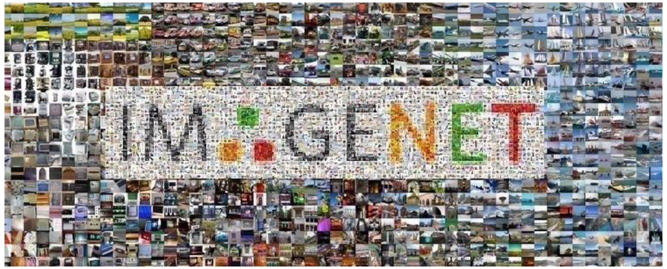
BenchOpt Tutorial

MIND

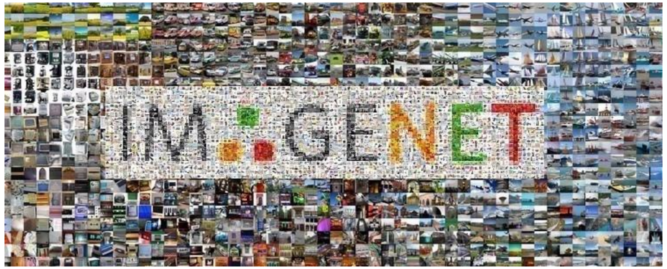
The ImageNet competition

- Annual competition since 2010



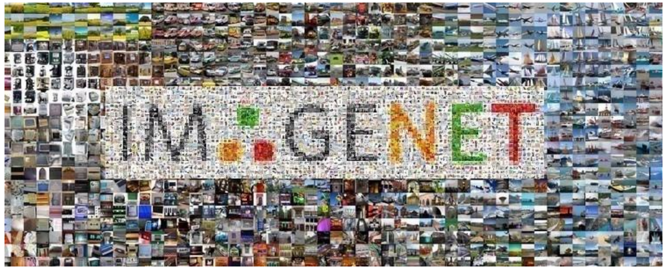
The ImageNet competition

- Annual competition since 2010
- Evaluate image classification methods with 14M labeled images among 1k categories



The ImageNet competition

- Annual competition since 2010
- Evaluate image classification methods with 14M labeled images among 1k categories
- Boosted AI and Deep Learning research when Alex Krizhevsky won in 2012.



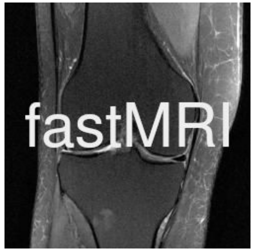
Many benchmarks

Many benchmarks followed ImageNet:

- Natural Language Processing: GLUE, SuperGLUE
- Reinforcement Learning: Atari, MuJoCo, OpenAI Gym
- Others: fastMRI, DAWNBench, MLPerf, etc.

GLUE

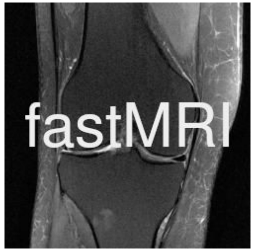
Gymnasium



Many benchmarks

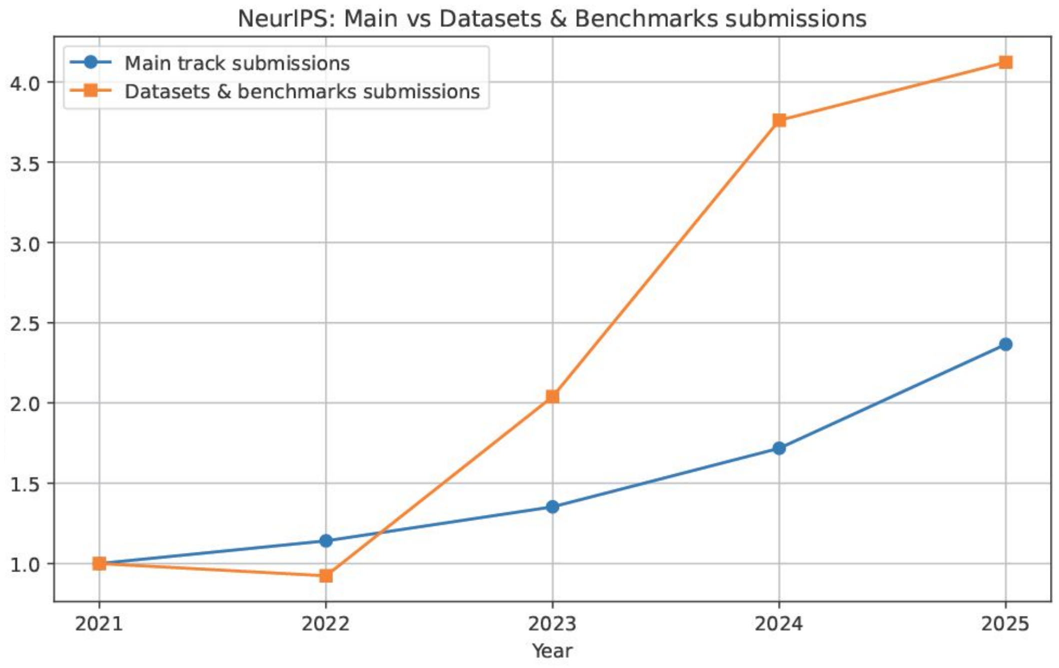
Many benchmarks followed ImageNet:

- Natural Language Processing: GLUE, SuperGLUE
- Reinforcement Learning: Atari, MuJoCo, OpenAI Gym
- Others: fastMRI, DAWNBench, MLPerf, etc.



➔ Benchmarks are now ubiquitous in AI research.

Too many benchmarks in AI?



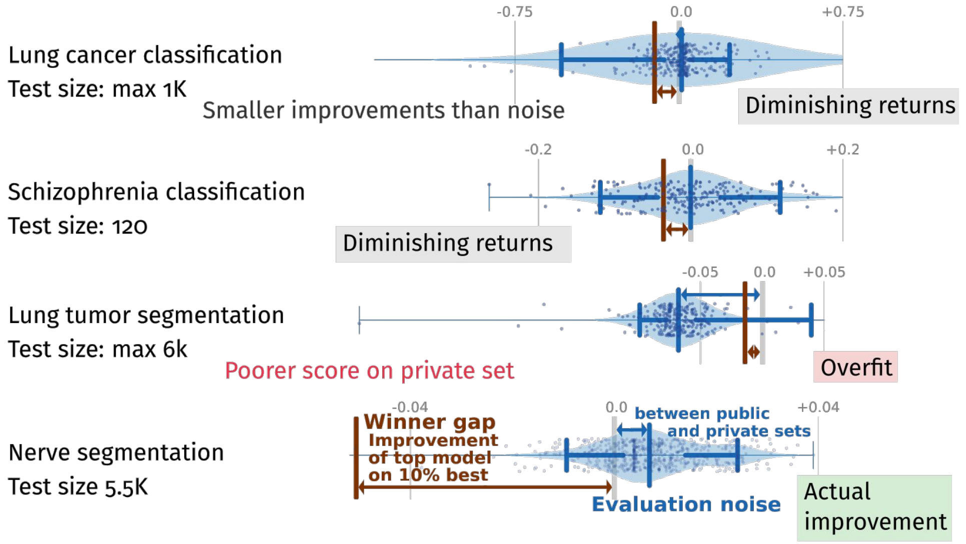
Benchmark goals in AI

	Short term progress	Long term progress
Task specific	Challenge/Competition ➔ push limits quickly	SOTA tracking ➔ measure progress
Generalizable	Research question ➔ empirical study	Benchmarking framework ➔ stable & extensible

➔ Most attention goes to the top-left quadrant for fast progress, but solid science requires the bottom-right.

Challenges in benchmarking AI

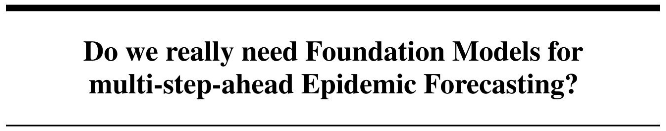
- Futile benchmarks



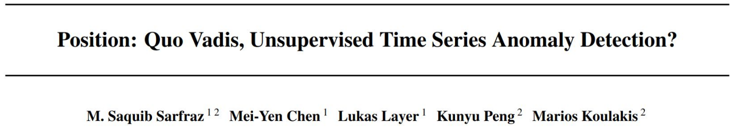
[Varoquaux and Cheplygina 2022]

Challenges in benchmarking AI

- Futile benchmarks
- Lack of proper baselines hinders scientific progress.

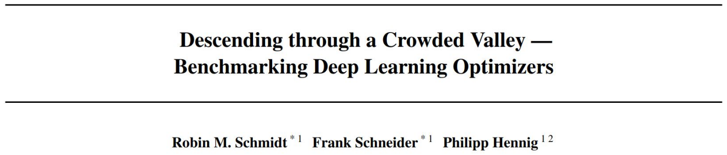


Do we really need Foundation Models for multi-step-ahead Epidemic Forecasting?



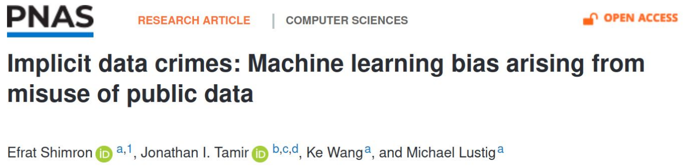
Position: Quo Vadis, Unsupervised Time Series Anomaly Detection?

M. Saquib Sarfraz[1 2] Mei-Yen Chen[1] Lukas Layer[1] Kunyu Peng[2] Marios Koulakis[2]



Descending through a Crowded Valley — Benchmarking Deep Learning Optimizers

Robin M. Schmidt[* 1] Frank Schneider[* 1] Philipp Hennig[1 2]



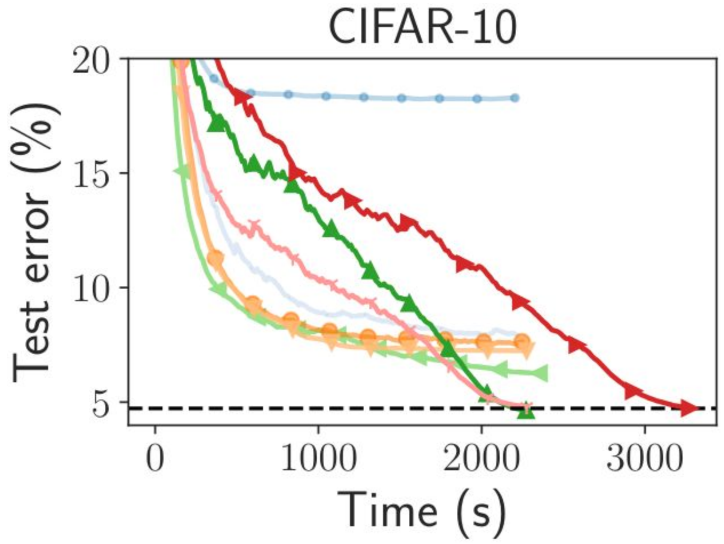
PNAS RESEARCH ARTICLE | COMPUTER SCIENCES OPEN ACCESS

Implicit data crimes: Machine learning bias arising from misuse of public data

Efrat Shimron[a,1], Jonathan I. Tamir[b,c,d], Ke Wang[a], and Michael Lustig[a]

Challenges in benchmarking AI

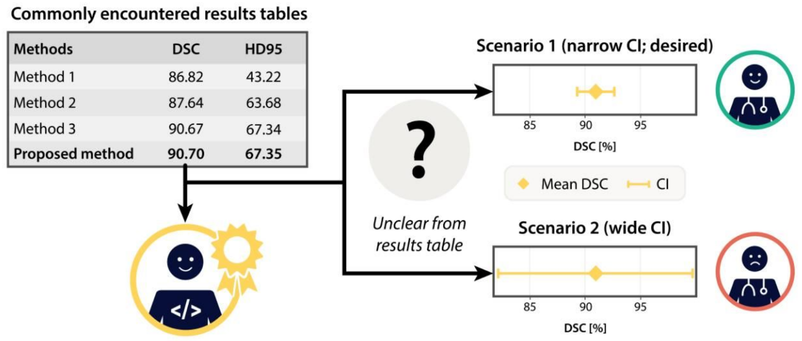
- Futile benchmarks
- Lack of proper baselines hinders scientific progress.
- Reproducing benchmarks is hard.



[Moreau et al. 2022]

Challenges in benchmarking AI

- Futile benchmarks
- Lack of proper baselines hinders scientific progress.
- Reproducing benchmarks is hard.
- Statistical validity is often missing.


Commonly encountered results tables

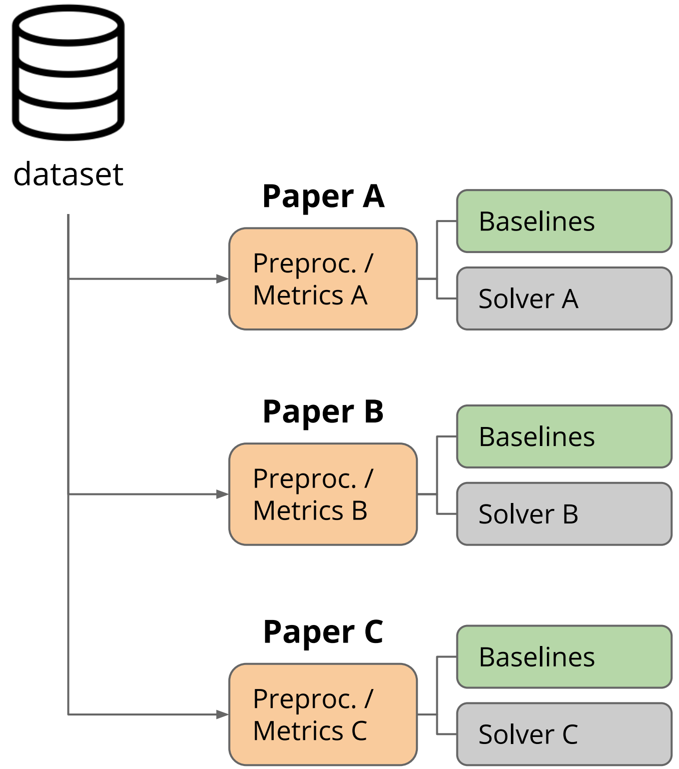
Methods	DSC	HD95
Method 1	86.82	43.22
Method 2	87.64	63.68
Method 3	90.67	67.34
Proposed method	**90.70**	**67.35**

Fig. 1: Common practice in medical imaging algorithm performance reporting leaves many open questions.

[Christodoulou et al. 2024]

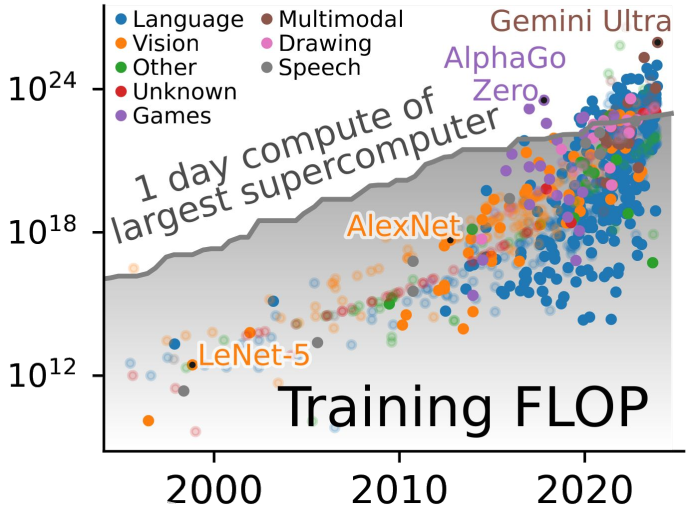
Challenges in benchmarking AI

- Futile benchmarks
- Lack of proper baselines hinders scientific progress.
- Reproducing benchmarks is hard.
- Statistical validity is often missing.
- Benchmarking cost is duplicated across groups.



Challenges in benchmarking AI

- Futile benchmarks
- Lack of proper baselines hinders scientific progress.
- Reproducing benchmarks is hard.
- Statistical validity is often missing.
- Benchmarking cost is duplicated across groups.



[Varoquaux et al. 2025]

Challenges in benchmarking AI

Technical challenges for repeatability:

- Multiple frameworks: PyTorch, Tensorflow, jax,...
- Hardware dependence: CPU, GPU, TPUs,...
- Large scale datasets: hard to share, pre-process,...

Challenges in benchmarking AI

Technical challenges for repeatability:

- Multiple frameworks: PyTorch, Tensorflow, jax,...
- Hardware dependence: CPU, GPU, TPUs,...
- Large scale datasets: hard to share, pre-process,...

Challenges for reusability:

- Usability: API, modularity, documentation, tests,...
- Maintenance: dependencies, code rot, long term support,...

Challenges in benchmarking AI

Technical challenges for repeatability:

- Multiple frameworks: PyTorch, Tensorflow, jax,...
- Hardware dependence: CPU, GPU, TPUs,...
- Large scale datasets: hard to share, pre-process,...

Challenges for reusability:

- Usability: API, modularity, documentation, tests,...
- Maintenance: dependencies, code rot, long term support,...

Statistical challenges for replicability:

- Stochastic algorithms: random initialization, data shuffling,...
- Data handling: splitting, data preprocessing, model selection,...

Running benchmarks with Benchopt

Benchopt provides a framework to organize and run benchmarks

➔ Moreau, Thomas, et al. (2022) "Benchopt: Reproducible, efficient and collaborative optimization benchmarks." In *NeurIPS*

Running benchmarks with Benchopt

Examples of existing benchmarks (https://github.com/benchopt):

- Image Classification (resnet)
- Logistic regression
- Lasso
- ICA
- Unsup. Domain Adaptation
- Bilevel Optimization
- Brain Computer Interface
- . . .

Running benchmarks with Benchopt

Clone the repo https://github.com/Etyl/benchmark_classification

Create virtual environment and install dependencies:

With conda:

```
conda create -n benchopt-clf python=3.11
conda activate benchopt-clf
python -m pip install benchopt
benchopt install . --env-name benchopt-clf
```

With venv:

```
python -m venv venv
source venv/bin/activate
python -m pip install -r requirements.txt
```

Running benchmarks with Benchopt

Running the benchmark

```
benchopt run .
```

Running benchmarks with Benchopt

Running the benchmark

```
benchopt run .
```

Running the benchmark with a config

```
benchopt run . --config run_config.yml
```

```
# run_config.yml

dataset:
  simulated

solver:
  svm:
    kernel: ["linear"]
  torch-logreg:
    lr: [0.01]
    batch_size: [64, 256]

n_repetitions: 3
timeout: 30
seed: 0
n-jobs: 4
```

Running benchmarks with Benchopt

Running the benchmark

```
benchopt run .
```

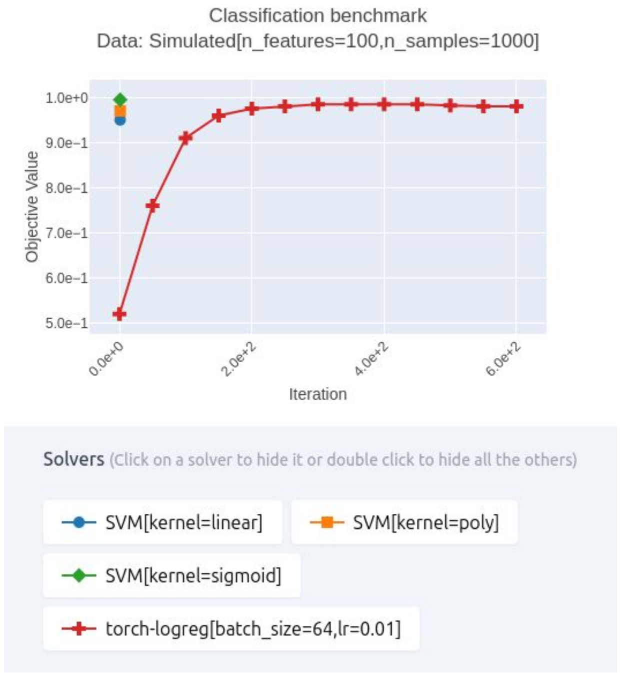
Running the benchmark with a config

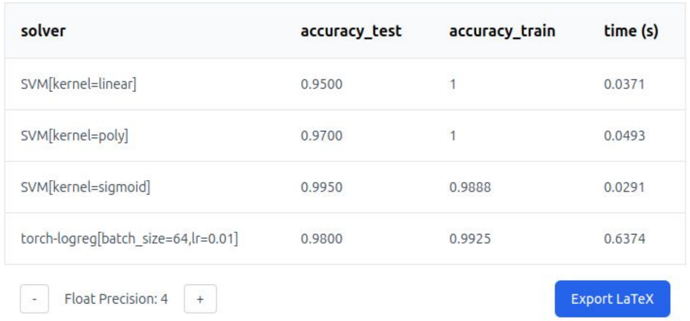
```
benchopt run . --config run_config.yml
```

➔ results are cached when running the same setup

```
# run_config.yml

dataset:
  simulated

solver:
  svm:
    kernel: ["linear"]
  torch-logreg:
    lr: [0.01]
    batch_size: [64, 256]

n_repetitions: 3
timeout: 30
seed: 0
n-jobs: 4
```

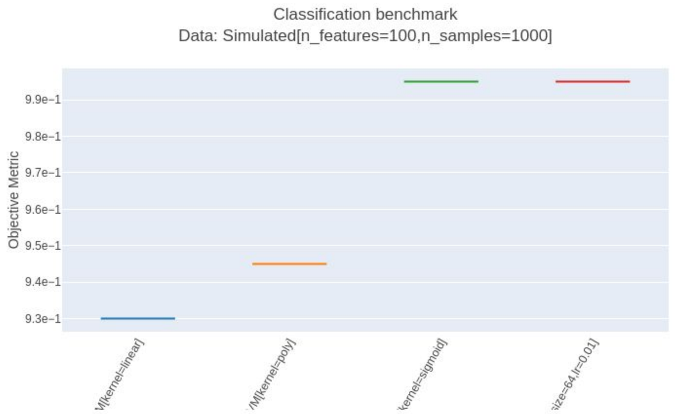
Interactive results exploration



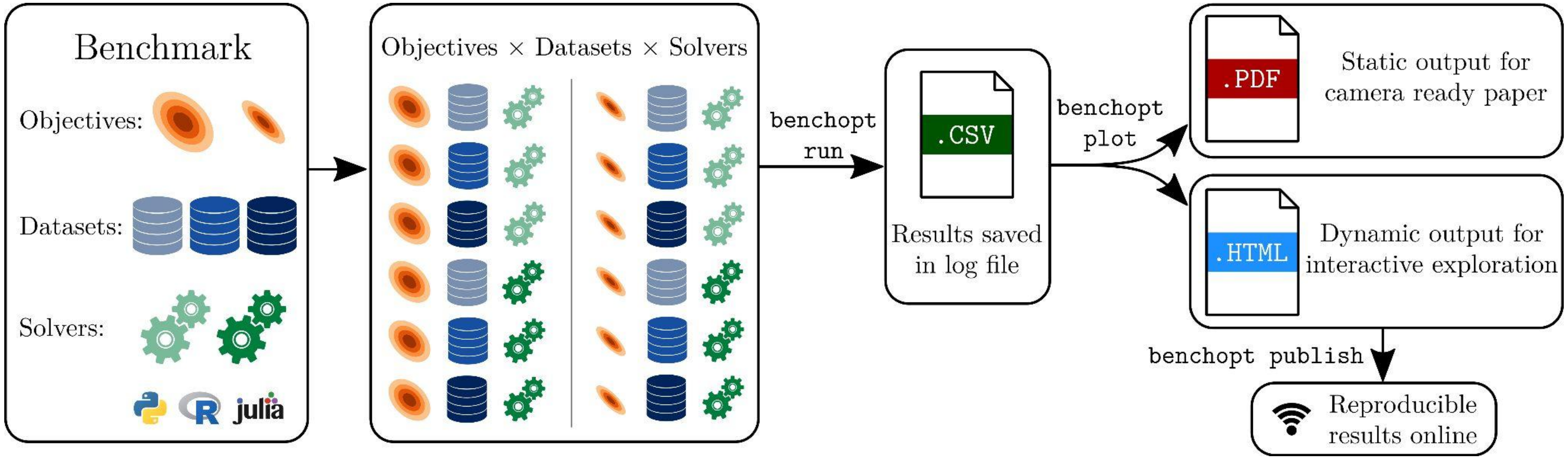


solver	accuracy_test	accuracy_train	time (s)
SVM[kernel=linear]	0.9500	1	0.0371
SVM[kernel=poly]	0.9700	1	0.0493
SVM[kernel=sigmoid]	0.9950	0.9888	0.0291
torch-logreg[batch_size=64,lr=0.01]	0.9800	0.9925	0.6374

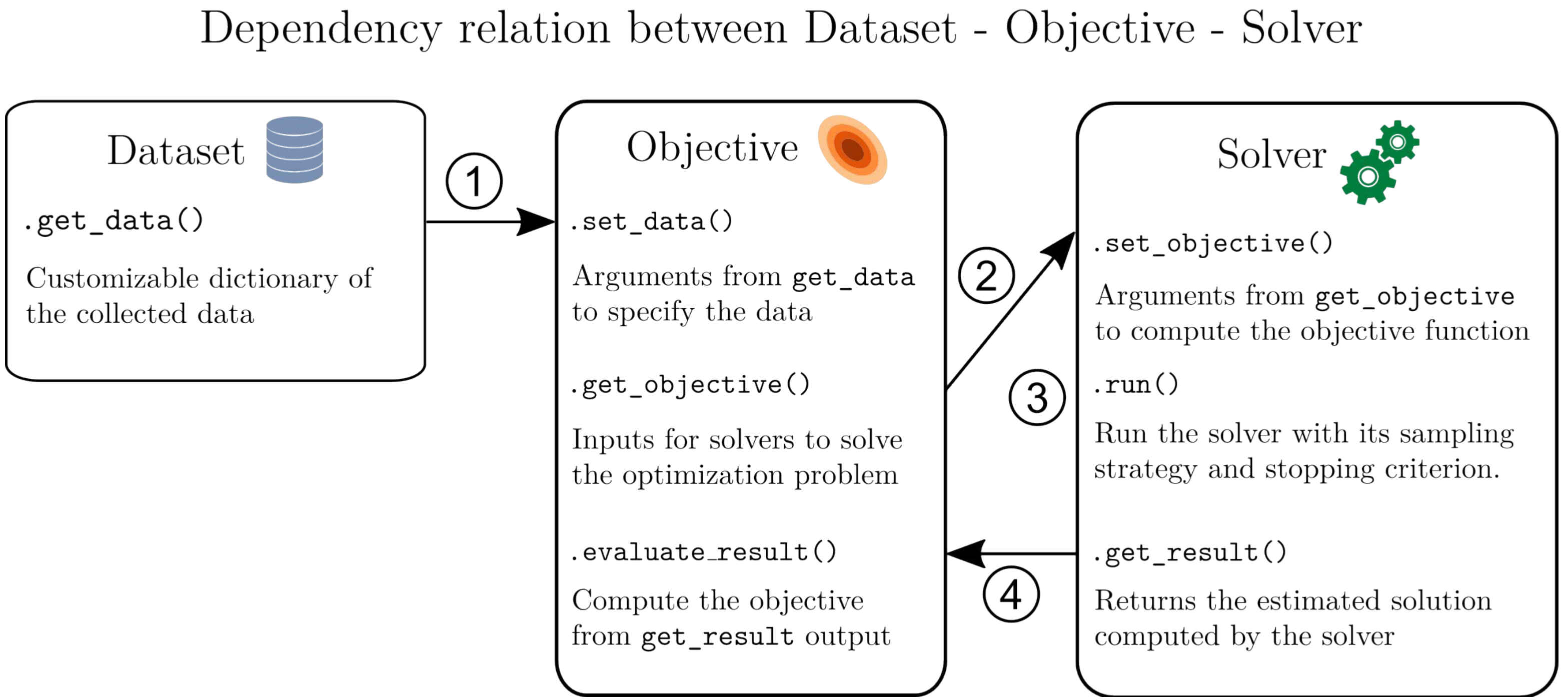
-
Float Precision: 4
+
Export LaTeX



A modular framework to create benchmarks


Benchmark
Objectives:
Datasets:
Solvers:
julia
Objectives × Datasets × Solvers
benchopt run
.CSV
Results saved in log file
benchopt plot
.PDF
Static output for camera ready paper
.HTML
Dynamic output for interactive exploration
benchopt publish
Reproducible results online

A modular framework to create benchmarks


Dependency relation between Dataset - Objective - Solver
Dataset
.get_data()
Customizable dictionary of the collected data
1
Objective
.set_data()
Arguments from get_data to specify the data
.get_objective()
Inputs for solvers to solve the optimization problem
.evaluate_result()
Compute the objective from get_result output
2
3
4
Solver
.set_objective()
Arguments from get_objective to compute the objective function
.run()
Run the solver with its sampling strategy and stopping criterion.
.get_result()
Returns the estimated solution computed by the solver

Structure of a benchmark

```
benchmark/
├── objective.py
├── datasets/
│   ├── dataset1.py
│   └── dataset2.py
├── solvers/
│   ├── solver1.py
│   └── solver2.py
└── plots/
     ├── custom_plot1.py
     └── custom_plot2.py
```

Adding a dataset

```python
class Dataset(BaseDataset):
   name = "Simulated"
   parameters = {
       'n_samples, n_features': [
           (1000, 100),
           (5000, 10),
       ],
   }
   requirements = []

   def get_data(self):
       X, y = make_classification(
           n_samples=self.n_samples, n_features=self.n_features,
           n_informative=1, n_redundant=0, n_clusters_per_class=1,
           random_state=self.get_seed()
       )
       return dict(X=X, y=y)
```

Adding a dataset

Add a new dataset (for example the breast cancer Wisconsin dataset from sklearn)

Adding a dataset

```python
from sklearn.datasets import load_breast_cancer

class Dataset(BaseDataset):
   name = "Breast Cancer"

   def get_data(self):
       X, y = load_breast_cancer(return_X_y=True)
       return dict(X=X, y=y)
```

Adding metrics

```
class Objective(BaseObjective):
   name = "Classification"
   url = "https://github.com/#ORG/#BENCHMARK"
   requirements = ['scikit-learn']

   def set_data(self, X, y):
       self.X, self.y = X, y

   def evaluate_result(self, predict):
       return dict(
           accuracy_train=accuracy_score(self.y, predict(self.X)),
       )

   def get_objective(self):
       return dict(
           X_train=self.X,
           y_train=self.y,
       )
```

Adding metrics

```python
class Objective(BaseObjective):
   name = "Classification"
   url = "https://github.com/#ORG/#BENCHMARK"
   requirements = ['scikit-learn']

   def set_data(self, X, y):
       self.X, self.y = X, y
       self.cv = KFold(
           n_splits=5, shuffle=True, random_state=self.get_seed()
       )
       self.cv_metadata = {}

   def evaluate_result(self, predict):
       return dict(
           accuracy_train=accuracy_score(self.y_train, predict(self.X_train)),
           accuracy_test=accuracy_score(self.y_test, predict(self.X_test)),
       )

   def get_objective(self):
       self.X_train, self.X_test, self.y_train, self.y_test = self.get_split(self.X, self.y)
       return dict(
           X_train=self.X_train,
           y_train=self.y_train,
       )
```

Adding metrics

Add a new metric (for example f1 score)

Adding metrics

```
from sklearn.metrics import f1_score

class Objective(BaseObjective):
   name = "Classification benchmark"
   url = "https://github.com/#ORG/#BENCHMARK"
   requirements = ['scikit-learn']

   def evaluate_result(self, predict):
       return dict(
           accuracy_train=accuracy_score(self.y_train, predict(self.X_train)),
           accuracy_test=accuracy_score(self.y_test, predict(self.X_test)),
           f1_test=f1_score(self.y_test, predict(self.X_test))
       )
```

Adding solvers

```python
class Solver(BaseSolver):
   name = 'SVM'
   parameters = {
       'kernel': ['linear', 'poly', 'sigmoid'],
   }
   sampling_strategy = "run_once"

   def set_objective(self, X_train, y_train):
       self.X_train, self.y_train = X_train, y_train
       self.clf = SVC(kernel=self.kernel)

   def run(self, _):
       self.clf.fit(self.X_train, self.y_train)

   def get_result(self):
       return dict(predict=self.clf.predict)
```

Adding solvers

```python
class Solver(BaseSolver):
    name = 'torch-logreg'
    parameters = {'lr': [0.01], 'batch_size': [64]}
    requirements = ['pytorch:pytorch']

    stopping_criterion = SufficientProgressCriterion(
        strategy="callback", eps=1e-3, patience=3,
        key_to_monitor="accuracy_test", minimize=False
    )

    def get_next(self, stop_val):
        return stop_val + 50

    def run(self, callback):
        optim = torch.optim.SGD(self.clf.parameters(), lr=self.lr)
        loss_fn = nn.CrossEntropyLoss()
        max_epochs = 100
        for _ in range(max_epochs):
            for X_batch, y_batch in self.dataloader:
                optim.zero_grad()
                y_pred = self.clf(X_batch)
                loss = loss_fn(y_pred, y_batch)
                loss.backward()
                optim.step()
                if not callback():
                    return
```

Adding solvers

Add a new solver (for example Logistic Regression from sklearn) or improve an existing one, by adding new parameters (optimiser, regularisation, batch size, etc.)

Adding plots

```python
class Plot(BasePlot):
   name = "Accuracy"
   type = "bar_chart"  # Or "scatter", "boxplot", "table"
   options = {"dataset": ..., "objective": ..., "split": ["train", "test"]}

   def plot(self, df, dataset, objective, split):
       df = df.query("dataset_name == @dataset and objective_name == @objective")
       plots = []
       for solver_name, df_filtered in df.groupby('solver_name'):
           df_filtered = df_filtered.select_dtypes(include=['number'])
           stop_val = df_filtered['stop_val'].max()
           this_df = df_filtered[df_filtered['stop_val'] == stop_val]
           plots.append({
               "y": this_df[f"accuracy_{split}"].tolist(),
               "text": "",
               "label": solver_name,
               "color": self.get_style(solver_name)["color"]
           })
       return plots

   def get_metadata(self, df, dataset, solver, objective, objective_column):
       return {
           "title": f"{objective}\nData: {dataset}\nSolver: {solver}",
           "ylabel": objective_column,
       }
```

Adding plots

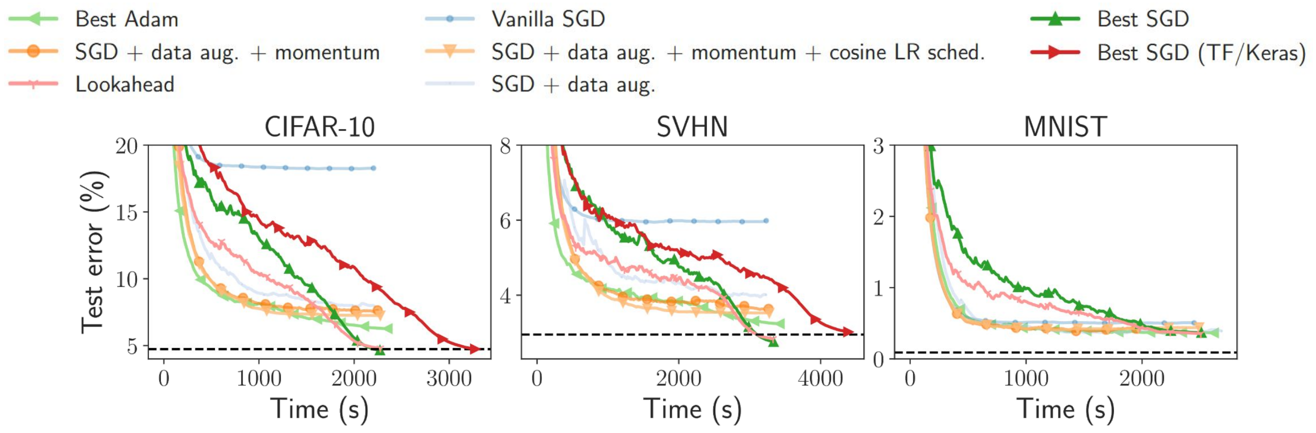
More info on plots:
https://benchopt.github.io/stable/user_guide/add_custom_plot.html

Benchopt makes your life easy

- Build on previous benchmarks
- Use solvers in Python, R, Julia, binaries...
- Monitor any metric you want altogether (test/train loss, ...)
- Add parameters to solvers and use config files
- Share and publish HTML results
- Run all benchmarks in parallel, on HPC clusters...
- Cache runs and results
- and much more!

Example: Optimisation for ResNet on image classification

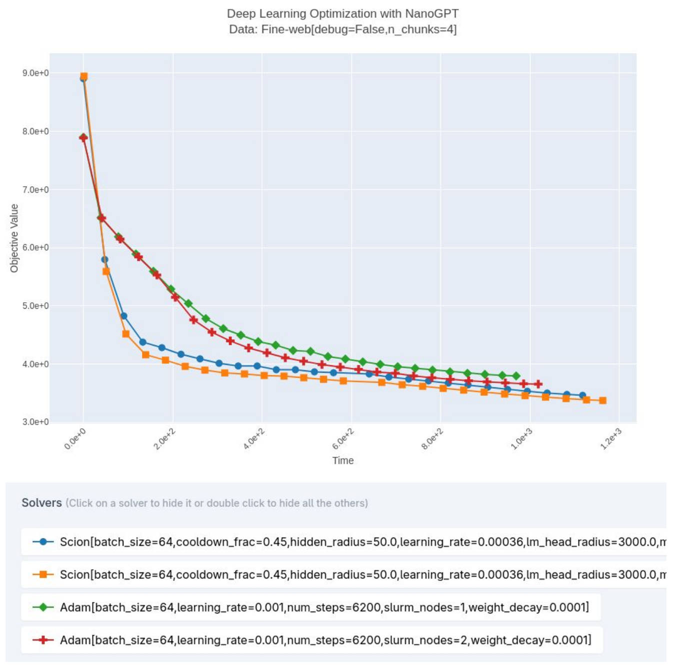
- Image classification with resnet18
- Various optimization strategies
- Compare pytorch and tensorflow
- Publish reproducible SOTA for baselines



https://github.com/benchopt/benchmark_resnet_classif/

Example: Large scale-optimisation for Deep learning

- Use modern large-scale datasets and models
- Classical optimization tricks
- Distributed training and mixed precision



Solvers (Click on a solver to hide it or double click to hide all the others)

- ● Scion[batch_size=64,cooldown_frac=0.45,hidden_radius=50.0,learning_rate=0.00036,lm_head_radius=3000.0,n
- ■ Scion[batch_size=64,cooldown_frac=0.45,hidden_radius=50.0,learning_rate=0.00036,lm_head_radius=3000.0,n
- ◆ Adam[batch_size=64,learning_rate=0.001,num_steps=6200,slurm_nodes=1,weight_decay=0.0001]
- ✚ Adam[batch_size=64,learning_rate=0.001,num_steps=6200,slurm_nodes=2,weight_decay=0.0001]

`https://github.com/benchopt/benchmark_nanogpt`

Mini Project: Creating a benchmark

Some ideas:

1. MNIST classification
   - Create the dataset (using MNIST from sklearn)
   - Create deep learning solvers

2. Inverse Problems (deepinv)
   - Create the datasets for a specific inverse problem
   - Benchmark deepinv solvers

3. Optimization (Lasso)
   - Create the datasets for regression or classification problems
   - Benchmark different optimisers

You can also come up with your own ideas/benchmark, and we will also help you!