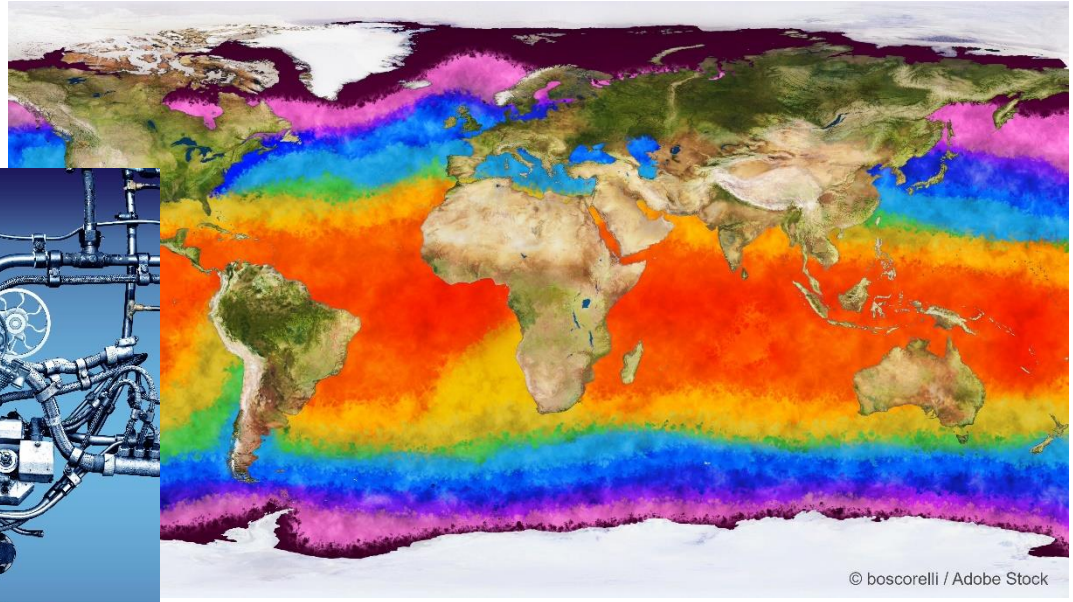
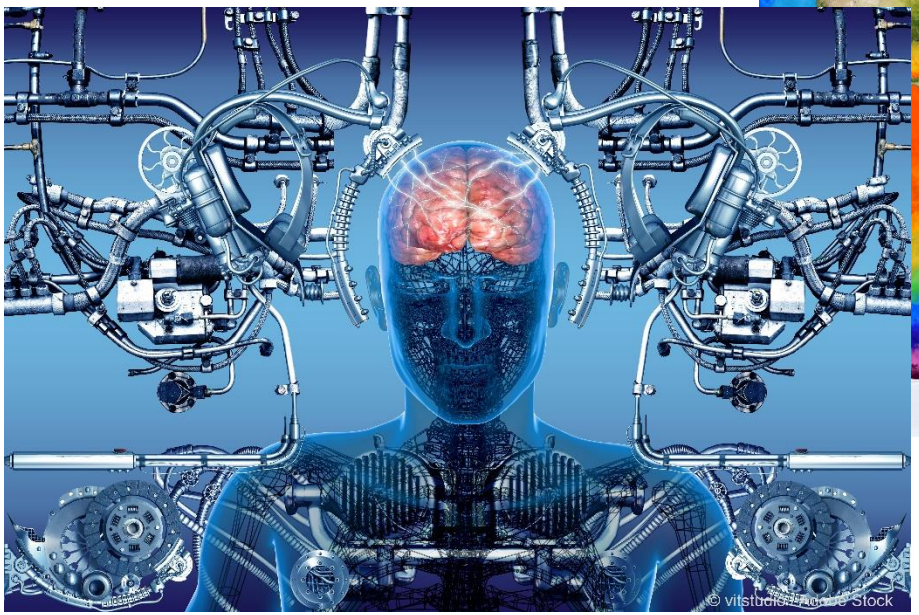
Collective Knowledge framework to automate, crowdsource, reproduce and share HPC experiments





Grigori Fursin (@grigori_fursin)

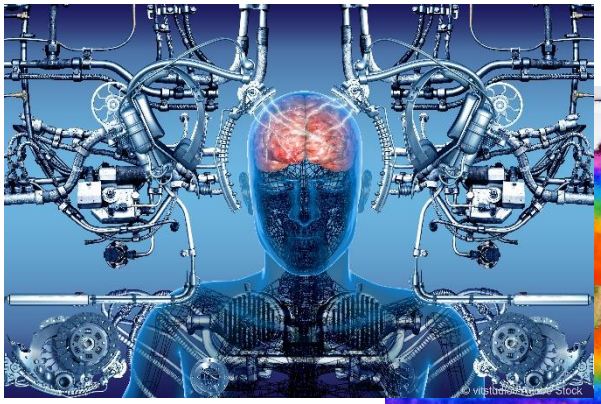
Founder and CEO, non-profit cTuning foundation, France

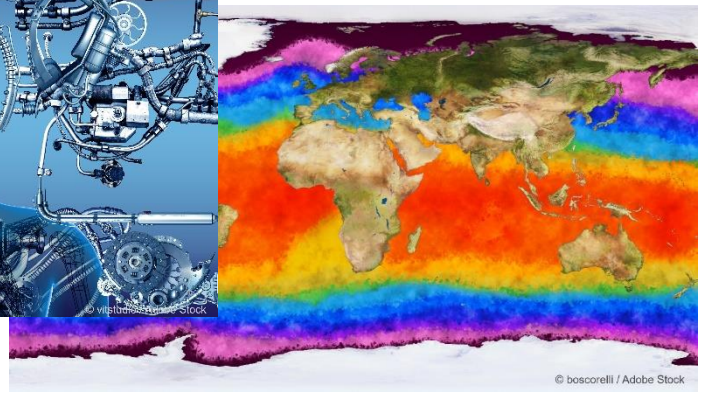
Co-founder and CTO, dividiti, UK

AI, ML, systems and quantum research is booming - 1000+ papers every year …

Applications

- Meteorology
- Health
- Robotics
- Automotive
- Economics
- Physics
- Astronomy
- Education





Platforms

- HPC
- Desktops
- IoT
- Mobile
- Cloud sevices

Many great tools, data sets and models to help researchers …

Applications
- Meteorology
- Health
- Robotics
- Automotive
- Economics
- Physics
- Astronomy
- Education

Scientific tools
- MATLAB
- Scilab
- Simulink
- LabVIEW
- Gnuplot
- LaTeX
- Ipython

Build tools
- Make
- Cmake
- SCons
- Bazel
- Gradle
- Ninja

Languages
- C++
- C#
- C
- Go
- PHP
- Fortran
- Java
- Python

Compilers
- LLVM
- GCC
- Intel
- PGI
- TVM
- CUDA

DevOps tools
- Git
- Jenkins
- Docker
- Kubernetes
- Singularity

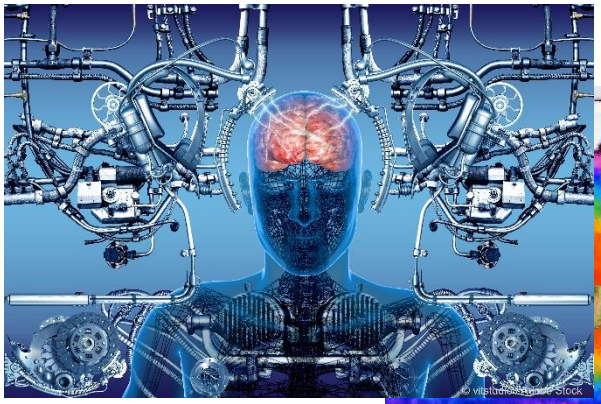
Package managers
- Anaconda
- Go
- Npm
- Pip
- Sbt
- dpkg
- Spack
- EasyBuild

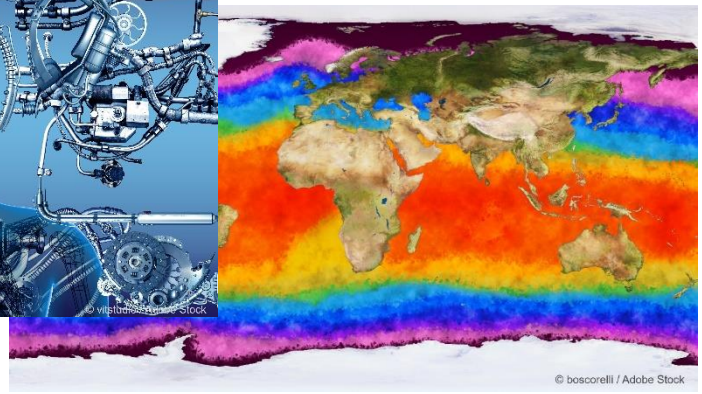
Libraries
- SciPy
- TFLite
- OpenBLAS
- MAGMA
- cuDNN
- cuFFT
- ArmNN
- CLBlast
- gemmlowp
- Boost
- HDF5
- MPI
- OpenCV
- Protobuf

OS
- Linux
- MacOS
- BSD
- Windows
- Android

Shells
- bash
- sh
- csh
- ksh
- Windows shell

Programs
- Image classification
- Object detection
- Natural Language processing
- Text processing
- Video processing
- Personal assistant





Benchmarks
- SPEC
- EEMBC
- HPCG
- LINPACK
- cBench
- MLPerf

Hardware
- CPU
- GPU
- TPU / NN
- DSP
- FPGA
- Quantum
- Simulators
- Interconnects

AI/ML frameworks
- TensorFlow
- PyTorch
- MXNet
- Caffe
- MCT (CNTK)
- Keras
- Kubeflow
- AutoML
- SageMaker
- Apache Spark

Models
- GoogleNet
- AlexNet
- VGG
- ResNet
- MobileNets
- SSD
- SqueezeNet
- DeepSpeech

Datasets
- ImageNet
- KITTI
- COCO
- MiDataSets
- Human Cell Atlas
- 1000 Genomes
- Earth models
- OpenStreetMap

Workload managers
- MPI
- SLURM
- PBS
- FLUX

Web services
- GitHub
- GitLab
- BitBucket
- Travis
- JupyterHub
- Codelabs
- SageMaker

Databases / experiments
- MySQL
- PostgreSQL
- MongoDB
- CouchDB
- Text files
- JSON files
- XLS files

Knowledge sharing
- ArXiv
- ACM DL
- IEEE DL
- GitHub
- Zenodo
- FigShare
- Web pages

Platforms
- HPC
- Desktops
- IoT
- Mobile
- Cloud sevices

Let's innovate …

Applications
- Meteorology
- Health
- Robotics
- Automotive
- Economics
- Physics
- Astronomy
- Education

Scientific tools
- MATLAB
- Scilab
- Simulink
- LabVIEW
- Gnuplot
- LaTeX
- Ipython

Build tools
- Make
- Cmake
- SCons

Languages
- C++
- C#
- C
- Go

Compilers
- LLVM
- GCC
- Intel
- PGI

DevOps tools
- Git
- Jenkins
- Docker

Package managers
- Anaconda
- Go
- Npm

Libraries
- SciPy
- TFLite
- OpenBLAS
- MAGMA
- cuDNN
- cuFFT
- ArmNN
- CLBlast
- gemmlowp
- Boost
- HDF5
- MPI
- OpenCV
- Protobuf

OS
- Linux
- MacOS
- BSD
- Windows
- Android

Shells
- bash
- sh
- csh
- ksh
- Windows shell

Programs
- Image classification
- Object detection
- Natural Language processing
- Text processing
- Video processing
- Personal assistant


© Elnur / Adobe Stock

Benchmarks
- SPEC
- EEMBC
- HPCG
- LINPACK
- cBench
- MLPerf

Hardware
- CPU
- GPU
- TPU / NN
- DSP
- FPGA
- Quantum
- Simulators
- Interconnects

AI/ML frameworks
- TensorFlow
- PyTorch
- MXNet
- Caffe
- MCT (CNTK)
- Keras
- Kubeflow
- AutoML
- SageMaker
- Apache Spark

Models
- GoogleNet
- AlexNet
- VGG
- ResNet
- MobileNets
- SSD
- SqueezeNet
- DeepSpeech

Knowledge sharing
- ArXiv
- ACM DL
- IEEE DL
- GitHub
- Zenodo
- FigShare
- Web pages

Platforms
- HPC
- Desktops
- IoT
- Mobile
- Cloud sevices

- MiDataSets
- Human Cell Atlas
- 1000 Genomes
- Earth models
- OpenStreetMap

managers
- MPI
- SLURM
- PBS
- FLUX

- GitLab
- BitBucket
- Travis
- JupyterHub
- Codelabs
- SageMaker

- PostgreSQL
- MongoDB
- CouchDB
- Text files
- JSON files
- XLS files

cTuning.org/ae: what I noticed during artifact evaluation at PPoPP, PACT, SC

How the community run, share, reproduce and reuse experiments

- Download an archive or some container with artifacts from an accepted paper (manually)

- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  What if different shell or even OS?

  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    What if some sources are already available?

  - Download dataset(s) from some external sources (often automated unless included)

    What if URL changed or file is not available?

    What if I want to try different datasets or my own ones?

  - Download model(s) from some external sources (often automated unless included)

    What if I want to try different models or my own ones? Will they be compatible?

  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?

  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!

  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?

  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

Let's hide the mess …

How the community run, share, reproduce and reuse experiments

- Download an archive or some container with artifacts from an accepted paper (manually)

- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  What if different shell or even OS?

  - Do … GitLab, BitBucket)

    …dy available?

  - Do … included)

    …ot available?

    …y own ones?

  - Do … included)

    … compatible?

  - Ins … y)

    …y are newer?

    … for my system?

  - Compile progr … "automated")

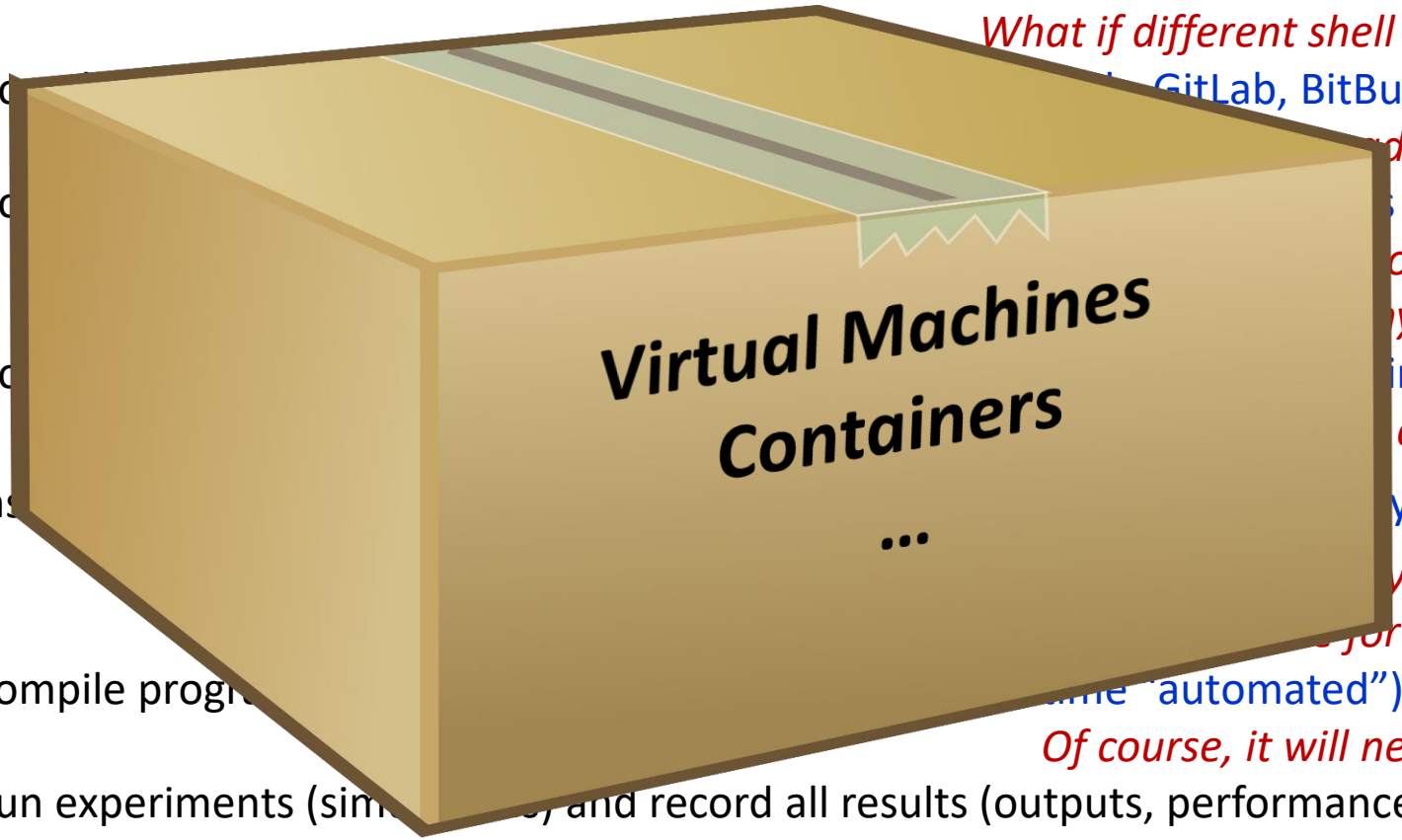
    Of course, it will never fail ;) !!!

  - Run experiments (sim … and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?

  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!



How can we automate all those steps and enable DevOps?

How the community run, share, reproduce and reuse experiments

- Download an archive or some container with artifacts from an accepted paper (manually)

- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  What if different shell or even OS?

  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    What if some sources are already available?

  - Download dataset(s) from some external sources (often automated unless included)

    What if URL changed or file is not available?

    What if I want to try different datasets or my own ones?

  - Download model(s) from some external sources (often automated unless included)

    What if I want to try different models or my own ones? Will they be compatible?

  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?

  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!

  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?

  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  What if different shell or even OS?
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    What if some sources are already available?
  - Download dataset(s) from some external sources (often automated unless included)

    What if URL changed or file is not available?

    What if I want to try different datasets or my own ones?
  - Download model(s) from some external sources (often automated unless included)

    What if I want to try different models or my own ones? Will they be compatible?
  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?
  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    What if some sources are already available?
  - Download dataset(s) from some external sources (often automated unless included)

    What if URL changed or file is not available?

    What if I want to try different datasets or my own ones?
  - Download model(s) from some external sources (often automated unless included)

    What if I want to try different models or my own ones? Will they be compatible?
  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?
  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    What if URL changed or file is not available?

    What if I want to try different datasets or my own ones?
  - Download model(s) from some external sources (often automated unless included)

    What if I want to try different models or my own ones? Will they be compatible?
  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?
  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    PI-API: install *dataset* compatible *with a program* from *Supercomputing18 paper*

    PI-API: detect *datasets* compatible *with a program* from *Supercomputing18 paper*
  - Download model(s) from some external sources (often automated unless included)

    What if I want to try different models or my own ones? Will they be compatible?
  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?
  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    PI-API: install *dataset* compatible *with a program* from *Supercomputing18 paper*

    PI-API: detect *datasets* compatible *with a program* from *Supercomputing18 paper*
  - Download model(s) from some external sources (often automated unless included)

    PI-API: install *model* compatible *with a program* from *Supercomputing18 paper*
  - Install numerous software dependencies (often manually or semi-manually)

    What if some are already installed? Can I reuse them? What if they are newer?

    What if they are not available for my system?
  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    PI-API: install *dataset* compatible *with a program* from *Supercomputing18 paper*

    PI-API: detect *datasets* compatible *with a program* from *Supercomputing18 paper*
  - Download model(s) from some external sources (often automated unless included)

    PI-API: install *model* compatible *with a program* from *Supercomputing18 paper*
  - Install numerous software dependencies (often manually or semi-manually)

    PI-API: install *package* for *a program* from *Supercomputing18 paper*

    PI-API: detect *soft* compatible *with a program* from *Supercomputing18 paper*
  - Compile program and some dependencies (most of the time "automated")

    Of course, it will never fail ;) !!!
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    PI-API: install *dataset* compatible *with a program* from *Supercomputing18 paper*

    PI-API: detect *datasets* compatible *with a program* from *Supercomputing18 paper*
  - Download model(s) from some external sources (often automated unless included)

    PI-API: install *model* compatible *with a program* from *Supercomputing18 paper*
  - Install numerous software dependencies (often manually or semi-manually)

    PI-API: install *package* for *a program* from *Supercomputing18 paper*

    PI-API: detect *soft* compatible *with a program* from *Supercomputing18 paper*
  - Compile program and some dependencies (most of the time "automated")

    PI-API: compile *a program* from *Supercomputing18 paper* with *flags="-O3 –flto"*
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    What can possibly go wrong (of course APIs and command lines never change) ;) ?
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    PI-API: install *dataset* compatible *with a program* from *Supercomputing18 paper*

    PI-API: detect *datasets* compatible *with a program* from *Supercomputing18 paper*
  - Download model(s) from some external sources (often automated unless included)

    PI-API: install *model* compatible *with a program* from *Supercomputing18 paper*
  - Install numerous software dependencies (often manually or semi-manually)

    PI-API: install *package* for *a program* from *Supercomputing18 paper*

    PI-API: detect *soft* compatible *with a program* from *Supercomputing18 paper*
  - Compile program and some dependencies (most of the time "automated")

    PI-API: compile *a program* from *Supercomputing18 paper* with *flags="-O3 –flto"*
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    PI-API: run *a program* with *compatible dataset and model, automatically record steps*
  - Process results, compare with the paper, and report discrepancies (often manually)

    That's the simplest part - just check 5 tables, 10 graphs, and just spot the difference!!!

How can we automate all those steps and enable DevOps?

Platform independent and human readable APIs (from any language or CMD)

- Download an archive or some container with artifacts from an accepted paper (manually)

  Platform Independent API: pull *repository* for *Supercomputing18 paper*
- Go to *scripts* directory (ad-hoc *.sh or python scripts, and occasionally Jupyter notebooks)

  PI-API: find *scripts* for *Supercomputing18 paper*
  - Download source code (typically with recursion from GitHub, GitLab, BitBucket)

    PI-API: install *soft* for *a program* from *Supercomputing18 paper*
  - Download dataset(s) from some external sources (often automated unless included)

    PI-API: install *dataset* compatible *with a program* from *Supercomputing18 paper*

    PI-API: detect *datasets* compatible *with a program* from *Supercomputing18 paper*
  - Download model(s) from some external sources (often automated unless included)

    PI-API: install *model* compatible *with a program* from *Supercomputing18 paper*
  - Install numerous software dependencies (often manually or semi-manually)

    PI-API: install *package* for *a program* from *Supercomputing18 paper*

    PI-API: detect *soft* compatible *with a program* from *Supercomputing18 paper*
  - Compile program and some dependencies (most of the time "automated")

    PI-API: compile *a program* from *Supercomputing18 paper* with *flags="-O3 –flto"*
  - Run experiments (simulations) and record all results (outputs, performance …) in raw, text and CSV files (usually semi-automated)

    PI-API: run *a program* with *compatible dataset and model*, automatically record steps
  - Process results, compare with the paper, and report discrepancies (often manually)

    PI-API: validate results from *a program* using *pre-recorded ones,* auto-generate paper

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

$ sudo pip install **ck**

Now can implement, share and reuse APIs as Python modules via CK repositories

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

$ sudo pip install **ck**

Now can implement, share and reuse APIs as Python modules via CK repositories

$ **ck** add repo:my-paper --quiet

Local directory: $HOME/CK/my-paper/.ckr.json (repo description and dependencies)

$ **ck** ls repo or **ck** search repo

my-paper local default

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

$ sudo pip install **ck**

Now can implement, share and reuse APIs as Python modules via CK repositories

$ **ck** add repo:my-paper --quiet

Local directory: $HOME/CK/my-paper/.ckr.json (repo description and dependencies)

$ **ck** ls repo or **ck** search repo

my-paper local default

$ **ck** add my-paper:module:hello

$ **ck** add_action module:hello --func=say

Local directory: $HOME / CK / my-paper / module / hello / module.py

$HOME / CK / my-paper / module / hello / .cm / meta.json

```
def say(i):
    print (json.dumps(i))
    actions=cfg['actions']
    return {'return':0, 'error':''}
```

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

```
$ sudo pip install ck
```

Now can implement, share and reuse APIs as Python modules via CK repositories

```
$ ck add repo:my-paper --quiet
```

Local directory: $HOME/CK/my-paper/.ckr.json (repo description and dependencies)

```
$ ck ls repo     or     ck search repo
```

my-paper local default

```
$ ck add my-paper:module:hello
$ ck add_action module:hello --func=say
```

Local directory: $HOME / CK / my-paper / module / hello / module.py

$HOME / CK / my-paper / module / hello / .cm / meta.json

```
def say(i):
    print (json.dumps(i))
    actions=cfg['actions']
    return {'return':0, 'error':''}
```

```
$ ck say hello --fosdem --is=cool @input.json
```

{ "action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool" …}

```
$ python      (or Jupyter notebooks)
import ck.kernel as ck
r=ck.access({"action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool"})
print (r)
{'return':0}
```

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

```
$ sudo pip install ck
```

Now can implement, share and reuse APIs as Python modules via CK repositories

```
$ ck add repo:my-paper --quiet
```

Local directory: $HOME/CK/my-paper/.ckr.json (repo description and dependencies)

```
$ ck ls repo     or     ck search repo
```

my-paper local default

```
$ ck add my-paper:module:hello
$ ck add_action module:hello --func=say
```

```
def say(i):
    print (json.dumps(i))
    actions=cfg['actions']
    return {'return':0, 'error':''}
```

Local directory: $HOME / CK / my-paper / module / hello / module.py

$HOME / CK / my-paper / module / hello / .cm / meta.json

```
$ ck say hello --fosdem --is=cool @input.json
```

{ "action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool" …}

```
$ python      (or Jupyter notebooks)
import ck.kernel as ck
r=ck.access({"action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool"})
print (r)
{'return':0}
```

```
$ ck add my-paper:hello:world –tags=fosdem,20190203
```

{ "tags":["fosdem","20190203"] }

Local directory: $HOME / CK / my-paper / hello / world / .cm / meta.json

$HOME / CK / my-paper / hello / world / <- holder for files and dirs

```
$ ck search hello –tags=fosdem --all
```

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

```
$ sudo pip install ck
```

Now can implement, share and reuse APIs as Python modules via CK repositories

```
$ ck add repo:my-paper --quiet
```

Local directory: $HOME/CK/my-paper/.ckr.json (repo description and dependencies)

```
$ ck ls repo     or     ck search repo
```

my-paper local default

```
$ ck add my-paper:module:hello
$ ck add_action module:hello --func=say
```

```
def say(i):
    print (json.dumps(i))
    actions=cfg['actions']
    return {'return':0, 'error':''}
```

Local directory: $HOME / CK / my-paper / module / hello / module.py

$HOME / CK / my-paper / module / hello / .cm / meta.json

```
$ ck say hello --fosdem --is=cool @input.json
```

{ "action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool" …}

```
$ python      (or Jupyter notebooks)
import ck.kernel as ck
r=ck.access({"action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool"})
print (r)
{'return':0}
```

```
$ ck add my-paper:hello:world –tags=fosdem,20190203
```

{ "tags":["fosdem","20190203"] }

Local directory: $HOME / CK / my-paper / hello / world / .cm / meta.json

$HOME / CK / my-paper / hello / world / <- holder for files and dirs

```
$ ck search hello –tags=fosdem --all
$ ck load hello:world --min
```

{ "tags":["fosdem","20190203"] }

Collective Knowledge basics: github.com/ctuning/ck/wiki and cKnowledge.org

```
$ sudo pip install ck
```

Now can implement, share and reuse APIs as Python modules via CK repositories

```
$ ck add repo:my-paper --quiet
```

Local directory: $HOME/CK/my-paper/.ckr.json (repo description and dependencies)

```
$ ck ls repo     or     ck search repo
```

my-paper local default

```
$ ck add my-paper:module:hello
$ ck add_action module:hello --func=say
```

```
def say(i):
    print (json.dumps(i))
    actions=cfg['actions']
    return {'return':0, 'error':''}
```

Local directory: $HOME / CK / my-paper / module / hello / module.py

$HOME / CK / my-paper / module / hello / .cm / meta.json

```
$ ck say hello --fosdem --is=cool @input.json
```

{ "action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool" …}

```
$ python      (or Jupyter notebooks)
import ck.kernel as ck
r=ck.access({"action":"say", "module_uoa":"hello", "fosdem":"yes", "is":"cool"})
print (r)
{'return':0}
```

```
$ ck add my-paper:hello:world –tags=fosdem,20190203
```

{ "tags":["fosdem","20190203"] }

Local directory: $HOME / CK / my-paper / hello / world / .cm / meta.json

$HOME / CK / my-paper / hello / world / <- holder for files and dirs

```
$ ck search hello –tags=fosdem --all
$ ck load hello:world --min
```

{ "tags":["fosdem","20190203"] }

```
$ ck say hello:world -f
```

{ "action":"say", "module_uoa":"hello", "data_uoa":"world", "f":"yes" …}

Collective knowledge: collaboratively abstract, describe and reuse everything!

Applications
- Meteorology
- Health
- Robotics
- Automotive
- Economics
- Physics
- Astronomy
- Education

Scientific tools
- MATLAB
- Scilab
- Simulink
- LabVIEW
- Gnuplot
- LaTeX
- Ipython

Build tools
- Make
- Cmake
- SCons
- Bazel
- Gradle
- Ninja

Languages
- C++
- C#
- C
- Go
- PHP
- Fortran
- Java
- Python

Compilers
- LLVM
- GCC
- Intel
- PGI
- TVM
- CUDA

DevOps tools
- Git
- Jenkins
- Docker
- Kubernetes
- Singularity

Package managers
- Anaconda
- Go
- Npm
- Pip
- Sbt
- dpkg
- Spack
- EasyBuild

Libraries
- SciPy
- TFLite
- OpenBLAS
- MAGMA
- cuDNN
- cuFFT
- ArmNN
- CLBlast
- gemmlowp
- Boost
- HDF5
- MPI
- OpenCV
- Protobuf

OS
- Linux
- MacOS
- BSD
- Windows
- Android

Shells
- bash
- sh
- csh
- ksh
- Windows shell

Programs
- Image classification
- Object detection
- Natural Language processing
- Text processing
- Video processing
- Personal assistant

Collective Knowledge

- **Simple Python APIs with JSON (dictionary) I/O**
- **Simple JSON meta-description of all components**
- **Simple access from command line, different languages and web**
- **Simple sharing of all components via GitHub, Zenodo, etc …**

Benchmarks
- SPEC
- EEMBC
- HPCG
- LINPACK
- cBench
- MLPerf

Hardware
- CPU
- GPU
- TPU / NN
- DSP
- FPGA
- Quantum
- Simulators
- Interconnects

AI/ML frameworks
- TensorFlow
- PyTorch
- MXNet
- Caffe
- MCT (CNTK)
- Keras
- Kubeflow
- AutoML
- SageMaker
- Apache Spark

Models
- GoogleNet
- AlexNet
- VGG
- ResNet
- MobileNets
- SSD
- SqueezeNet
- DeepSpeech

Datasets
- ImageNet
- KITTI
- COCO
- MiDataSets
- Human Cell Atlas
- 1000 Genomes
- Earth models
- OpenStreetMap

Workload managers
- MPI
- SLURM
- PBS
- FLUX

Web services
- GitHub
- GitLab
- BitBucket
- Travis
- JupyterHub
- Codelabs
- SageMaker

Databases / experiments
- MySQL
- PostgreSQL
- MongoDB
- CouchDB
- Text files
- JSON files
- XLS files

Knowledge sharing
- ArXiv
- ACM DL
- IEEE DL
- GitHub
- Zenodo
- FigShare
- Web pages

Platforms
- HPC
- Desktops
- IoT
- Mobile
- Cloud sevices

Implement workflows (pipelines) adaptable to any SW/HW! Focus on innovation!

Applications
- Meteorology
- Health
- Robotics
- Automotive
- Economics
- Physics
- Astronomy
- Education

Scientific tools
- MATLAB
- Scilab
- Simulink
- LabVIEW
- Gnuplot
- LaTeX
- Ipython

Build tools
- Make
- Cmake
- SCons
- Bazel
- Gradle
- Ninja

Languages
- C++
- C#
- C
- Go
- PHP
- Fortran
- Java
- Python

Compilers
- LLVM
- GCC
- Intel
- PGI
- TVM
- CUDA

DevOps tools
- Git
- Jenkins
- Docker
- Kubernetes
- Singularity

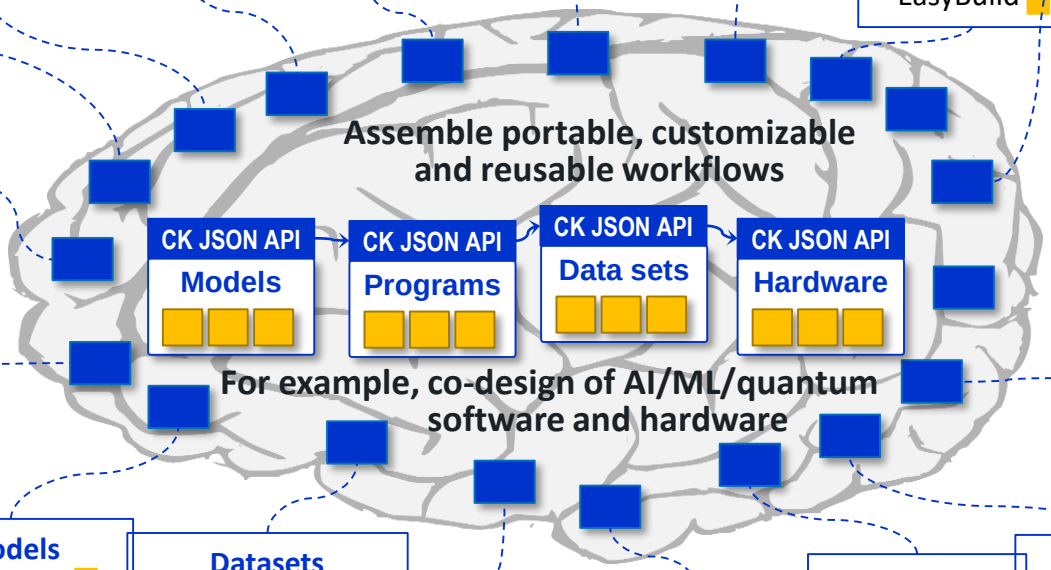
Package managers
- Anaconda
- Go
- Npm
- Pip
- Sbt
- dpkg
- Spack
- EasyBuild

Libraries
- SciPy
- TFLite
- OpenBLAS
- MAGMA
- cuDNN
- cuFFT
- ArmNN
- CLBlast
- gemmlowp
- Boost
- HDF5
- MPI
- OpenCV
- Protobuf

OS
- Linux
- MacOS
- BSD
- Windows
- Android

Shells
- bash
- sh
- csh
- ksh
- Windows shell

Programs
- Image classification
- Object detection
- Natural Language processing
- Text processing
- Video processing
- Personal assistant



Benchmarks
- SPEC
- EEMBC
- HPCG
- LINPACK
- cBench
- MLPerf

Hardware
- CPU
- GPU
- TPU / NN
- DSP
- FPGA
- Quantum
- Simulators
- Interconnects

AI/ML frameworks
- TensorFlow
- PyTorch
- MXNet
- Caffe
- MCT (CNTK)
- Keras
- Kubeflow
- AutoML
- SageMaker
- Apache Spark

Models
- GoogleNet
- AlexNet
- VGG
- ResNet
- MobileNets
- SSD
- SqueezeNet
- DeepSpeech

Datasets
- ImageNet
- KITTI
- COCO
- MiDataSets
- Human Cell Atlas
- 1000 Genomes
- Earth models
- OpenStreetMap

Workload managers
- MPI
- SLURM
- PBS
- FLUX

Web services
- GitHub
- GitLab
- BitBucket
- Travis
- JupyterHub
- Codelabs
- SageMaker

Databases / experiments
- MySQL
- PostgreSQL
- MongoDB
- CouchDB
- Text files
- JSON files
- XLS files

Knowledge sharing
- ArXiv
- ACM DL
- IEEE DL
- GitHub
- Zenodo
- FigShare
- Web pages

Platforms
- HPC
- Desktops
- IoT
- Mobile
- Cloud sevices

Anyone can share reusable components (code and data) with common APIs and meta descriptions

Started converting all my own past R&D on autotuning and machine learning to CK

cKnowledge.org/shared-repos.html cKnowledge.org/shared-modules.html

Artifact automated and reusable

Collective Knowledge COMPATIBLE

Workflow CK

Anyone can share reusable components (code and data) with common APIs and meta descriptions

Started converting all my own past R&D on autotuning and machine learning to CK

cKnowledge.org/shared-repos.html cKnowledge.org/shared-modules.html

Artifact automated and reusable Collective Knowledge COMPATIBLE Workflow CK

1) Describing different operating systems (github.com/ctuning/ck-env)

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

Anyone can share reusable components (code and data) with common APIs and meta descriptions

Started converting all my own past R&D on autotuning and machine learning to CK

cKnowledge.org/shared-repos.html cKnowledge.org/shared-modules.html

Artifact automated and reusable Collective Knowledge COMPATIBLE Workflow CK

1) Describing different operating systems (github.com/ctuning/ck-env)

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

2) Detecting and unifying information about platforms

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

Anyone can share reusable components (code and data) with common APIs and meta descriptions

Started converting all my own past R&D on autotuning and machine learning to CK

cKnowledge.org/shared-repos.html cKnowledge.org/shared-modules.html

Artifact | automated and reusable Collective Knowledge | COMPATIBLE Workflow | CK

1) Describing different operating systems (github.com/ctuning/ck-env)

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

2) Detecting and unifying information about platforms

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

3) Detecting installed "software" (**both** code and data):

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

cKnowledge.org/shared-soft-detection-plugins.html

```
$ ck show env --tags=llvm
```

Anyone can share reusable components (code and data) with common APIs and meta descriptions

Started converting all my own past R&D on autotuning and machine learning to CK

cKnowledge.org/shared-repos.html cKnowledge.org/shared-modules.html

Artifact automated and reusable Collective Knowledge COMPATIBLE Workflow CK

1) Describing different operating systems (github.com/ctuning/ck-env)

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

2) Detecting and unifying information about platforms

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

3) Detecting installed "software" (**both** code and data):

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

cKnowledge.org/shared-soft-detection-plugins.html

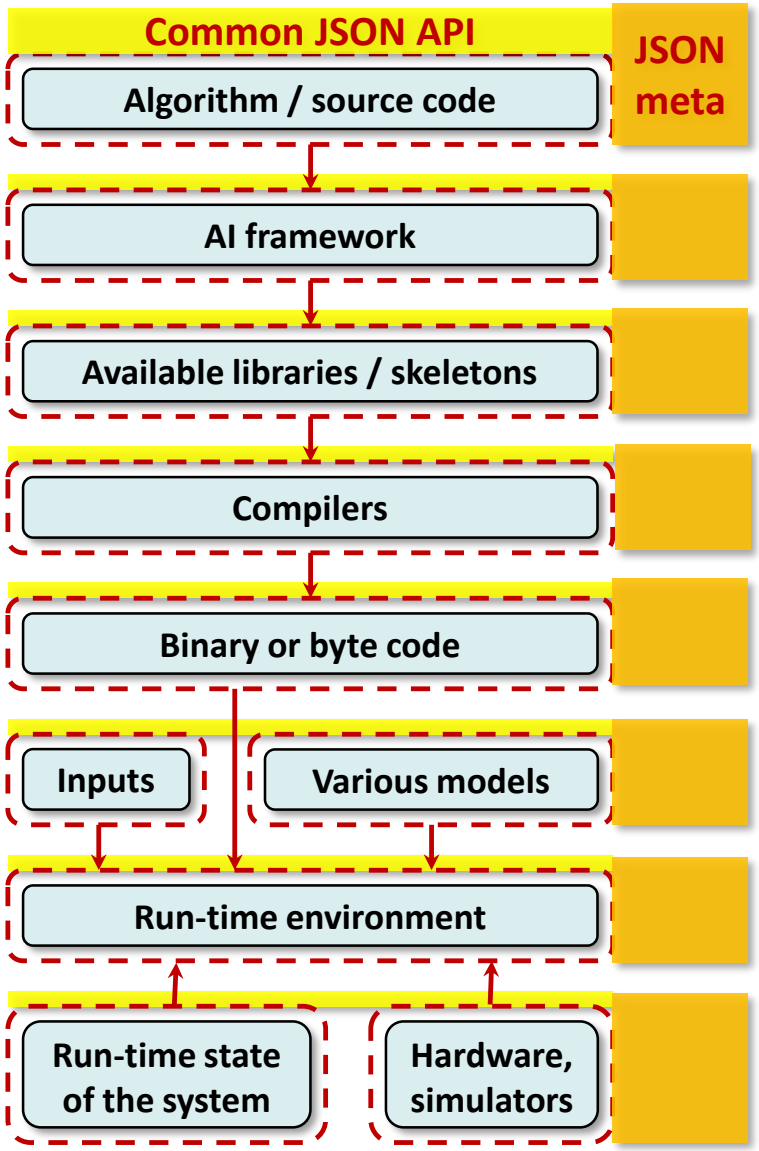
```
$ ck show env --tags=llvm
```

4) Installing missing packages (**both** code and data): front-end to EasyBuild, Spack, scons, cmake

```
$ ck search package --tags=model
$ ck install compiler:compiler-llvm-7.0.0-universal
$ ck show env --tags=llvm
$ ck virtual env –tags=llvm,v7.0.0
```

cKnowledge.org/shared-packages.html

Enabling customizable and portable workflows by connecting CK components



Universal program workflow to compile, run and profile diverse benchmarks with different data sets, validate results, record experiments, share and reproduce them, and report discrepancies

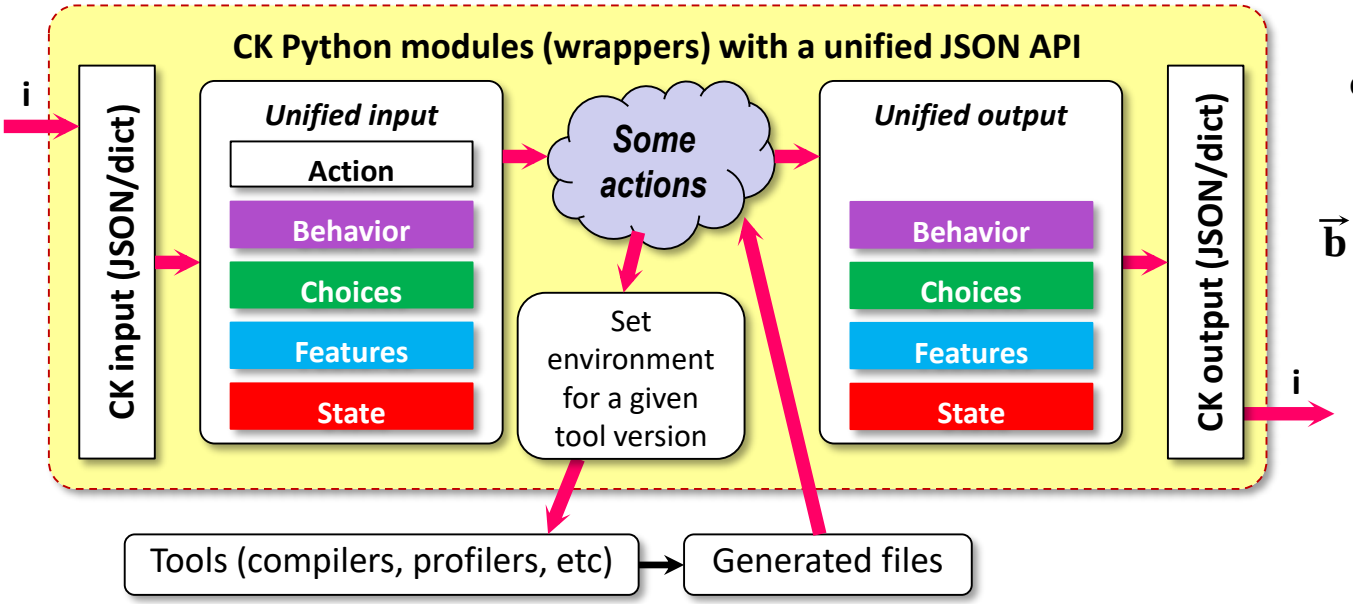
http://cKnowledge.org/shared-programs.html

```
$ 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
```

Having APIs and JSON meta enables DevOps and easy integration with Jenkins, Travis, etc.
CK complements containers and can be easily used in Jupyter notebooks!

Implementing autotuning pipeline of the whole AI/ML/SW/HW stack!

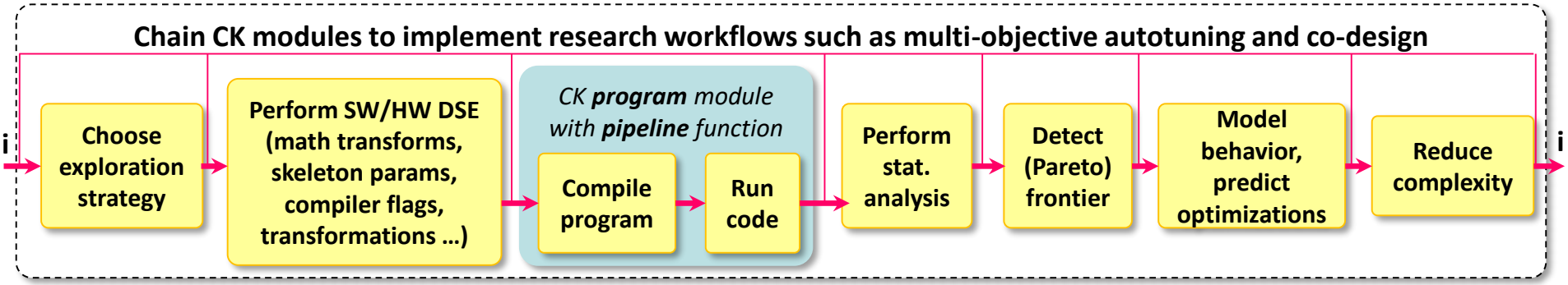
Collaboratively expose choices, features, system state and behavior characteristics



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



We can even automatically generate reproducible and interactive articles (collaboration with Raspberry Pi foundation): cKnowledge.org/rpi-crowd-tuning

Real-world use cases with our partners: cKnowledge.org/partners

Repositories of customizable, portable and reusable research components with CK API

cKnowledge.org/shared-repos.html

CK JSON API
AI frameworks
- TensorFlow
- Caffe
- Caffe2
- CNTK
- Torch
- MXNet
- ...

CK JSON API
Models
- AlexNet
- GoogleNet
- VGG
- ResNet
- SqueezeDet
- SSD
- MobileNets
- ...

CK JSON API
Libraries
- cuDNN
- ArmCL
- OpenBLAS
- ViennaCL
- CLBlast
- cuBLAS
- TVM
- ...

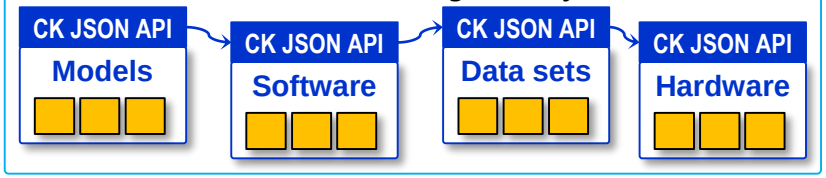
CK JSON API
Data sets
- KITTI
- COCO
- VOC
- ImageNet
- Real life objects from the community
- ...

CK JSON API
OS
- Linux
- Windows
- MacOS
- Android
- ...

CK JSON API
Hardware
- CPU
- GPU
- DSP
- NN accelerators
- FPGA
- Simulators
- ...

Customizable CK workflows for real-world user tasks

Assemble scenarios such as image classification as LEGO™



Share complete workflows along with published papers to automate artifact evaluation and help the community build upon prior work

Crowdsource experiments with the help of volunteers across diverse models, data sets and platforms

Present best results, workflows and components on a live scoreboard for fair comparison and reuse

cKnowledge.org/repo

Help students learn multidisciplinary techniques, quickly prototype new ones, validate them in practice with companies, and even contribute back new research components

Help companies select the most appropriate workflows, save R&D costs, accelerate adoption of new techniques!

Open science: organizing reproducible tournaments and sharing research components

2018: many cross-disciplinary R&D groups (ML/AI/systems)

AI hardware
- All major vendors (Google, NVIDIA, ARM, Intel, IBM, Qualcomm, Apple, AMD …)

AI models
Many groups in academia & industry (Google, OpenAI, Microsoft, Facebook …)

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

AI integration/services
- Cloud 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, dividiti/cTuning foundation
Anton Lokhmotov, dividiti
Thierry Moreau, University of Washington
Adrian Sampson, Cornell University
Phillip Stanley Marbell, University of Cambridge

Real use-cases

Healthcare
Agriculture
Finances
Automotive
Aerospace
Meteorology
Retail
Robotics
…

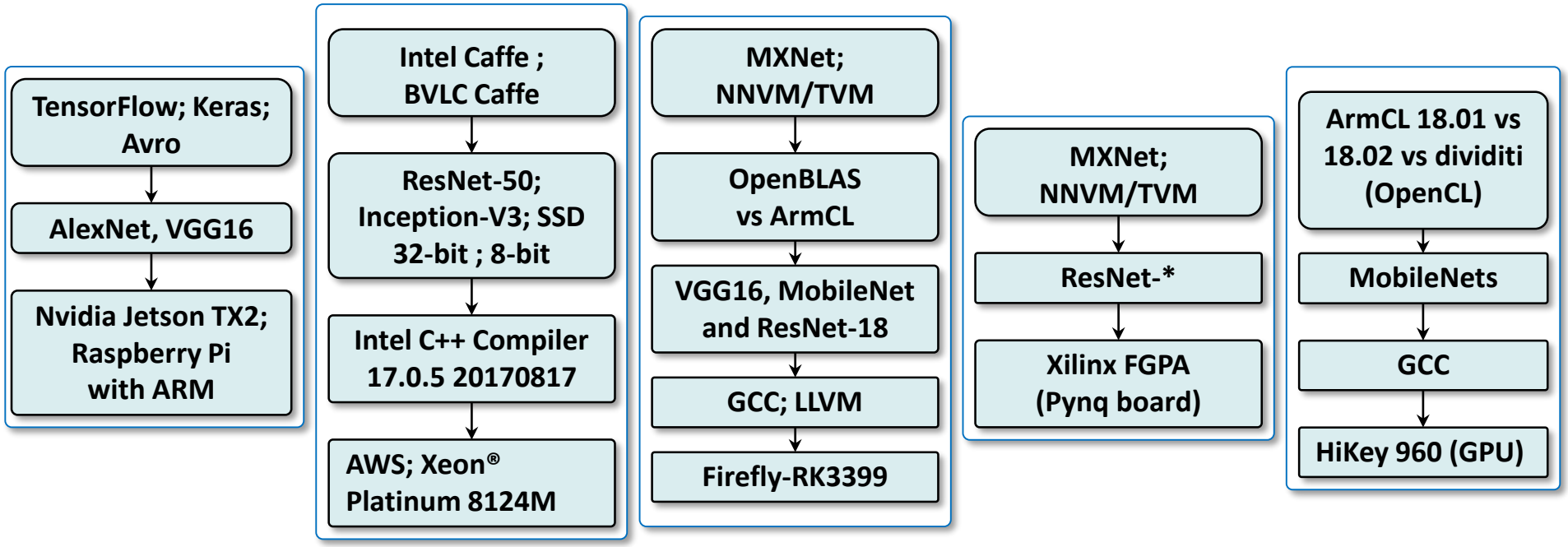
Collective Knowledge Platform

Interdisciplinary community

* Workshop organizers

We organized the 1st reproducible tournament at ACM ASPLOS'18

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



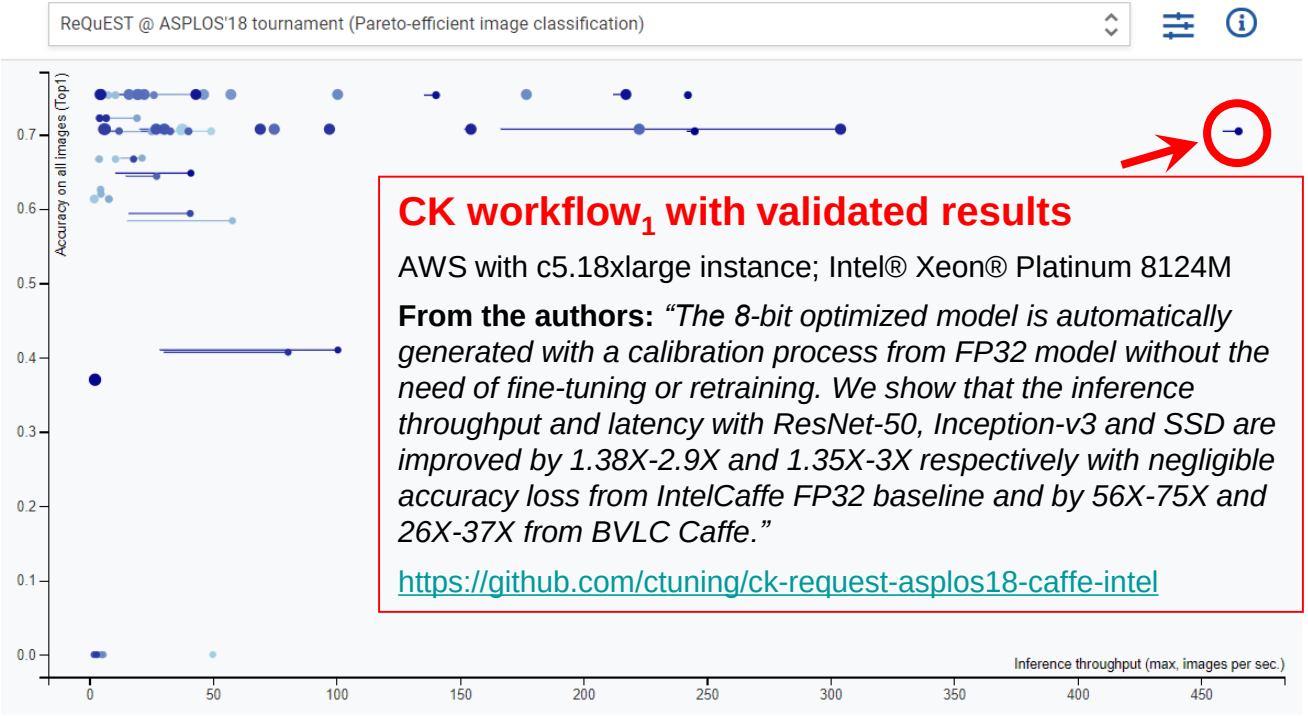
Public validation at github.com/ctuning/ck-request-asplos18-results via GitHub issues.

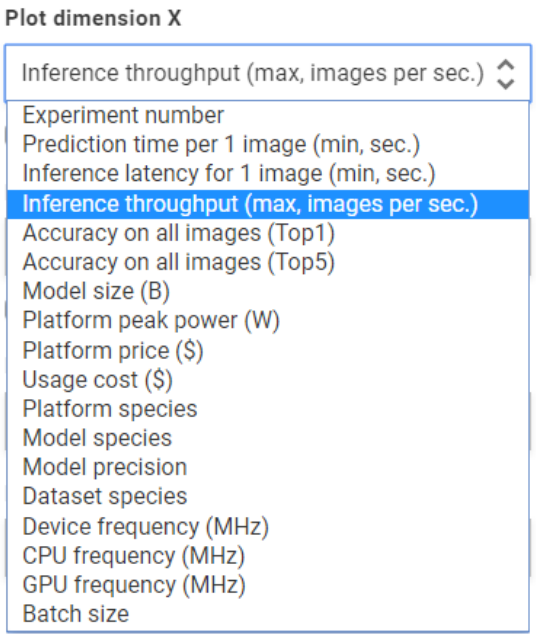
All validated papers are published in the ACM DL
with **portable, customizable and reusable CK components and workflows**:
dl.acm.org/citation.cfm?doid=3229762

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

All results and research components are available via a live CK scoreboard

Multi-objective results for all AI/SW/HW stacks are presented on a live scoreboard and become available for public comparison and further customization, optimization and reuse!



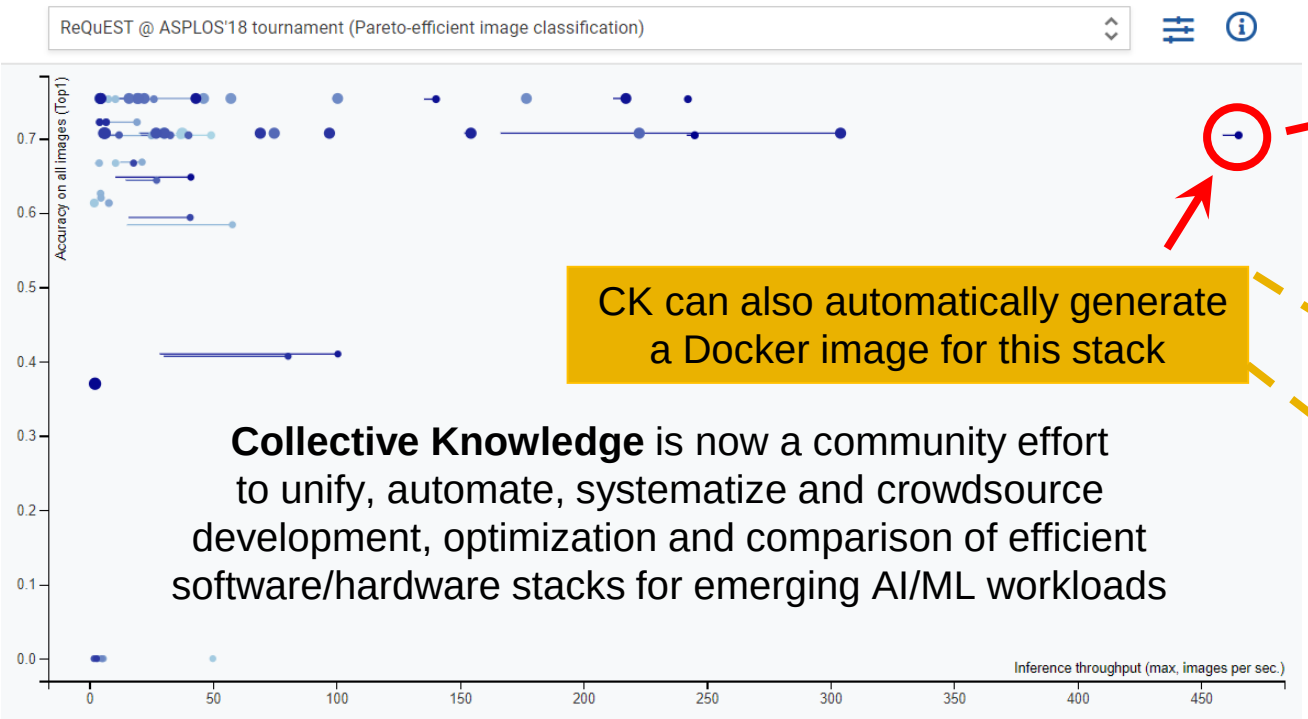


We are not announcing a single winner! We show all multi-dimensional results at cKnowledge.org/dashboard/request.asplos18

and let users select best ML/SW/HW stacks depending on multiple constraints!

Other companies managed to reproduce results and started using CK

Multi-objective results for all AI/SW/HW stacks are presented on a live scoreboard and become available for public comparison and further customization, optimization and reuse!



CK can also automatically generate a Docker image for this stack

Collective Knowledge is now a community effort to unify, automate, systematize and crowdsource development, optimization and comparison of efficient software/hardware stacks for emerging AI/ML workloads

CK assists AWS market place with collaboratively optimized AI/ML stacks

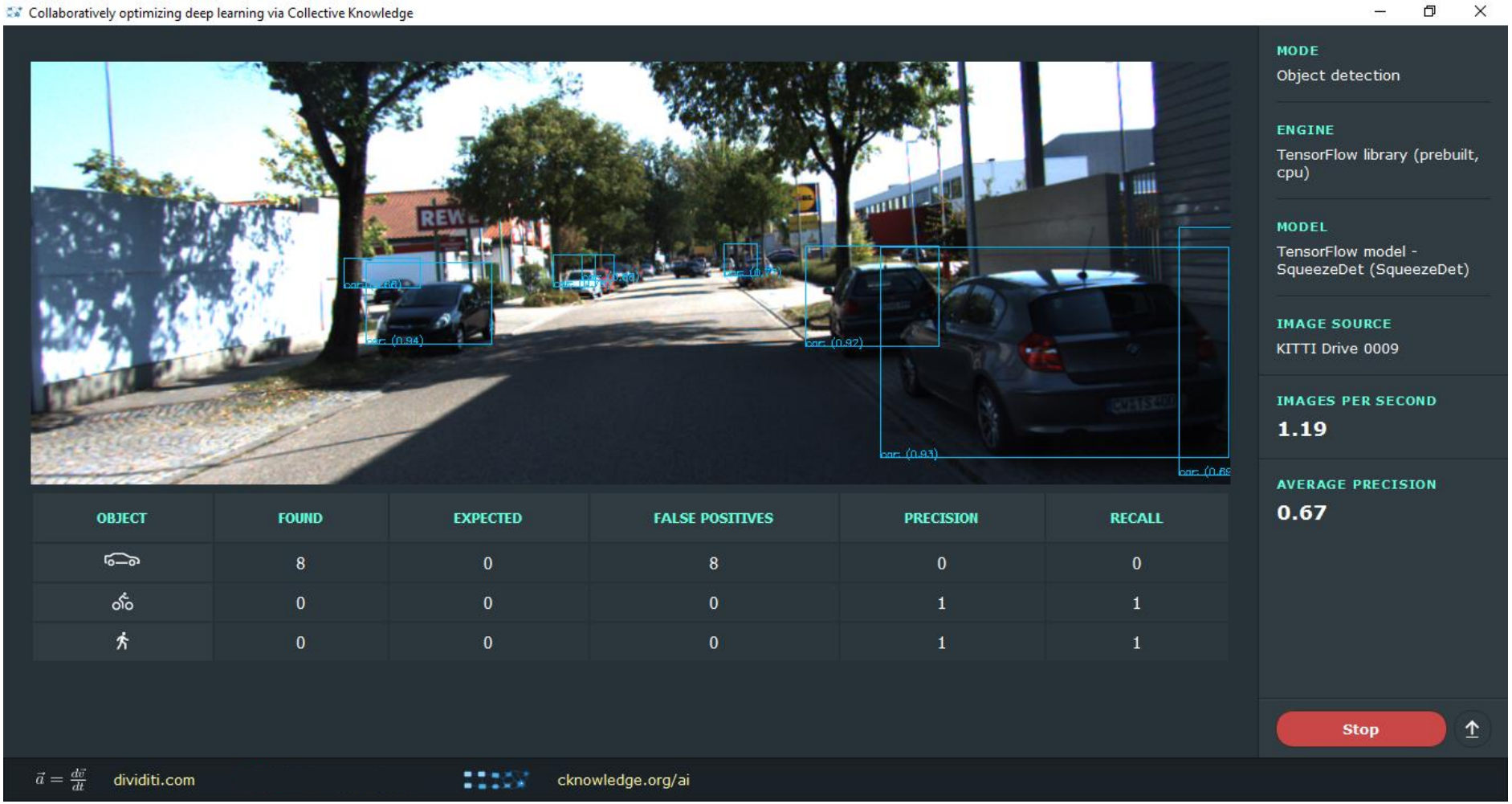
amazon web services

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

See Amazon presentation at O'Reilly AI conference:

conferences.oreilly.com/artificial-intelligence/ai-eu/public/schedule/detail/71549

CK helps General Motors to select the most efficient SW/HW stacks



OBJECT	FOUND	EXPECTED	FALSE POSITIVES	PRECISION	RECALL
	8	0	8	0	0
	0	0	0	1	1
	0	0	0	1	1

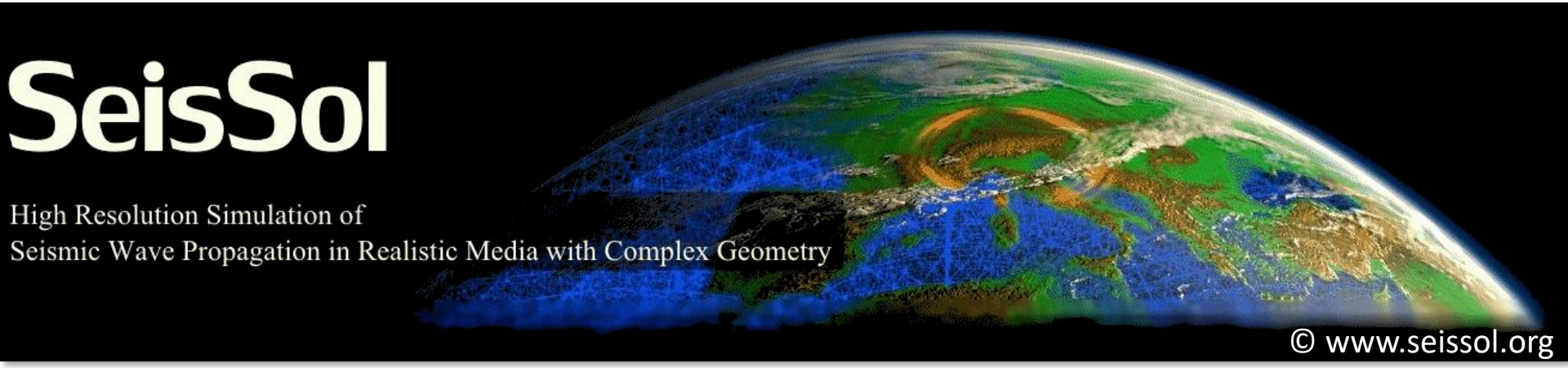
Performance, accuracy, power consumption practically never match official reports!

CK allows to select the most efficient SW/HW stacks on a Pareto frontier (performance, accuracy, energy, memory usage, costs) for object detection, image classification and other tasks: www.youtube.com/watch?v=1ldgVZ64hEI

CK helps to automate Student Cluster competitions

github.com/ctuning/ck-scc18/wiki - proof-of-concept CK workflow
to automate installation, execution and customization of SeisSol application
from the SC18 SCC Reproducibility Challenge
across different platforms, environments and datasets

Artifact automated and reusable | Collective Knowledge COMPATIBLE | Workflow CK



- Support automatic detection of already installed tools and data sets
- Can install missing dependencies via EasyBuild and Spack
- Can deploy application on different supercomputers with different job managers
- Can automatically validate the correctness of results (output, performance)

Technical Program

Call for Participation

Papers
March 1, 2019 – Submissions open
April 10, 2019 – Full paper deadline

Tutorials
February 15, 2019 – Submissions open
April 16, 2019 – Submissions close

Panels
February 15, 2019 – Submissions open
April 23, 2019 – Submissions close

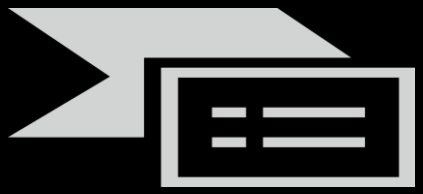
Workshops
January 1, 2019 – Submissions open
February 14, 2019 – Submissions close

Posters
February 15, 2019 – Submissions open
July 31, 2019 – Submissions close

More Opportunities
Awards
Birds of a Feather
Early Career
Exhibitor Forum

Being part of the SC Conference enhances your career – whether you are presenting new research, showcasing innovative work or practices, helping teach the next generation, or competing for peak performance. The SC selection process is highly competitive and being selected is extremely rewarding.

Submit your work to SC19!
sc19.supercomputing.org/submit/



Program: November 17–22, 2019
Exhibits: November 18–21, 2019

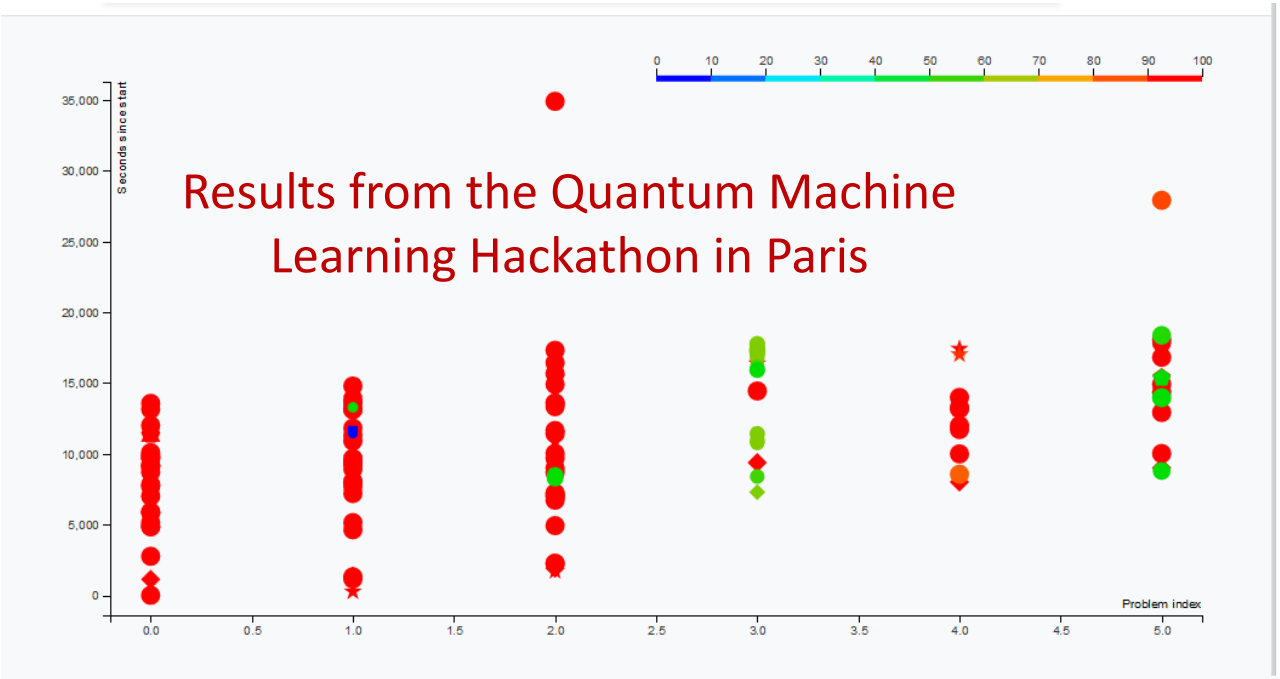
Colorado Convention Center, Denver, CO

The International Conference for High Performance Computing, Networking, Storage, and Analysis

CK is used to collaboratively advance quantum computing

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

cKnowledge.org/dashboard/hackathon.20190127



#	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

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

rigetti

RIVERLANE

ThoughtWorks®

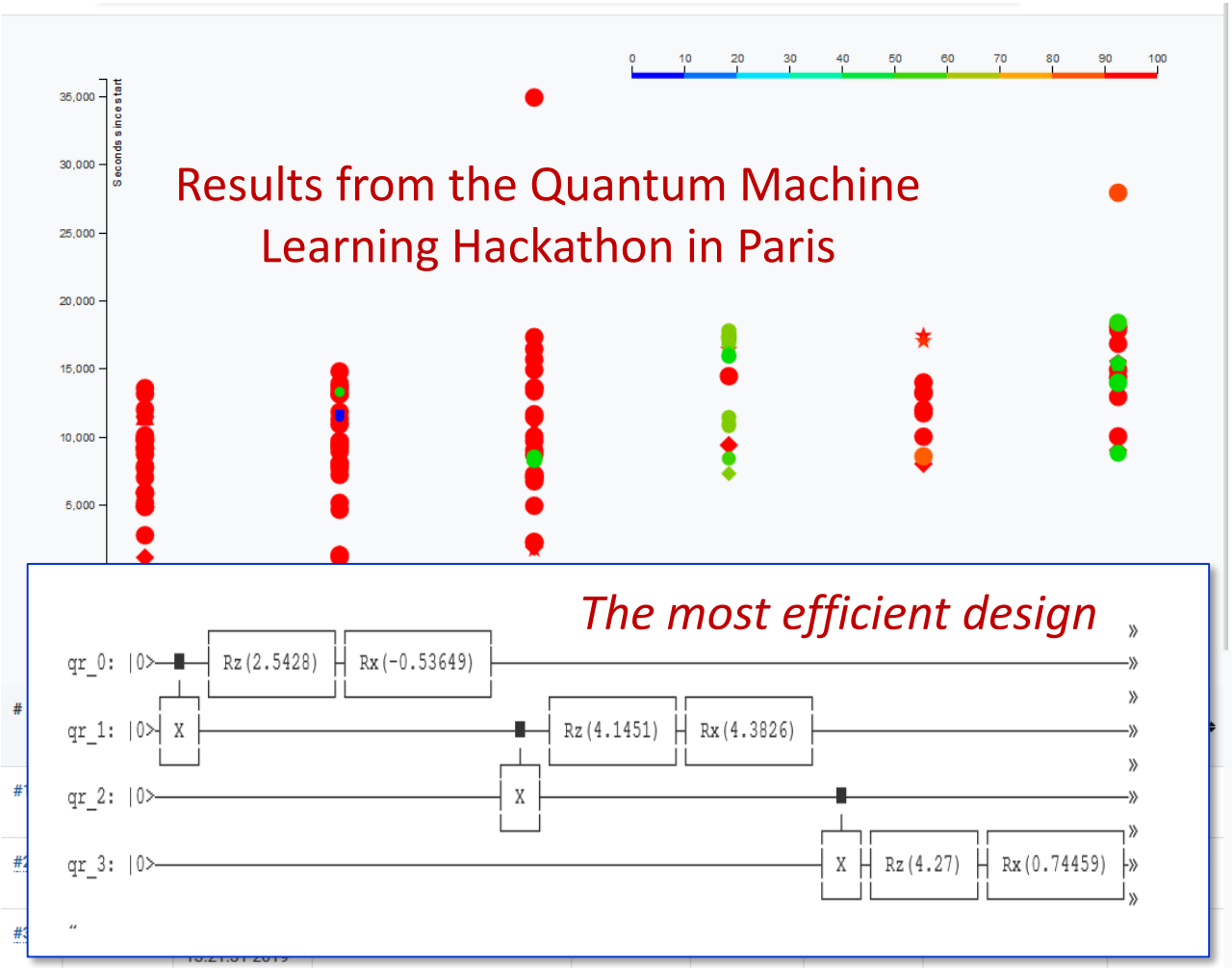
QUANTO NATION

Innovate UK

CK is used to collaboratively advance quantum computing

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

cKnowledge.org/dashboard/hackathon.20190127



cKnowledge.org: future plans to enable open science

From prototype to production quality (beginning of a long journey)

- Collaboratively standardize APIs and meta descriptions
- Improve installation and documentation
- Add more CK components and workflows for real-world tasks

Open to collaboration

- Joint R&D projects and tournaments (AI, ML, quantum)
- Automation and sharing of experiments
- Reproducible articles with reusable workflows

Websites:

- github.com/ctuning/ck
- cKnowledge.org/shared-repos.html

Contact

Grigori.Fursin@cTuning.org or grigori@dividiti.com