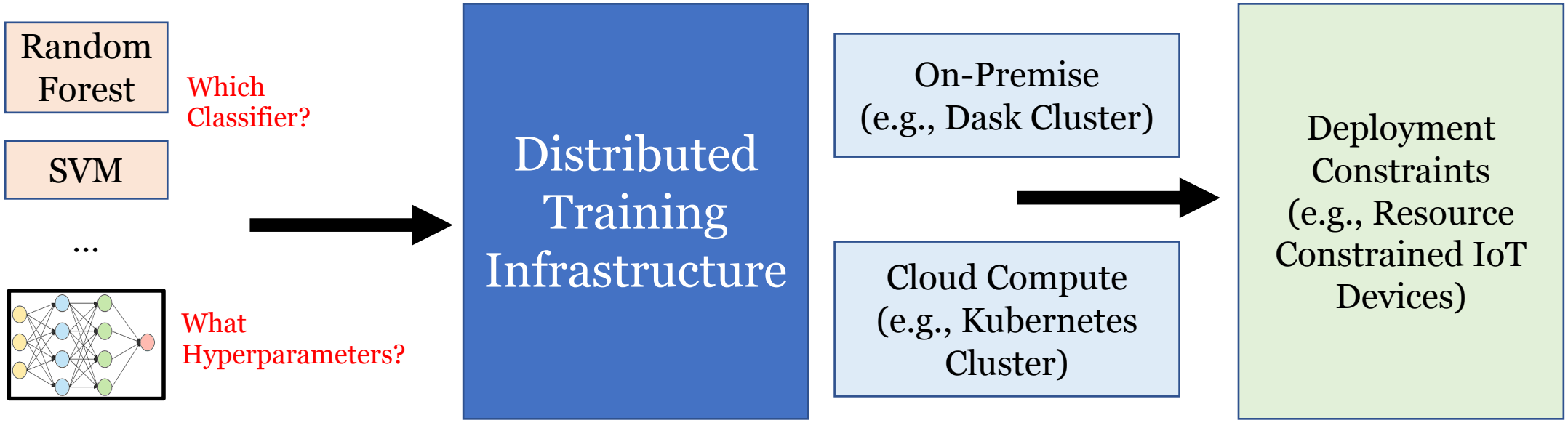
Enabling Hyperparameter Tuning of Machine Learning Classifiers in Production

Sandeep Singh Sandha, Mohit Aggarwal, Swapnil Sayan Saha & Mani Srivastava

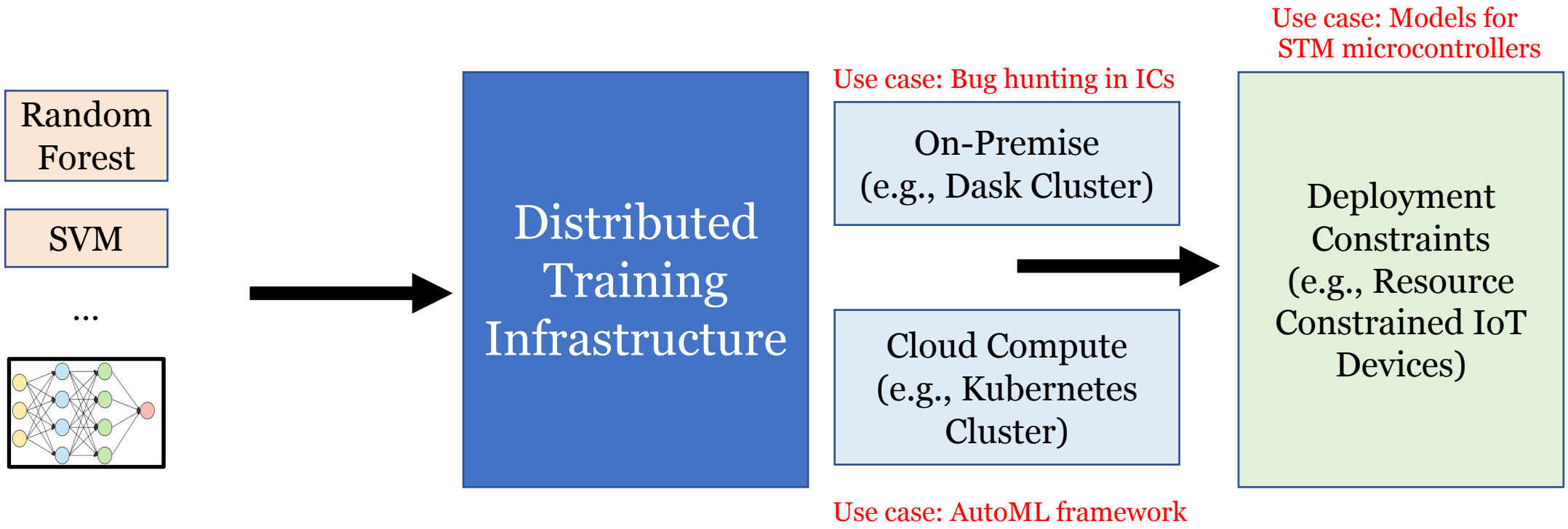
University of California, Los Angeles; Arm Research

https://github.com/ARM-software/mango

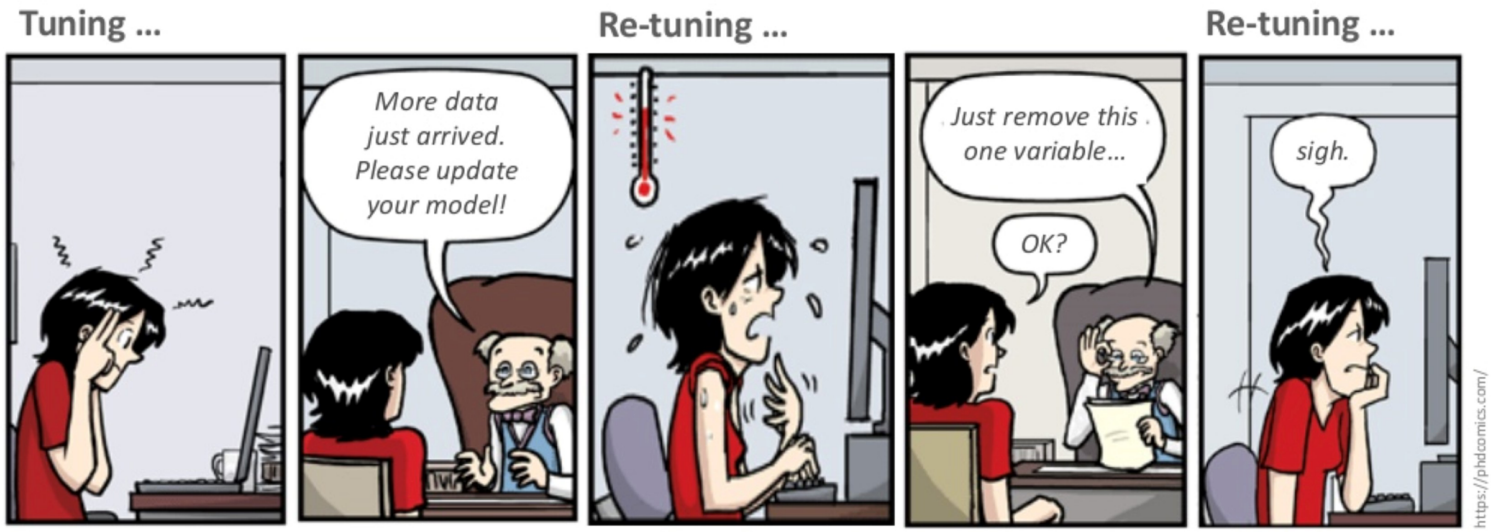
Production ML Pipelines



Production ML Pipelines



Tuning Hyperparameters is Crucial



Credits: "Piled Higher and Deeper"
by Jorge Cham, PhD Comics

Classifier Performance: Very sensitive to the choice of hyperparameters

Challenges to do Hyperparameter Tuning in Production

- **Complex deployments**
  - Production ML pipelines deployed on arbitrary infrastructures
    - Local custom schedulers, cloud computing
  - Runtime failures

- **Large complexity of search space**
  - Several choices of classifiers and their hyperparameters

Challenges in Adopting Existing Libraries/Tools

Existing libraries: Software dependencies hinder adoption

- Hyperopt[1], SMAC [2], Optuna [3], Spearmint [4], Auto-Sklearn [5]
- Scheduler abstraction are integrated with algorithms
- Schedulers don't support autoscaling, failovers

- E.g.,
  - Hyperopt: Parallel search depends on MongoWorkers or Apache Spark
  - SMAC: Parallel search require shared file system for workers
  - Optuna/Spearmint: Require a database for workers to run
  - Auto-Sklearn: Provides wrapper around SMAC for Scikit-learn

[1] https://github.com/hyperopt/hyperopt
[2] https://github.com/automl/SMAC3
[3] https://github.com/optuna/optuna
[4] https://github.com/HIPS/Spearmint
[5] https://github.com/automl/auto-sklearn

Challenges in Adopting Existing Libraries/Tools

Existing Tools: High overhead in adoption

- Katib [1], Polyaxon [2]: Require Kubernetes
- Tune [3]: Requires Ray
- Dask-ml [4]: Requires Dask cluster
- Expose algorithms from previous libraries on a particular underlying frameworks
- High overhead in adoption unless underlying system is also used for its complete capabilities
  - E.g., Katib requires Kubernetes, API Server, database, controller process

[1] https://github.com/kubeflow/katib
[2] https://github.com/polyaxon/polyaxon
[3] https://github.com/ray-project/ray/tree/master/python/ray/tune
[4] https://ml.dask.org/

We introduce Mango

https://github.com/ARM-software/mango

An opensource hyperparameter tuning library deployed in production for 25+ months at Arm

- Flexible architecture
  - Not dependent on any database, scheduling framework, or …
  - Deployed on different frameworks in production (Dask, Kubernetes)
  - Expose capabilities to consider failures
- Parallel search
  - Provides state-of-the-art algorithms based on Bayesian optimization
  - Include recent research advancement that beat existing libraries
- CASH: Search across classifier

Mango Example

Search Space: Python constructs
- Compatible with Scikit-learn
- Support all 60+ distributions from Scipy

Scheduler: Serial, Parallel jobs, Custom

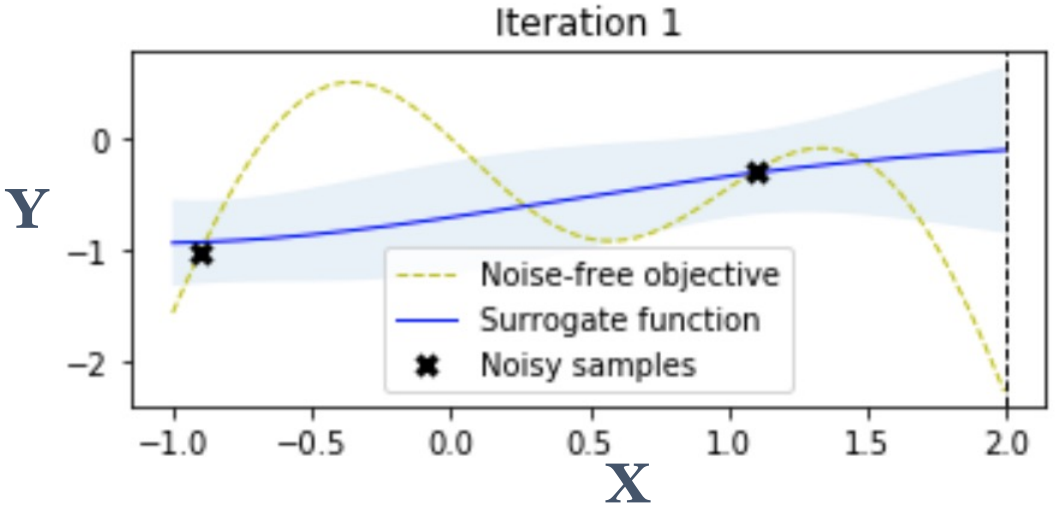
Optimizer: Maximize/Minimize

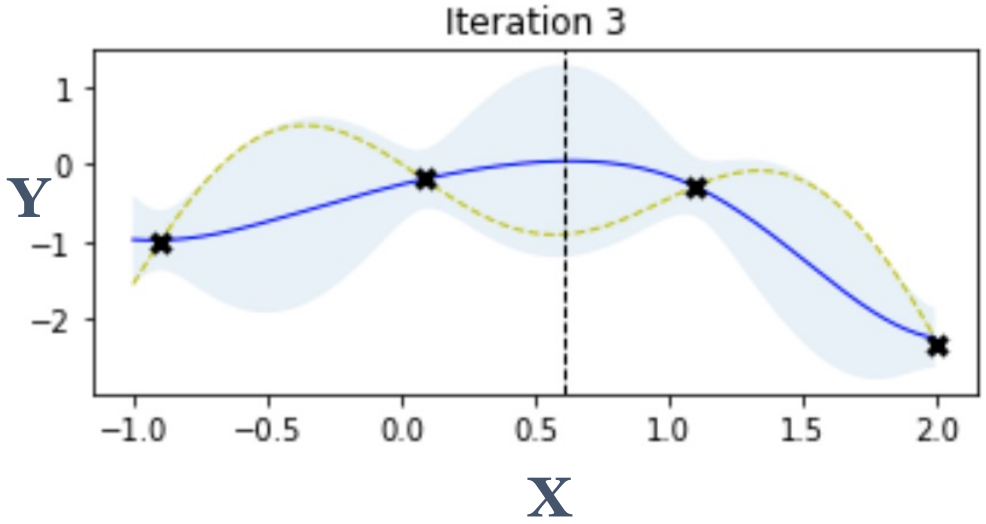
```
1  from mango import Tuner, scheduler
2  from scipy.stats import uniform
3  from xgboost import XGBClassifier
4  ...
5  param_space = {'learning_rate': uniform(0, 1),
6                 'gamma': uniform(0, 5),
7                 'max_depth': range(1, 21),
8                 'n_estimators': range(1, 11),
9                 'booster':['gbtree','gblinear','dart']}
10
11 @scheduler.parallel(n_jobs=4)
12 def objective(**params):
13     ...
14     clf = XGBClassifier(**params)
15     accuracy = ...
16     return accuracy
17 tuner = Tuner(param_dict, objective)
18 Study = tuner.maximize()
```

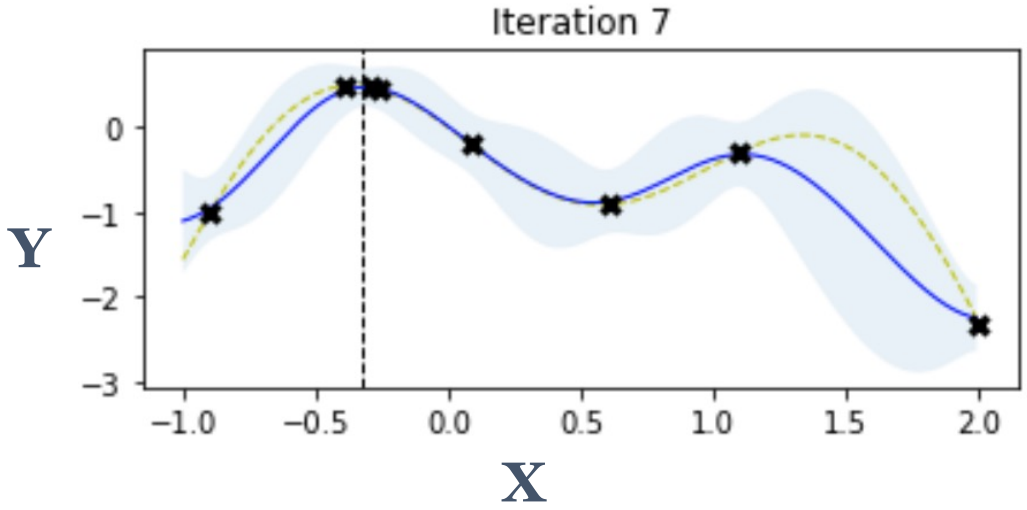
Algorithms in Mango

- *Optimizer*: Multi-armed bandit Bayesian optimizer
- *Surrogate function*: Gaussian Process **(Internal approximator)**
- *Acquisition function*: Created from Surrogate function **(Good regions to explore)**

Gaussian Process: Visualization





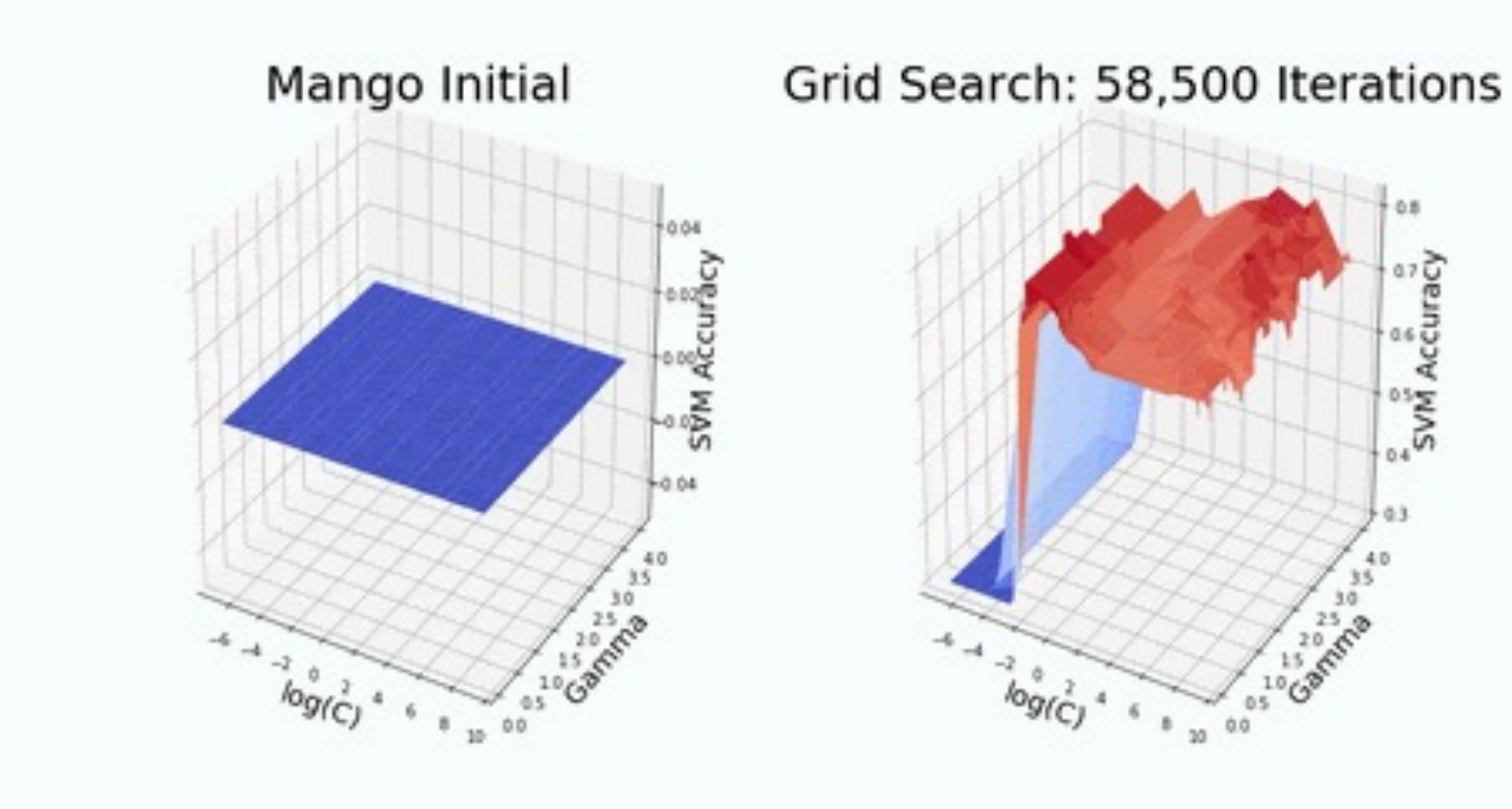


Mango: Minimize the Classifier Evaluations

Classifier: SVM
Dataset: Iris Dataset (2 features)

Mango Accuracy: ~83.5%

Grid Search Accuracy: ~83%



https://www.youtube.com/watch?v=hFmSdDLLUfY

Mango: 50 iterations

Algorithms in Mango

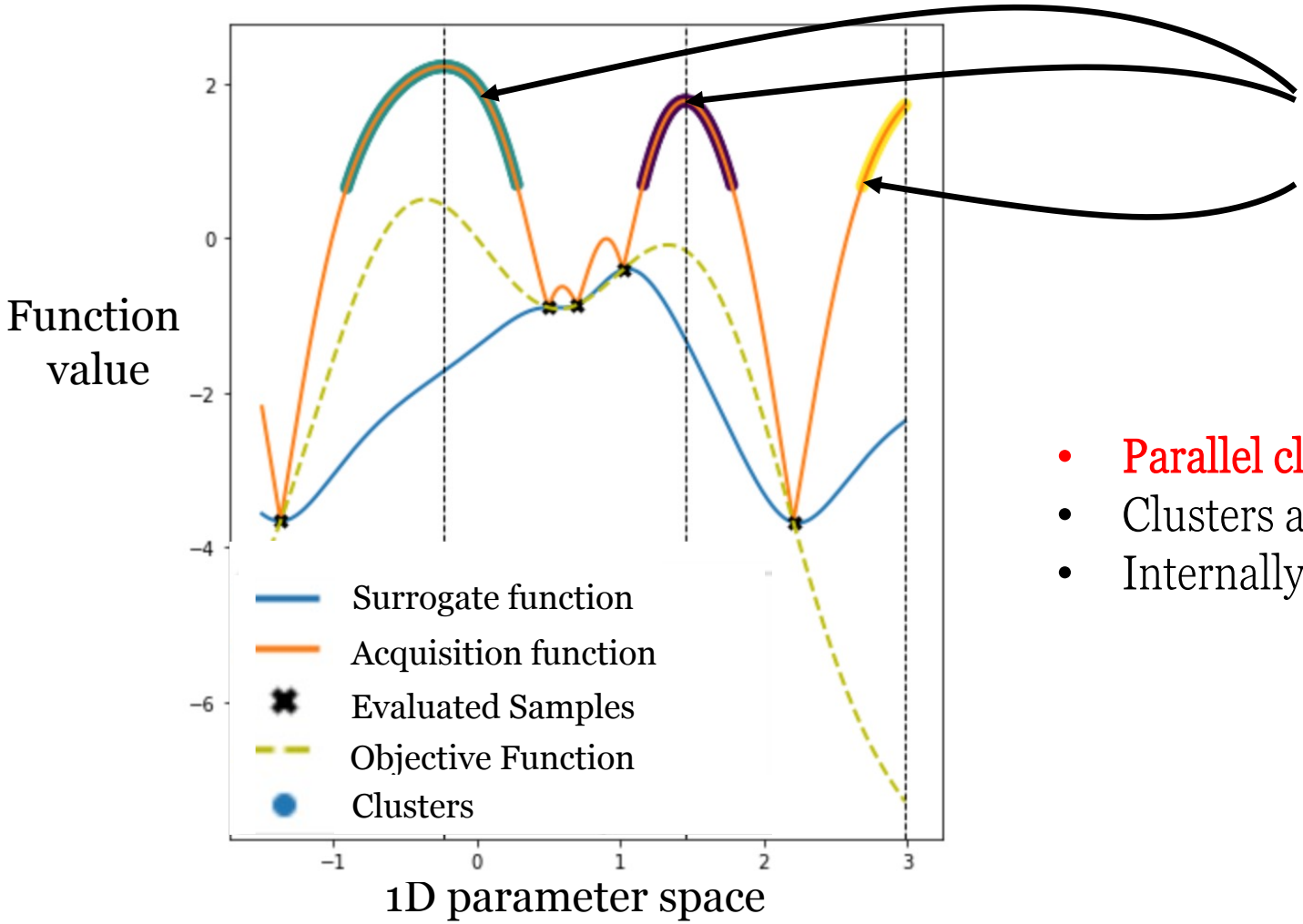
Mango address challenges with using the Gaussian Process

- Handling categorical variables [1]:
  - One-hot encoded with Monte-Carlo sampling for optimization
  - Gradient free optimizer: Evaluate Gaussian process only at the correct regions

- Parallel search:
  - Clustering-based [2,3]
  - Penalty-based [4]

[1] Garrido-Merchán, Eduardo C., and Daniel Hernández-Lobato. "Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes." *Neurocomputing* 380 (2020): 20-35.
[2] Groves, Matthew, and Edward O. Pyzer-Knapp. "Efficient and scalable batch Bayesian optimization using K-means." *arXiv preprint arXiv:1806.01159* (2018).
[3] Nguyen, Vu, et al. "Budgeted batch Bayesian optimization." *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016.
[4] Desautels, Thomas, Andreas Krause, and Joel W. Burdick. "Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. " *Journal of Machine Learning Research* 15 (2014): 3873-3923.

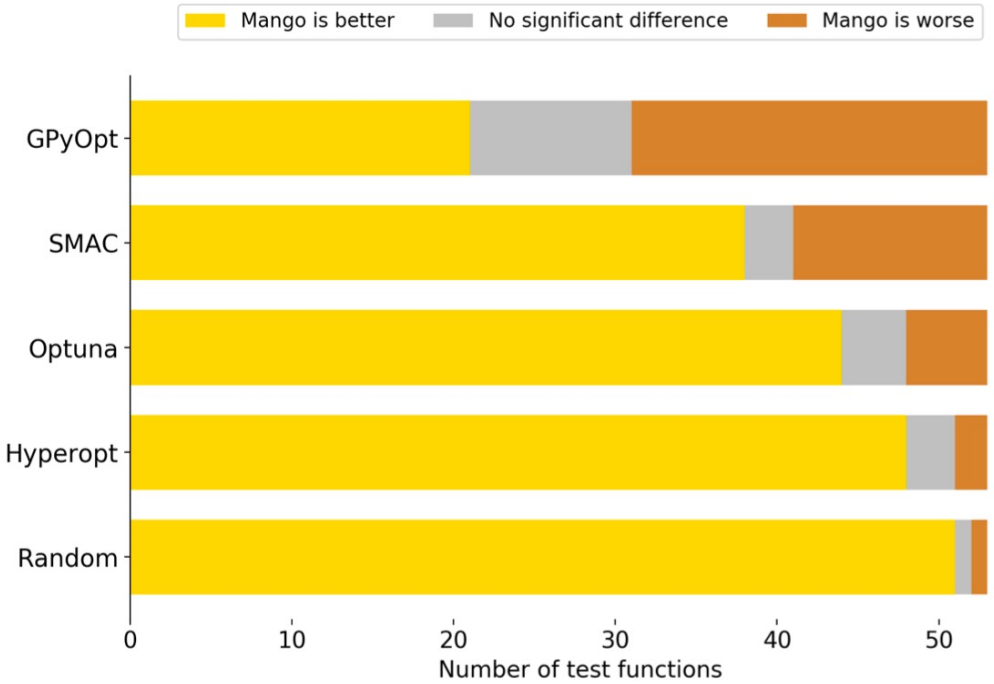
Parallel Search Clustering



3 cluster formed from where points can be explored in parallel to find optimal

- Parallel clusters from the acquisition function
- Clusters are in different promising regions
- Internally explores regions from different clusters

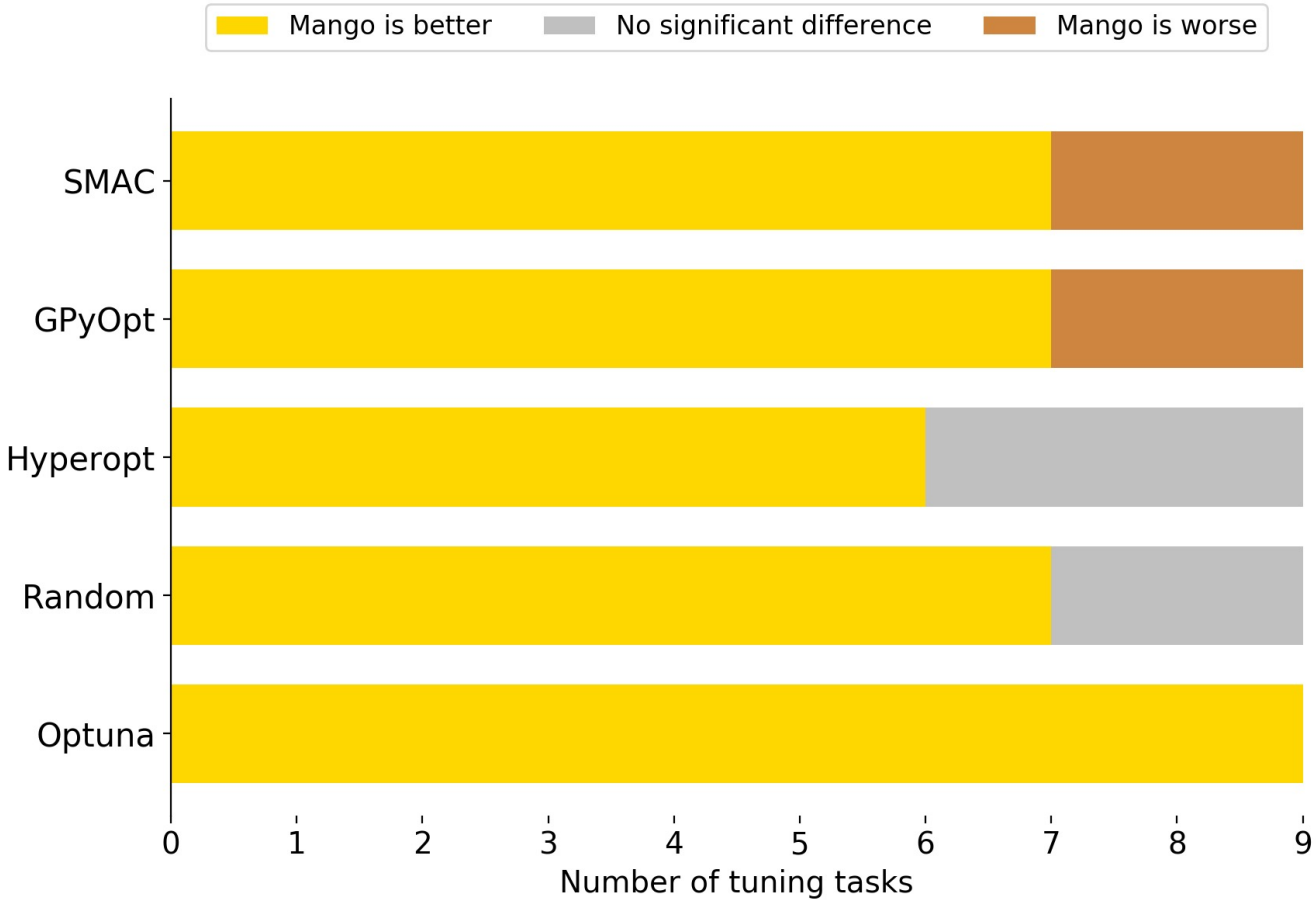
Comparison of Mango: Functional Benchmark



- 53 continuous proposed functions to compare optimizers [1, 2]
- 80 iterations and repeated 30 times
- Multiple criteria methodology proposed by [1]
- Performance by the solution's proximity to the optimal point (accuracy) and the number of iterations required to reach the optima (speed)

[1] Dewancker, Ian, et al. "A strategy for ranking optimization methods using multiple criteria." Workshop on Automatic Machine Learning. PMLR, 2016.
[2] https://github.com/optuna/optuna

Comparison of Mango: Classifiers



- **Classifiers**: SVM, Xgboost, KNN
- **Datasets**: Iris plants, wine recognition, and breast cancer Wisconsin [1]

[1] https://scikit-learn.org/

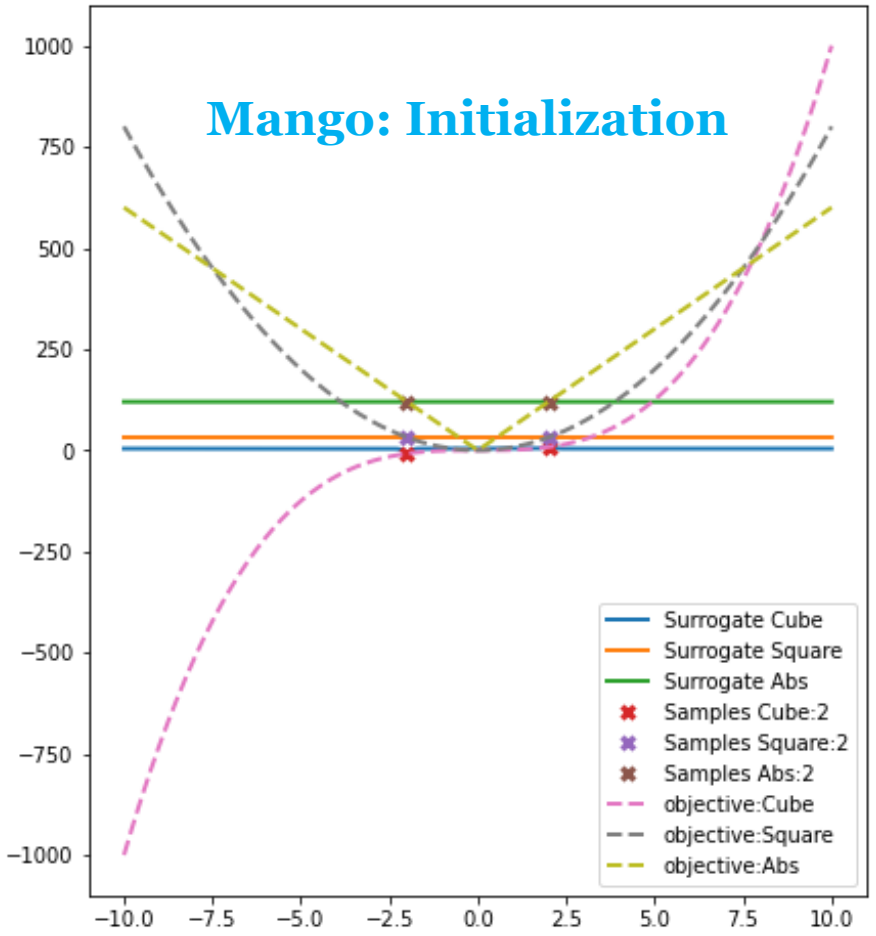
CASH: Support in Mango

CASH Problem [1]

- Which classifier to use?
- We still need to find the best optimal region/accuracy?
- Prefer better classifier early on

[1] Thornton, Chris, et al. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms." *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013.

Prioritize Objectives Directly

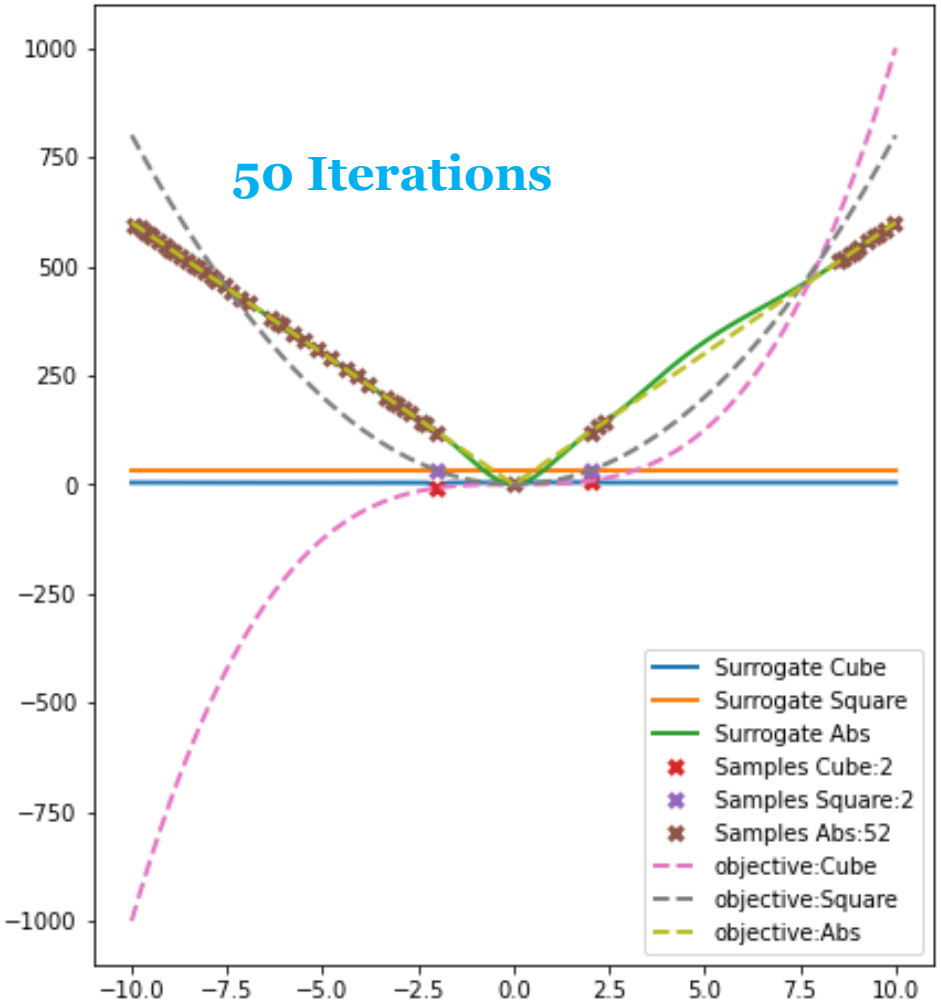


Functions: Cube, Abs, Square

Initialization: Bad Regions

Directly using multiple Surrogates: GP

Prioritize Objectives Directly



Functions: Cube, Abs, Square

Initialization: Bad Regions

Directly using multiple Surrogates: GP

Mango's CASH (*MetaTuner*) Algorithm

exploration = 1.0, *decay* = 0.9, *min_exploration* =0.1

Initialization:

1. do random evaluations
2. initialize surrogate for all objectives

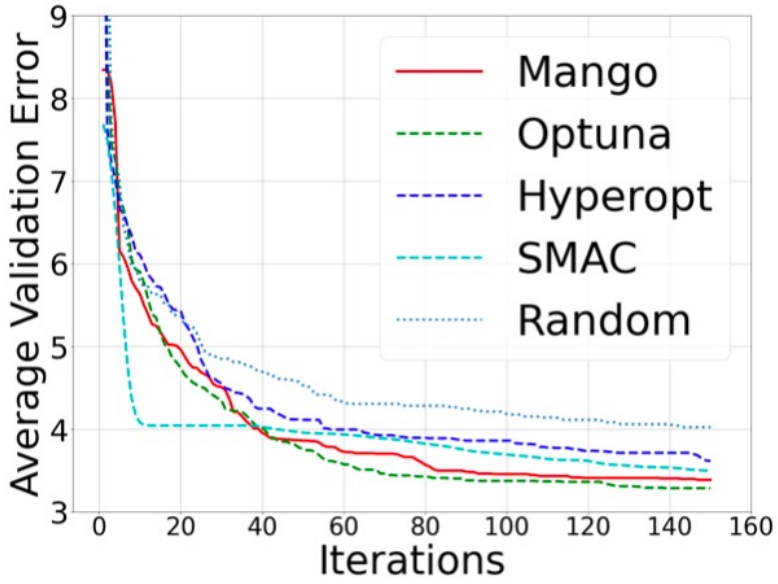
Main Logic:

1. *repeat* step 2 to 6 until max-iterations
2. *sample* *p* = random(0,1)
3. *if* *p*< exploration:
   - *evaluate* next objective randomly
   - exploration = exploration* *decay*
   - *if* exploration< *min_exploration:*
     - exploration = *min_exploration*
4. *else* sample next promising surrogates from all objectives
   - *evaluate* the objective with best surrogate
5. *scale* all the surrogates to same range
6. *update* the surrogates

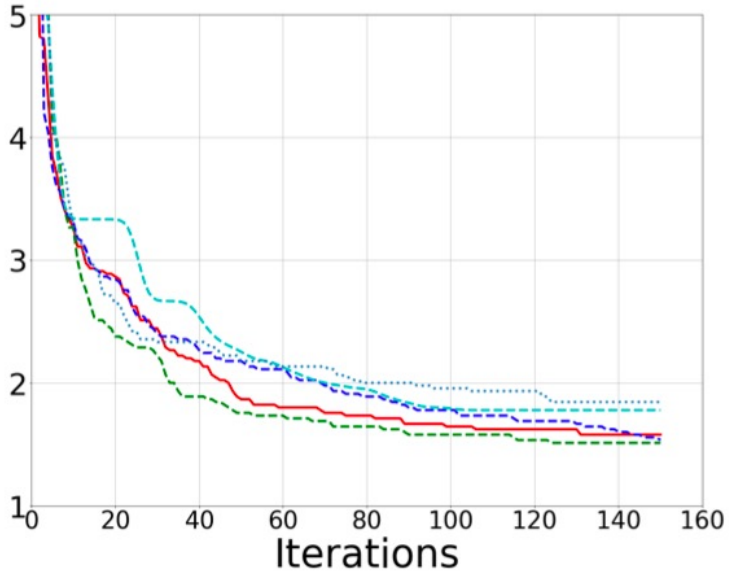
Motivated from [1] and uses **epsilon greedy** policy of exploration

[1] Jenatton, Rodolphe, et al. "Bayesian optimization with tree-structured dependencies." *International Conference on Machine Learning*. PMLR, 2017.

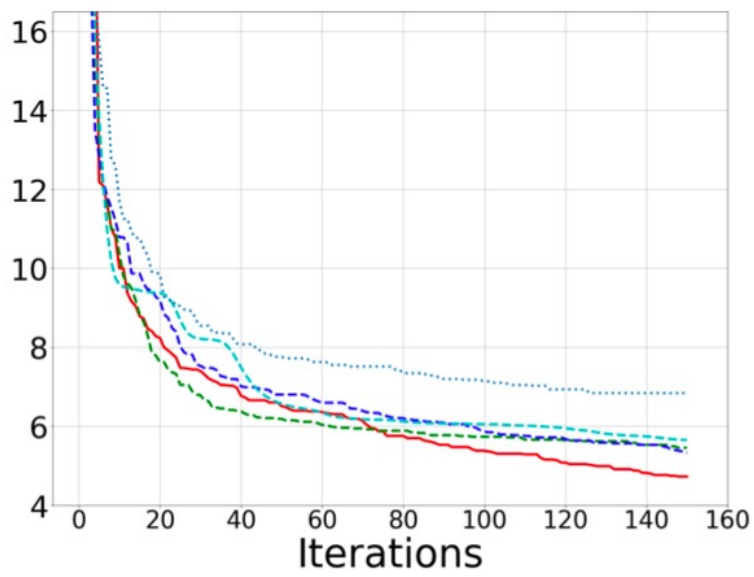
MetaTuner Comparisons



Breast Cancer Dataset



Iris Plan Dataset

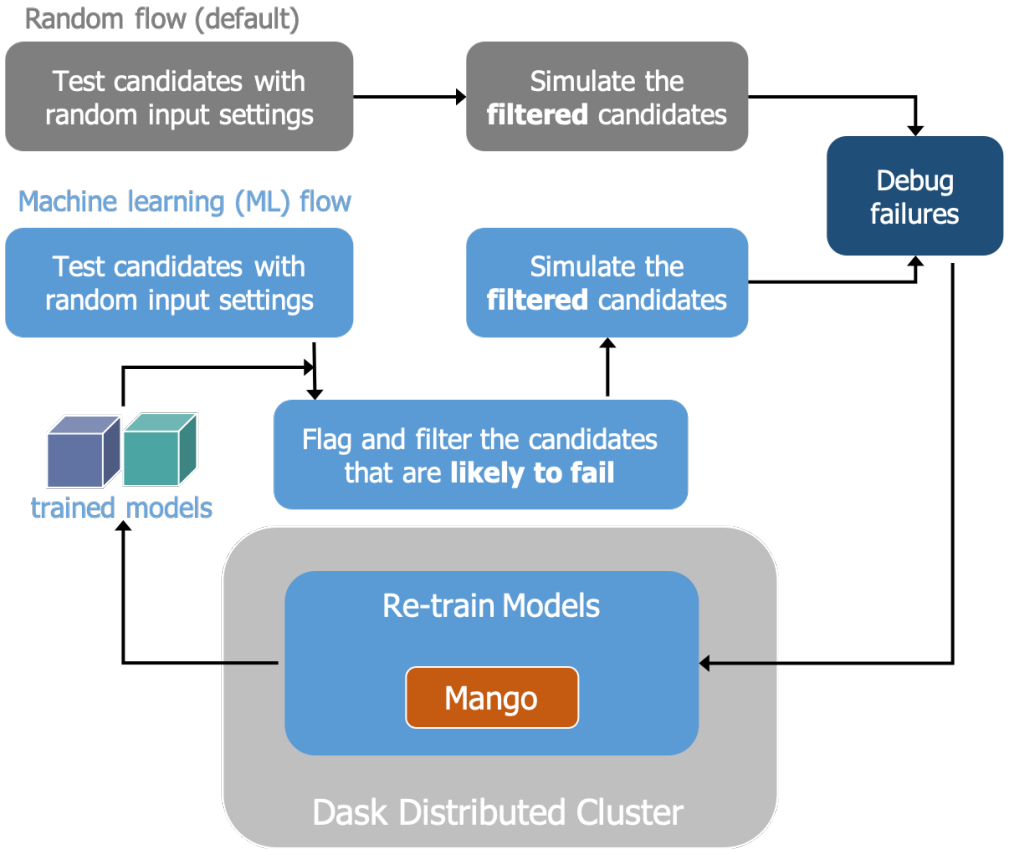


Wine Dataset

- **Classifiers**: NN (3-layer), KNN, Xgboost, Decision-Tree, SVM
- **Mango** is competitive in performance to the current state-of-the art

Mango Use case: Design verification of Arm ICs

50% of the entire compute for new design is used for design verification



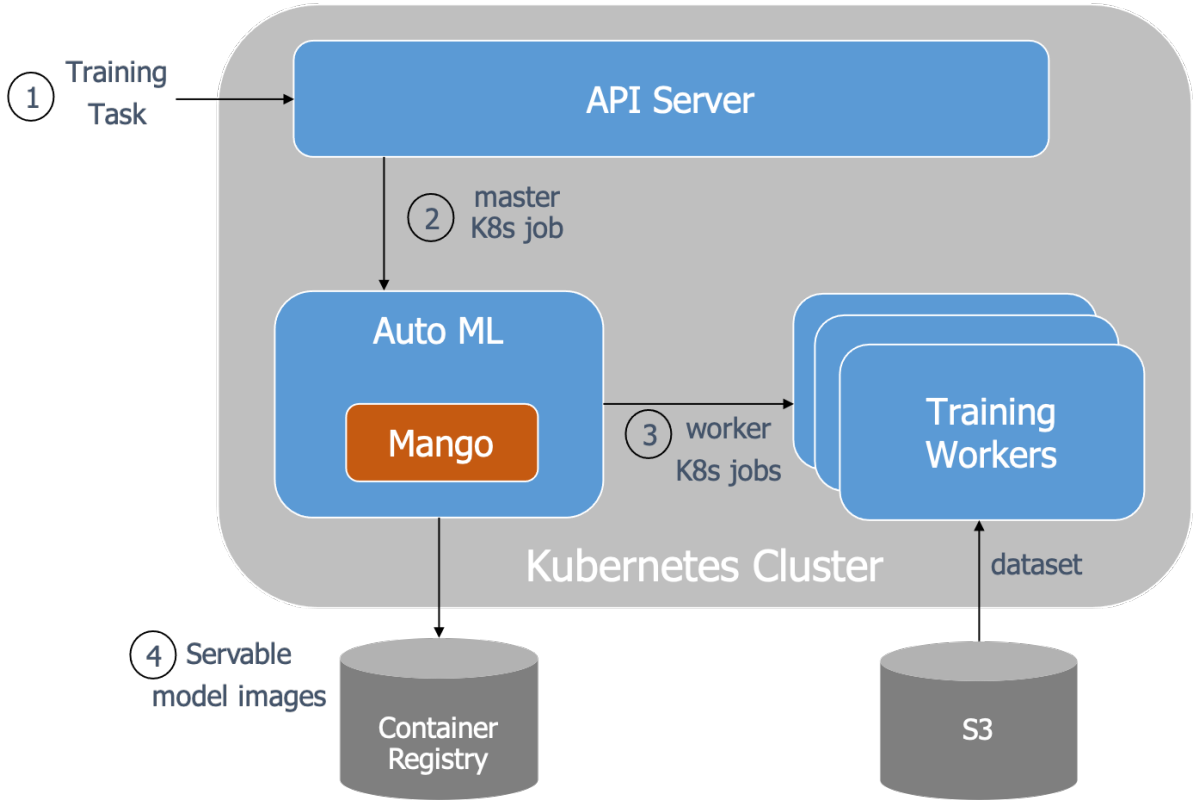

```python
from dask.distributed import Client
...
dask_client = Client()

param_clf_nn = {'type': 'clf_nn',...}
param_clf_svm = {'type': 'clf_svm',...}
param_spaces = [param_clf_nn, param_clf_svm]

def objective(params_batch):
    futures = []
    # Submit Jobs to the Dask cluster
    for params in params_batch:
        #schedule classifier based on the parameter type
        clf = params.pop('type')
        future = dask_client.submit(fit_and_score,
                                    clf, **params)
        futures.append(future)
    # Job completion or wait for timeout
    results = [future.result(timeout) for future in
               futures]
    return results
metatuner = MetaTuner(objective, param_spaces)
Study = metatuner.maximize()
```

Mango improved compute cycles on average by ~45% over the previous deployment

Mango Use case: AutoML Framework at Arm




```python
from kubernetes import client
...
param_space = ...
def objective(params_batch):
     # train on cluster using the sampled parameters
    jobs = [client.create_job(params, ...)
            for params in params_batch]
    # poll for job completion
    results = []
    while not timeout or not all_done:
        results = [job.result() for job in jobs
                   if job.complete()]
    return results
# control the max number of iterations, batch size ...
conf = {'num_iteration':100, 'initial_random':5, '
     batch_size':4, 'parallel':'clustering'}
tuner = Tuner(objective, param_space,conf)
Study = tuner.maximize()
```

Used to train different ML models on Kubernetes cluster at Arm by non-ML experts

Mango Use case: Training NN Models for Resource Constrained IoT Devices

- STM32F446RE: SRAM 128 kB, Flash 512 kB
- STM32L476RG: SRAM 128 kB, Flash 1 MB
- STM32F746ZG: SRAM 320 kB, Flash 1 MB

```
...
param_space = Neural Network Search Space
...
def objective(params_batch):
    #Step-1: Create NN with params_batch
    #Step-2: Get Resource of NN on a real-hardware
    #Step-3: Train if feasible and validate
    f_opt = ()f_error + ()f_RAM + ()f_flash
    return f_opt

tuner = Tuner(objective, param_space,conf)
...
```

Goal: Maximize the RAM and Flash usage with neural network still able to run on the device

Scenario: Neural networks trained to do Odometry
- Track humans using inertial sensors (Accelerometer and Gyro)
- Backbone Neural Network: TCN

https://github.com/ARM-software/mango/blob/master/benchmarking/Parameter_Spaces_Evaluated.ipynb

Mango Use case: Training NN Models for Resource Constrained IoT Devices

- STM32F446RE: SRAM 128 kB, Flash 512 kB
- STM32L476RG: SRAM 128 kB, Flash 1 MB
- STM32F746ZG: SRAM 320 kB, Flash 1 MB

$$f_{\text{opt}} = \lambda_1 f_{\text{error}}(\Omega) + \lambda_2 f_{\text{flash}}(\Omega) + \lambda_3 f_{\text{SRAM}}(\Omega) \quad (1)$$

where

$$f_{\text{error}}(\Omega) = \mathcal{L}_{\text{test}}(\Omega) \quad (2)$$

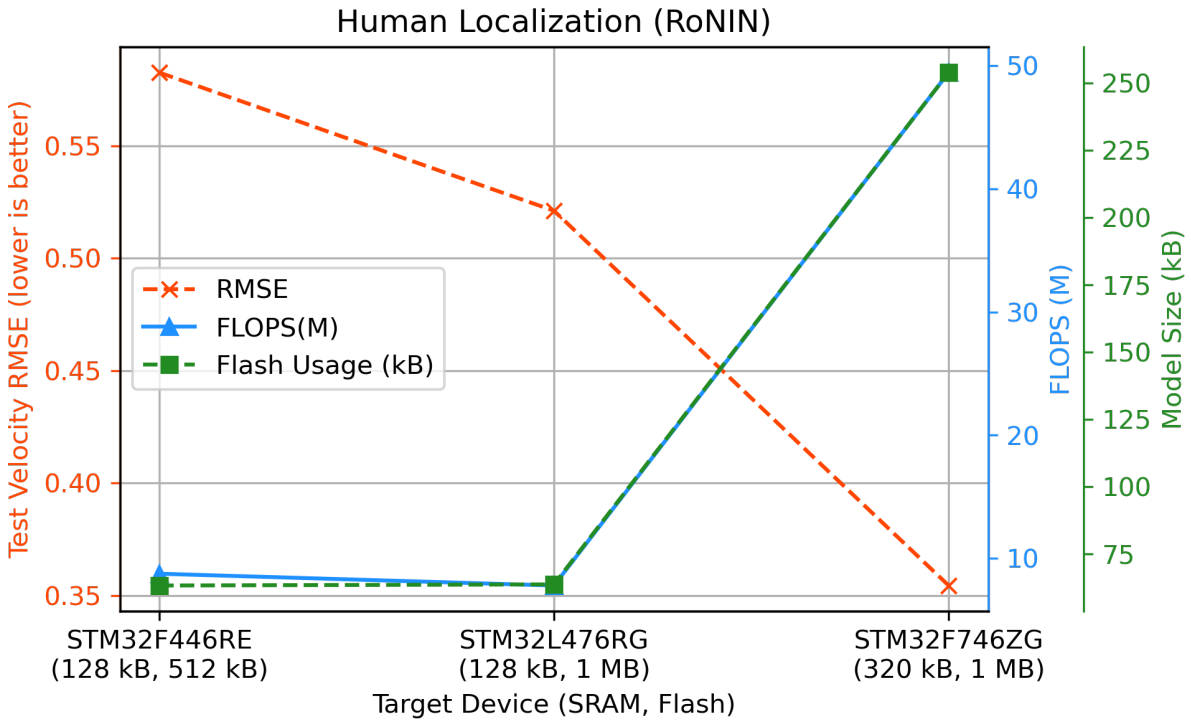
$$f_{\text{flash}}(\Omega) = \begin{cases} -\frac{flash_{proxy}}{\text{flash}_{\max}} \vee -\frac{\text{HIL information}}{flash_{\max}} \\ Penalty, f_{flash}(\Omega) > flash_{\max} \end{cases} \quad (3)$$

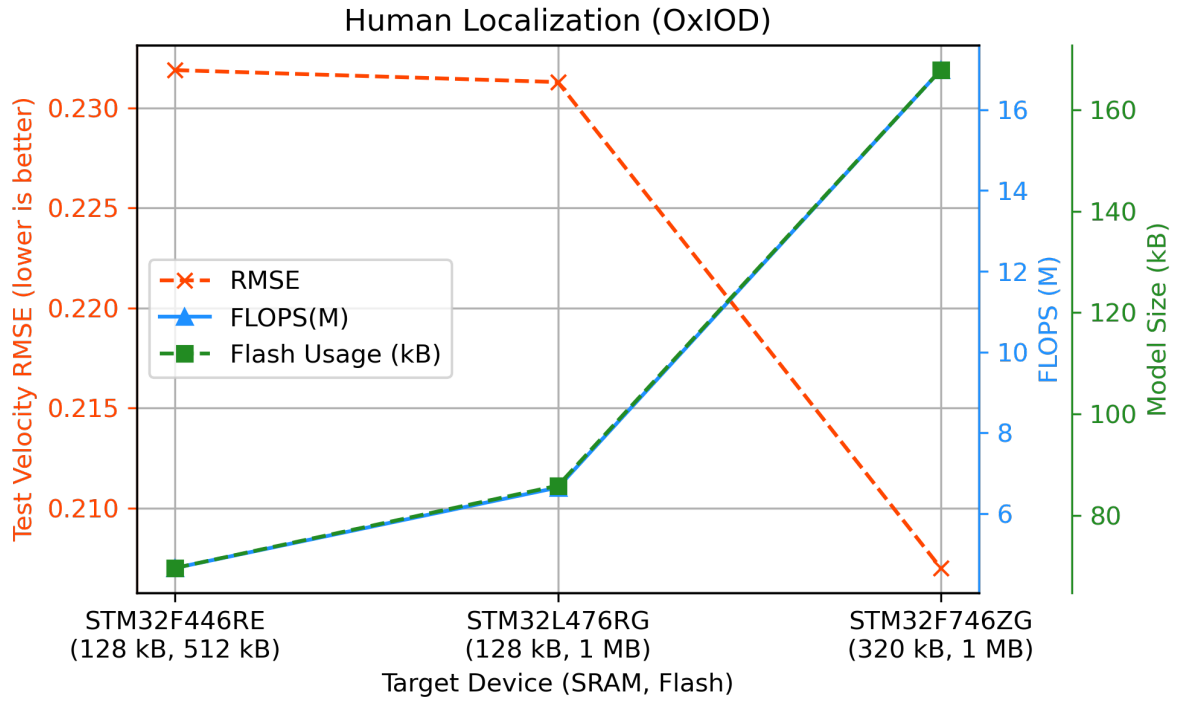
$$f_{SRAM}(\Omega) = \begin{cases} -\frac{SRAM_{Proxy}}{\text{SRAM}_{\max}} \vee -\frac{\text{HIL information}}{SRAM_{\max}} \\ Penalty, f_{SRAM}(\Omega) > SRAM_{\max} \end{cases} \quad (4)$$

SRAM Proxy [1], Flash Proxy [2]

[1] Fedorov, Igor, et al. "SpArSe: sparse architecture search for CNNs on resource-constrained microcontrollers." *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 2019.
[2] David, Robert, et al. "Tensorflow lite micro: Embedded machine learning on tinyml systems." *arXiv preprint arXiv:2010.08678* (2020).

Mango Use case: Training NN Models for Resource Constrained IoT Devices





Ronin [1], OxIOD [2]

[1] Herath, Sachini, Hang Yan, and Yasutaka Furukawa. "RoNIN: Robust Neural Inertial Navigation in the Wild: Benchmark, Evaluations, & New Methods." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.

[2] Chen, Changhao, et al. "Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference." IEEE Internet of Things Journal 7.5 (2020)

Runtime of Mango

Wall clock time (sec) taken by optimizers to sample next evaluation in sequential, parallel, and CASH settings

Optimizer (Surrogate)	Sequential	Parallel	CASH
Hyperopt (TPE)	0.001 ± 0.005	na	0.02 ± 0.001
Optuna (TPE)	0.07 ± 0.035	0.02 ± 0.006	0.02 ± 0.001
Mango (GP)	0.16 ± 0.008	0.12 ± 0.021	0.11 ± 0.002
GPyOpt (GP)	0.37 ± 0.051	1.76 ± 0.223	na
SMAC (Random forest)	0.70 ± 0.046	na	0.94 ± 0.037

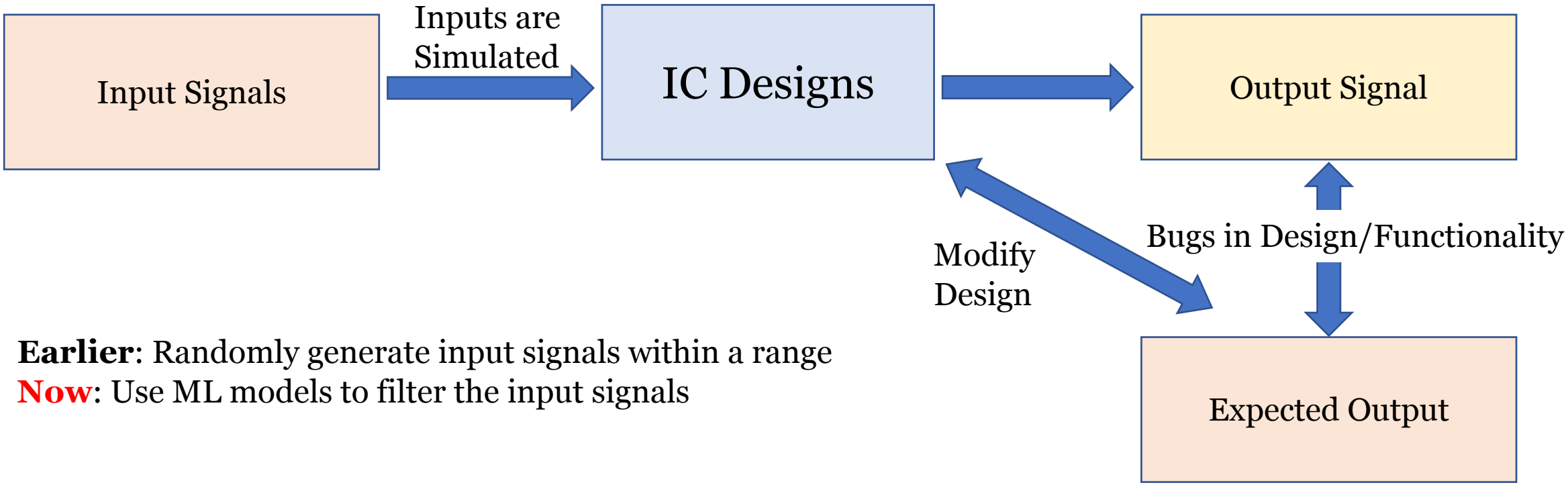
Summary

- Mango is **available opensource** to learn and contribute back to the community
- Mango **provides state-of-the-art** optimization capabilities
- Mango is **continuously test and improved** using production workloads
- We are working on **extending optimization capabilities** by including alternative optimization algorithms

https://github.com/ARM-software/mango

Backup

Example: Bug Hunting in Arm ICs



Earlier: Randomly generate input signals within a range
Now: Use ML models to filter the input signals

Design verification takes ~50% of total compute budget during development

Search Space

```
from scipy.stats import uniform
from mango.domain.distribution import loguniform

param_nn = {'type':'clf_nn',
            'num_of_nodes':range(10, 101)}

param_dtree = {'type':'clf_dtree',
               'max_features':['auto', 'sqrt', 'log2'],
               'max_depth':range(1,21),
               'splitter':['best','random'],
               'criterion':['gini','entropy']}

param_svm = {'type':'clf_svm',
             'gamma':uniform(0.1, 4),
             'C':loguniform(-7, 10)}

param_xgboost = {'type':'clf_xgboost',
                 'learning_rate':uniform(0, 1),
                 'gamma':uniform(0, 5),
                 'max_depth':range(1,21),
                 'n_estimators':range(1,11),
                 'booster':['gbtree','gblinear','dart']}

param_knn = {'type':'clf_knn',
             'n_neighbors': range(1, 51),
             'algorithm':['auto','ball_tree','kd_tree',
             'brute']}
```

Search Space: MetaTuner

```
from scipy.stats import uniform
from mango.domain.distribution import loguniform

param_svm = {'gamma':uniform(0.1, 4),
             'C':loguniform(-7, 10)}

param_xgboost = {'learning_rate':uniform(0, 1),
                 'gamma':uniform(0, 5),
                 'max_depth':range(1,11),
                 'n_estimators':range(1,301),
                 'booster':['gbtree','gblinear','dart']}

param_knn = {'n_neighbors':range(1, 51),
             'algorithm':['auto','ball_tree','kd_tree',
             'brute']}
```