Automating Hyperparameter Tuning

A framework overview and a practical introduction to the Optuna framework for future data scientists

Ali HOUSSENALY
Deep Learning Project
SDD 2025 - 2026

OPTUNA

Table of content

Overview of Hyperparameter Tuning Frameworks

- Introduction & Motivation
- Overview of Classical Frameworks and their Characteristics
- What Makes Optuna State-of-the-Art

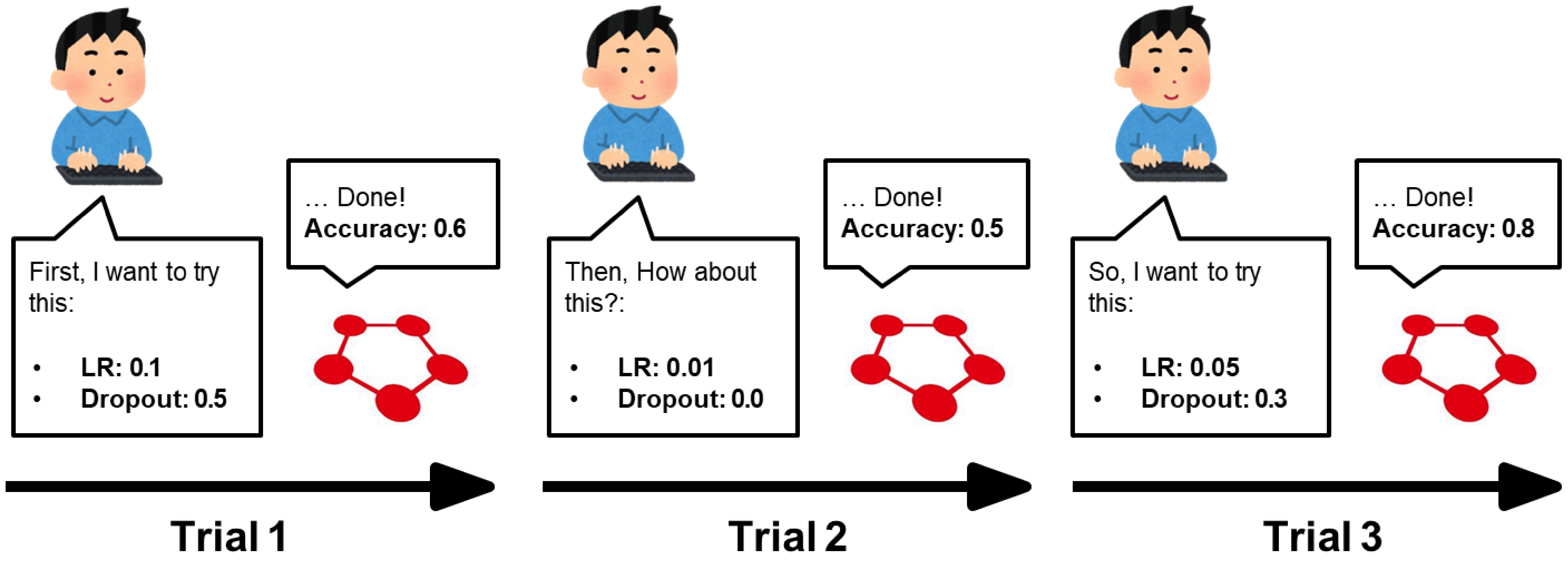
Deep-Dive into the Optuna Framework

- Code Example
- How Optuna Works
- Additional Functionalities :
    - Parallel Processing
    - Visualizations and Dashboard

Overview of Hyperparameter Tuning Frameworks

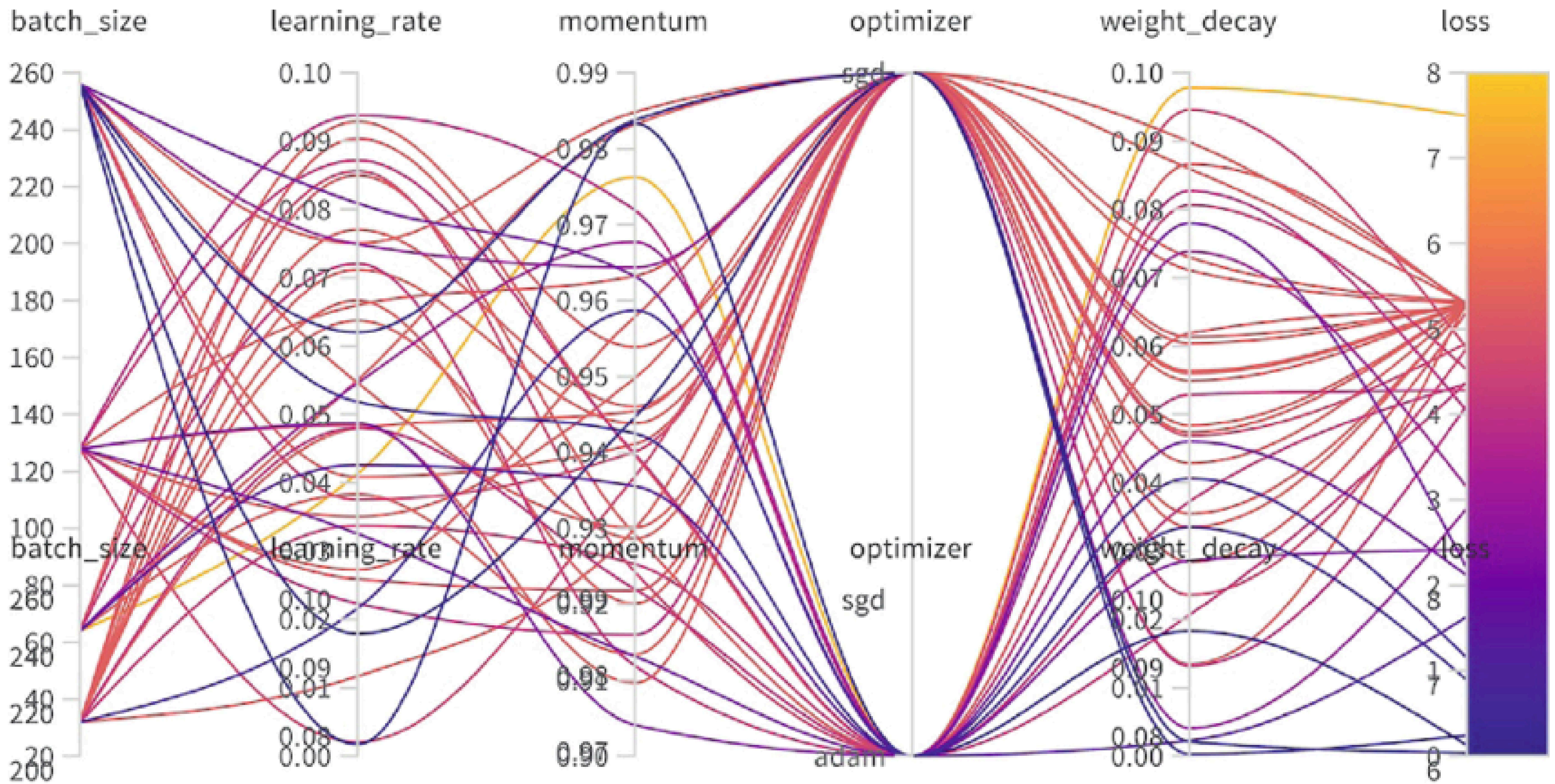
Introduction & Motivation

A classical and manual approach to hyperparameter tuning is very time consuming and repetitive



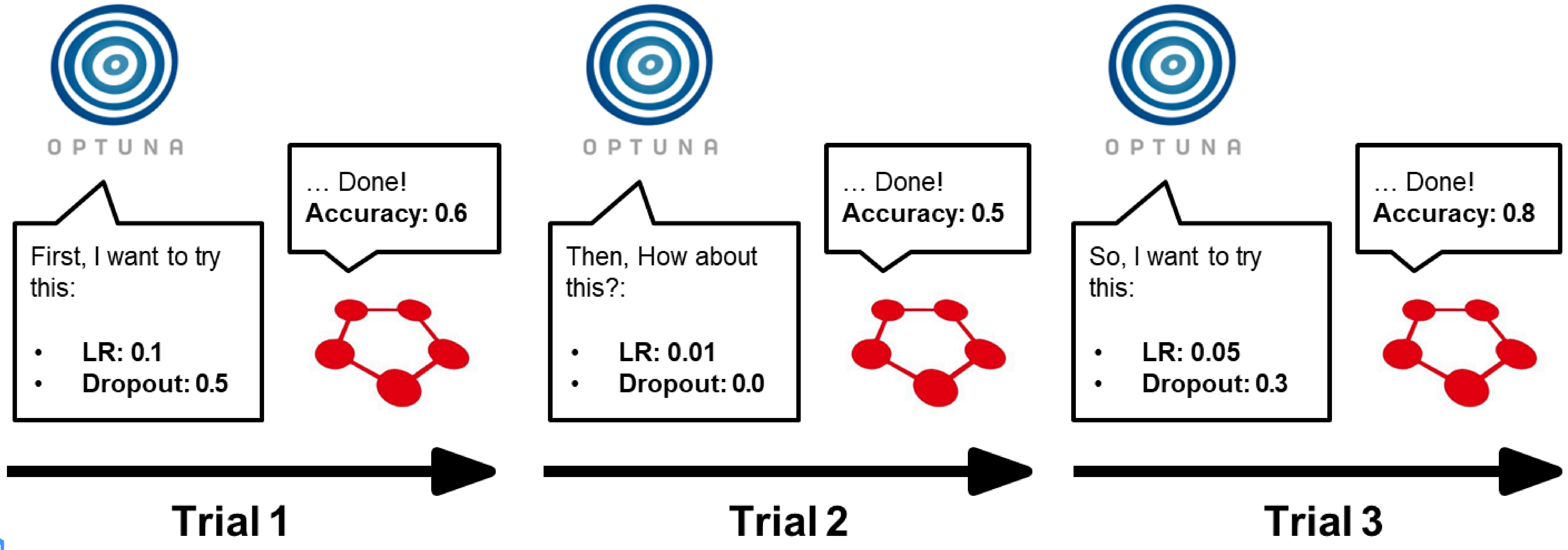
Introduction & Motivation

As models grow more complex, their performance increasingly depends on well-tuned hyperparameters



Introduction & Motivation

Efficiently automating this process has become increasingly critical



Overview of Classical Frameworks and their Characteristics

6 main characteristics define hyperparameter tuning frameworks

1
API Style

- **Define-and-run:** Static search space defined before the optimization
- **Define-by-run (Optuna):** Dynamically built search space during execution, allowing more flexible workflows

2
Pruning

- **Early stopping:** Terminates weak trials based on intermediate results
- **Efficiency gain:** Saves compute by avoiding full training of bad configuration

3
Lightweight

- **Minimal overhead:** Few dependencies, easy installation
- **Simple integration:** Fits naturally into existing Python workflow

4
Distributed

- **Scalable execution:** Runs trials across multiple machines or workers
- **Faster search:** Parallel exploration of the hyperparameter space

5
Visualiztions

- **Built-in visualizations:** Optimization curves, parameter importance, trial history
- **Interactive monitoring:** Inspect and compare trials in real time

6
Open-Source

- **Transparent:** Code is public, inspectable, and modifiable
- **Community-driven:** Maintained and improved by an active open-source ecosystem

What Makes Optuna State-of-the-Art

Among open-source hyperparameter optimization frameworks, Optuna is widely regarded as state-of-the-art

Framework	API Style	Pruning	Lightweight	Distributed	Dashboard	OSS
SMAC [3]	define-and-run	✗	✓	✗	✗	✓
GPyOpt	define-and-run	✗	✓	✗	✗	✓
Spearmint [2]	define-and-run	✗	✓	✓	✗	✓
Hyperopt [1]	define-and-run	✗	✓	✓	✗	✓
Autotune [4]	define-and-run	✓	✗	✓	✓	✗
Vizier [5]	define-and-run	✓	✗	✓	✓	✗
Katib	define-and-run	✓	✗	✓	✓	✓
Tune [7]	define-and-run	✓	✗	✓	✓	✓
Optuna (this work)	define-by-run	✓	✓	✓	✓	✓

Deep-Dive into the Optuna Framework

Code Example

This example showcases the define-by-run API-style (1) and lightweight (3) use of Optuna

```python
def objective(trial):
    iris = sklearn.datasets.load_iris()
    x, y = iris.data, iris.target

    classifier_name = trial.suggest_categorical("classifier", ["SVC", "RandomForest"])
    if classifier_name == "SVC":
        svc_c = trial.suggest_float("svc_c", 1e-10, 1e10, log=True)
        classifier_obj = sklearn.svm.SVC(C=svc_c, gamma="auto")
    else:
        rf_max_depth = trial.suggest_int("rf_max_depth", 2, 32, log=True)
        classifier_obj = sklearn.ensemble.RandomForestClassifier(
            max_depth=rf_max_depth, n_estimators=10
        )

    score = sklearn.model_selection.cross_val_score(classifier_obj, x, y, n_jobs=-1, cv=3)
    accuracy = score.mean()
    return accuracy


if __name__ == "__main__":
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=100)
    print(study.best_trial)
```

How Optuna works

Optuna combines sampling and pruning to guide the search intelligently

Search Space

Sampling Strategy (Samplers)

Samplers are algorithms that proposes the next hyperparameters to evaluate

Available sampler choices in Optuna :

- **Grid Search:** Exhaustively evaluates all combinations in a predefined discrete search space.
- **Random Search:** Samples hyperparameters uniformly at random.
- ***Tree-structured Parzen Estimator (TPE): Bayesian optimization method; default sampler in Optuna.***
- **CMA-ES:** Evolutionary strategy well-suited for continuous optimization.
- GPyTorchSampler ; PartialFixedSampler ; NSGA-II ; QMC and others ...

Pruning Strategy (Pruners)

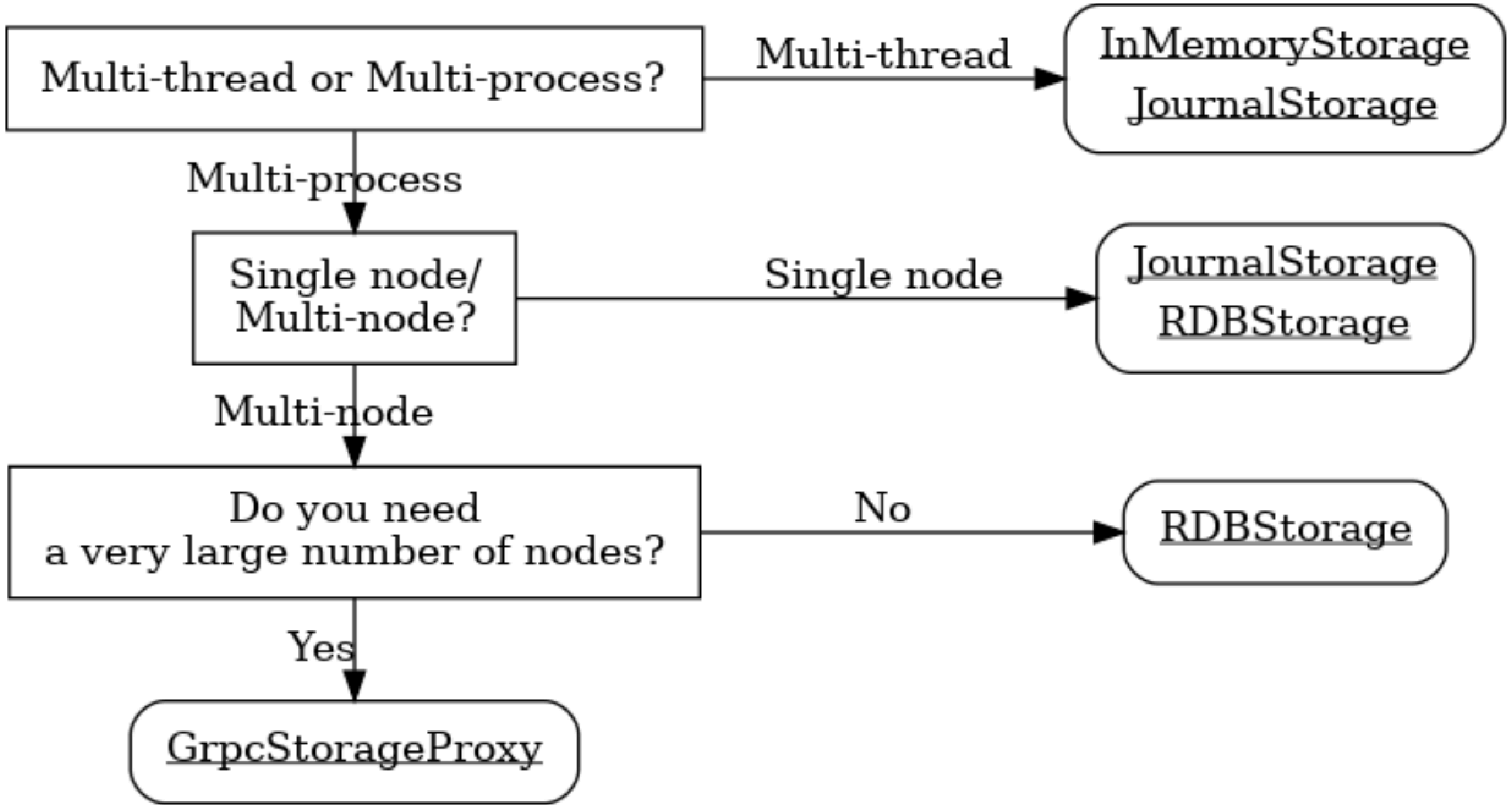
Pruners are algorithms that detects unpromising trials based on intermediate results

Available pruner choices in Optuna:

- **Median Pruner:** Stops a trial if its performance is below the median of completed trials.
- **Successive Halving Pruner:** Allocates resources progressively; eliminates poor performers early.
- **Threshold Pruner:** Stops trials that fail to reach a user-defined performance threshold.
- **Hyperband ; Percentile ; Patient Pruner and others ...**

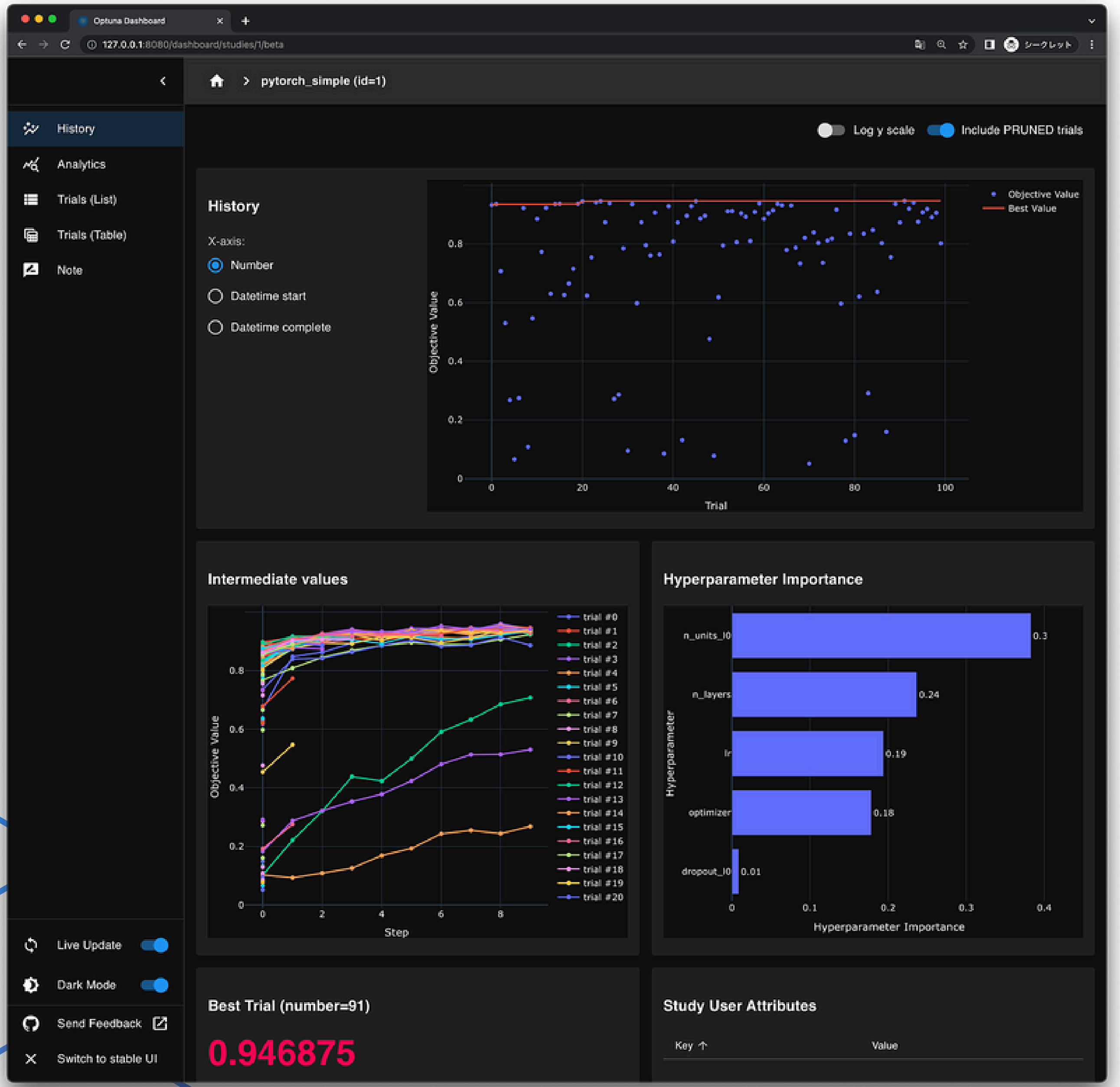
Additional Functionalities : Parallel Processing

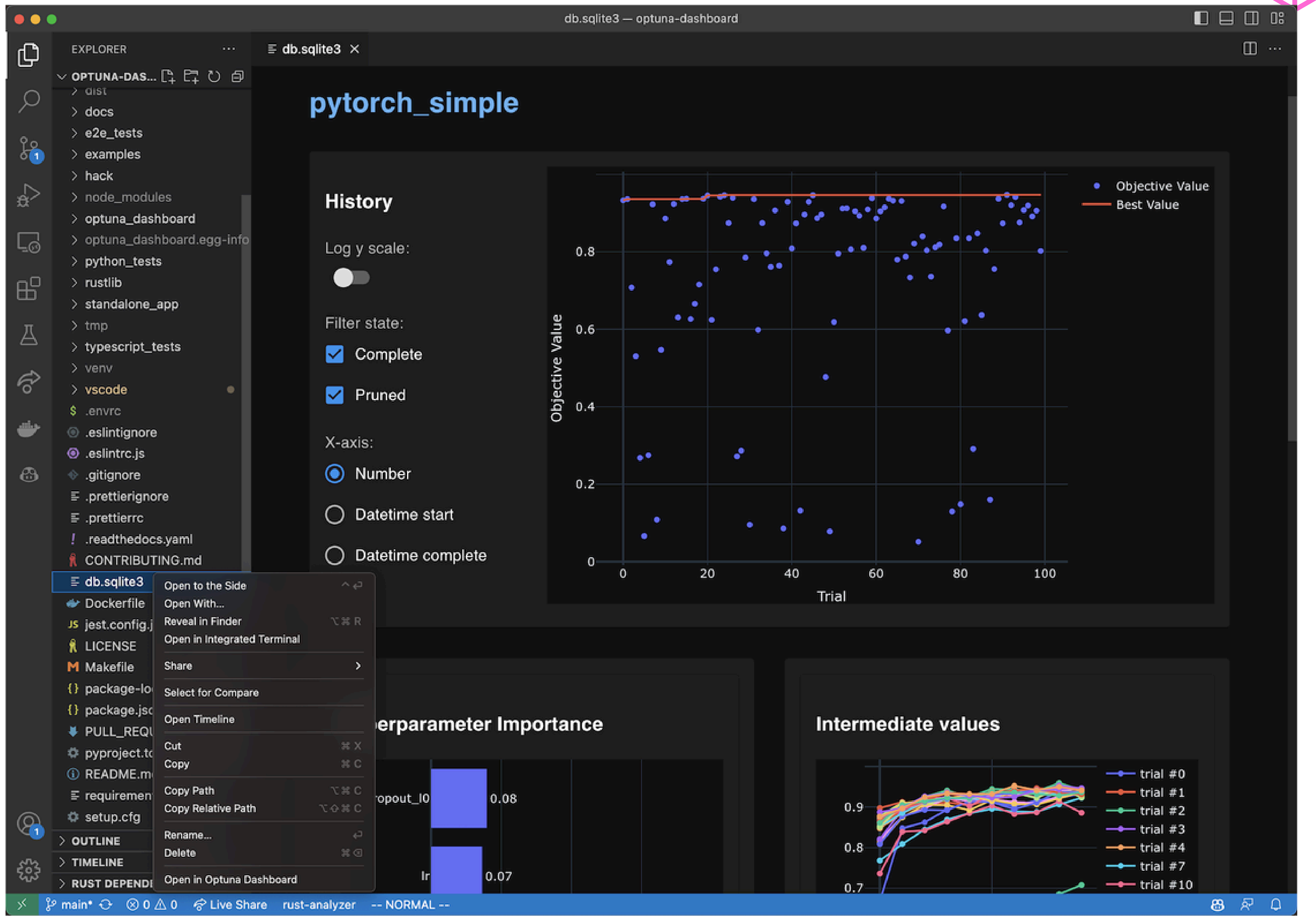
Optuna supports multi-thread, multi-process and multi-node optimization



Additional Functionalities : Visualizations and Dashboard

Optuna offers a comprehensive visualization toolbox with online and local dashboard software





Conclusion & Transition to Demo

Key Takeaways

A framework we can already put to use in our next hackathon

- Optuna provides an **easy-to-use**, **flexible**, and highly **efficient** approach to hyperparameter optimization
- Its **sampler–pruner** architecture and **define-by-run API** clearly set it apart from other frameworks
- It performs strongly across all **six evaluation criteria** and is widely regarded as the **state-of-the-art** open-source tuning framework.
- It’s the **perfect tool to leverage in our upcoming hackathon 😉 !**