows, and scoreboards at cKnowdege.io
- Connect CK with MLBox to automate and simplify MLPerf
- Add more packages and software detection plugins for all main ML models, datasets, frameworks, libraries, compilers, and hardware
- Use CK as a portable backend for SageMaker, MLFlow, Kedro, and other tools
- Enable auto-generated, reproducible and reusable research papers
- Support lifelong ML&systems crowd-tuning to enable efficient, reliable, and affordable computing everywhere (my long-term goal)

I feel that I have completed my mission to make ML&systems R&D more reproducible, reusable and trustable!

I now plan to come back to R&D on lifelong ML&systems optimization with the help of the cKnowledge.io platform particularly focusing on TinyML and ML/SW/HW crowd-tuning for edge devices.

Get in touch if you are interested to discuss CK, portable AI/ML workflows, ML/SW/HW co-design, cKnowledge.io platform, and new projects:

- LinkedIn: linkedin.com/in/grigorifursin
- Web page: cKnowledge.io/@gfursin
- Emails: Grigori.Fursin@cTuning.org