

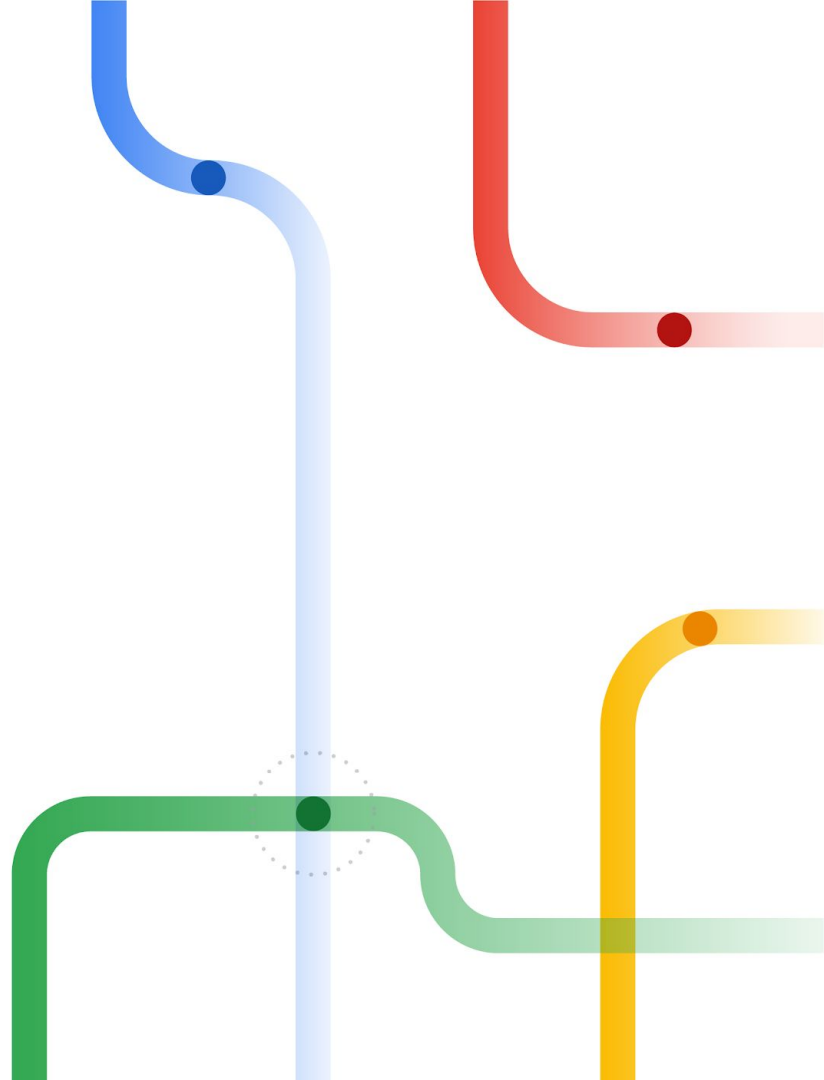
# Open Source Vizier Tutorial

## AutoML-Conf 2023

Xingyou (Richard) Song  
[xingyousong@google.com](mailto:xingyousong@google.com)

Qiuyi (Richard) Zhang  
[qiuyiz@google.com](mailto:qiuyiz@google.com)

Sagi Perel  
[sagipe@google.com](mailto:sagipe@google.com)



# Tutorial: Follow Along

**Website:** <https://sites.google.com/view/oss-vizier-automl-2023/>

**Colab:**

[https://colab.research.google.com/github/google/vizier/blob/main/docs/tutorials/automl\\_conf\\_2023.ipynb](https://colab.research.google.com/github/google/vizier/blob/main/docs/tutorials/automl_conf_2023.ipynb)

# Additional Links

**Code:** <https://github.com/google/vizier>

**Documentation:** <https://oss-vizier.readthedocs.io/en/latest/index.html>

**AI Blog:**

<https://ai.googleblog.com/2023/02/open-source-vizier-towards-reliable-and.html>

**Paper:** <https://arxiv.org/abs/2207.13676>

**OpenReview:** <https://openreview.net/forum?id=SfIRITSUxc>

# Table of Contents + Approximate Schedule

1. Overview (10 minutes)
2. Hands-On 1: Basics (20 minutes)
3. (Optional) Deep Dive: What's under the hood? (10 minutes)
4. Hands-On 2: Algorithm API (15 minutes)
5. Hands-On 3: Benchmarks API (25 minutes)
6. Questions (10 minutes)
7. (Optional) Extras

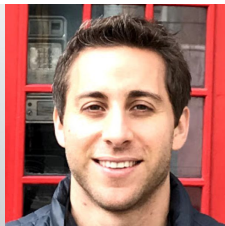
# Overview

# Vizier Team

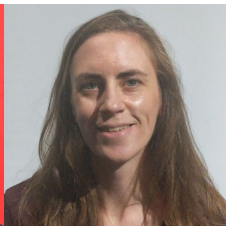
**Setareh Ariafar**



**Lior Belenki**



**Emily Fertig**



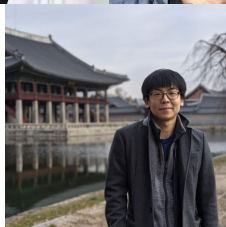
**Daniel Golovin**



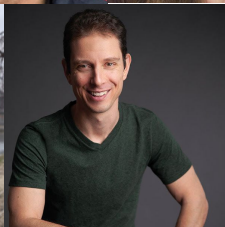
**Tzu-Kuo Huang**



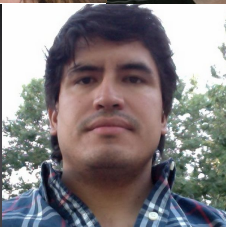
**Greg Kochanski**



**Chansoo Lee**



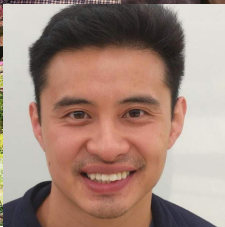
**Sagi Perel**



**Adrian Reyes**



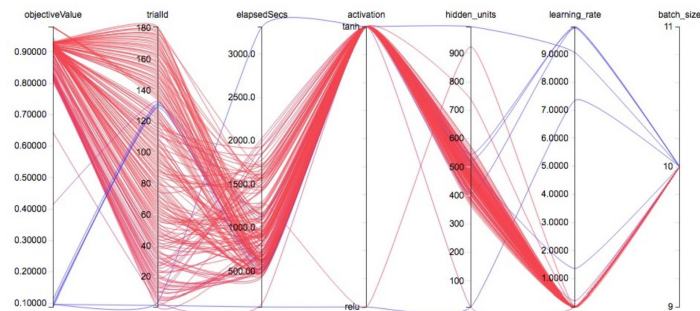
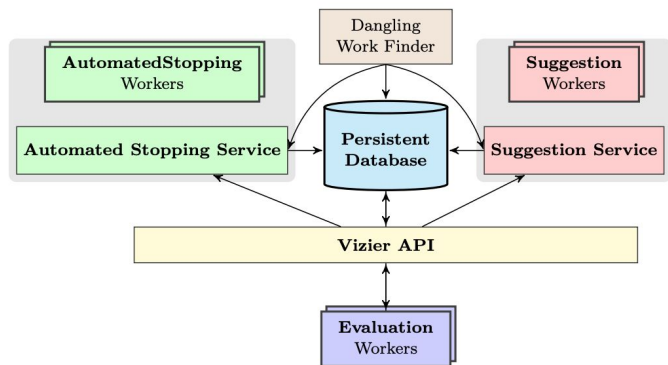
**Xingyou Song**



**Richard Zhang**

# Overview: Google Vizier (2017)

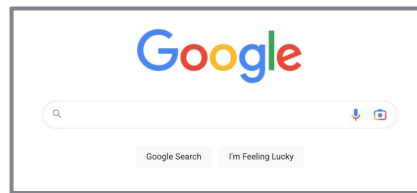
- Tunes many of Google's research + products
- Thousands of monthly users
- Tuned millions of objectives



# Notable Users / Downstream Wins

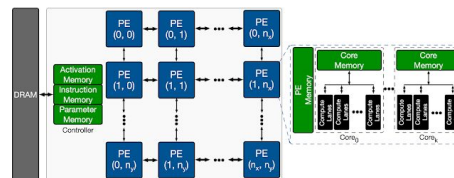
## Production

- [Search](#), [Ads](#), [Youtube](#)



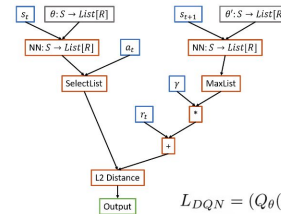
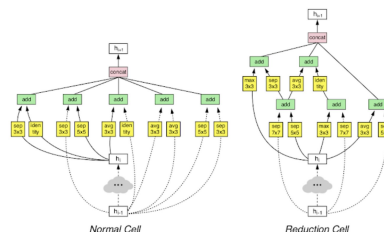
## Tuning Research Results:

- [Hardware Design](#), [Robotics](#)
- [Protein Design](#)



## Backend for Evolution:

- [Neural Architecture Search](#)
- [Symbolic Algorithm Search](#)



Node Types

Inputs

Operators

Parameters

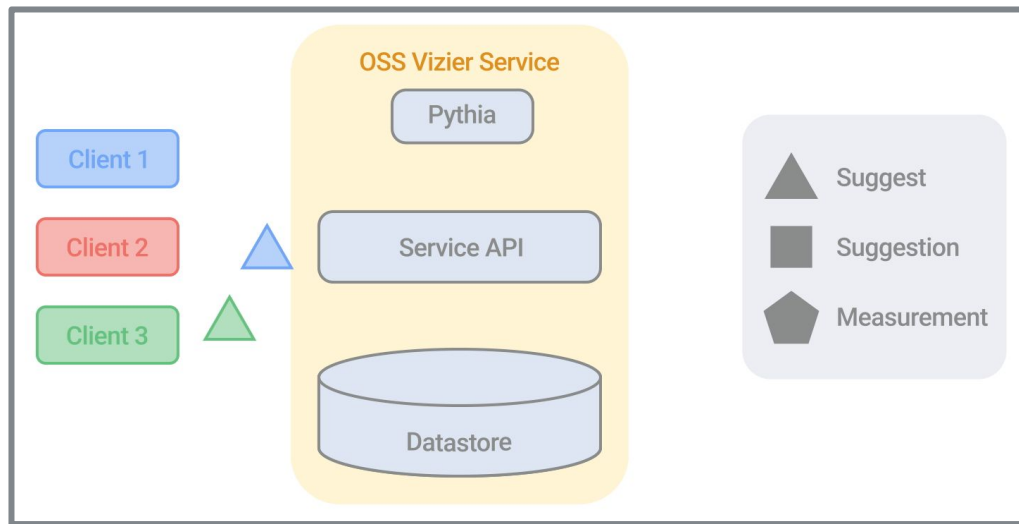
Output

$$L_{DQN} = (Q_{\theta}(s_t, a_t) - (r_t + \gamma \max_a Q_{\theta'}(s_{t+1}, a)))^2$$

Google Research

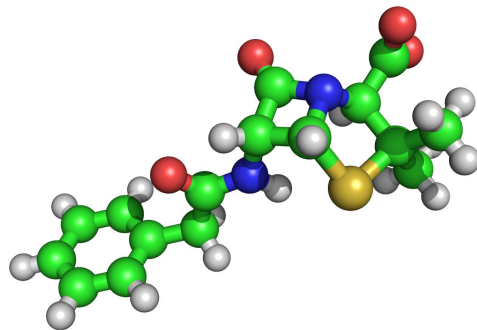
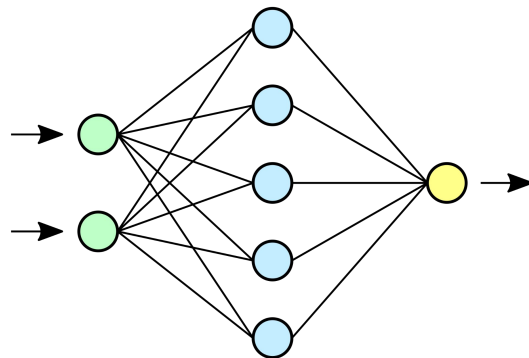


# Why a Service?



# The Wide Variety of Scenarios

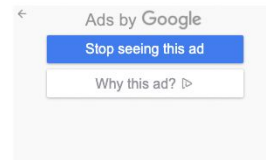
- Tuning large ML model hyperparameters
- [Chemical/Biological processes](#)
- [Optimizing cookie recipes](#)



Very different workflows!

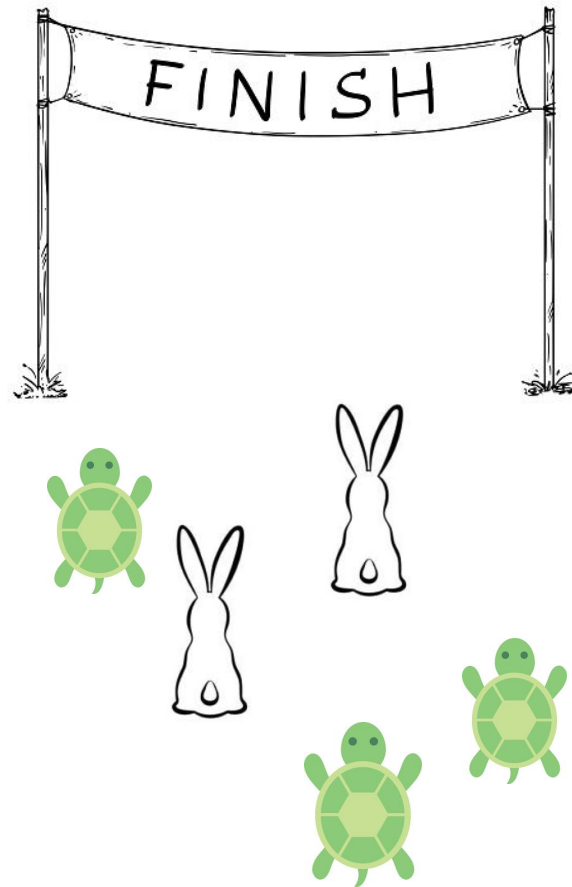
## Google's new AI learns by baking tasty machine learning cookies

The system "designs excellent cookies", according to its creators



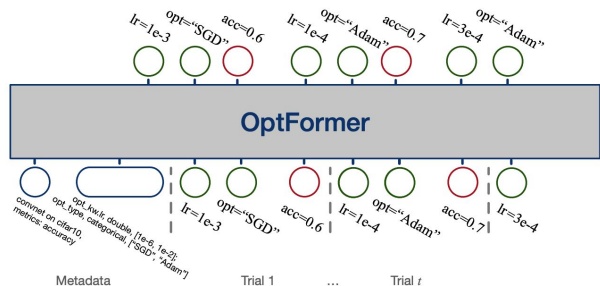
# Workflow Possibilities

- Eval Latency: Seconds to Weeks
- Eval Budget:  $10^1$  to  $10^7$  Trials
- Asynchronous or Synchronous (Batched)
- Failed evaluations: Retried or abandoned
- Early Stopping



# Benefits of a Service: No Evaluation Assumptions!

- Users have freedom of when to:
  - Request trials
  - Evaluate Trials
  - Report results
- Service can preserve data on prior usage
  - Led to [OptFormer paper](#)!












# OSS Vizier: 2022

- Standalone + customizable Python codebase
- User can host service

**Open Source Vizier: Reliable and Flexible Black-Box Optimization.**



Build status:  pypi package **0.1.1**  pytest\_core **passing**  pytest\_clients **passing**  pytest\_algorithms **passing**  pytest\_benchmarks **passing**

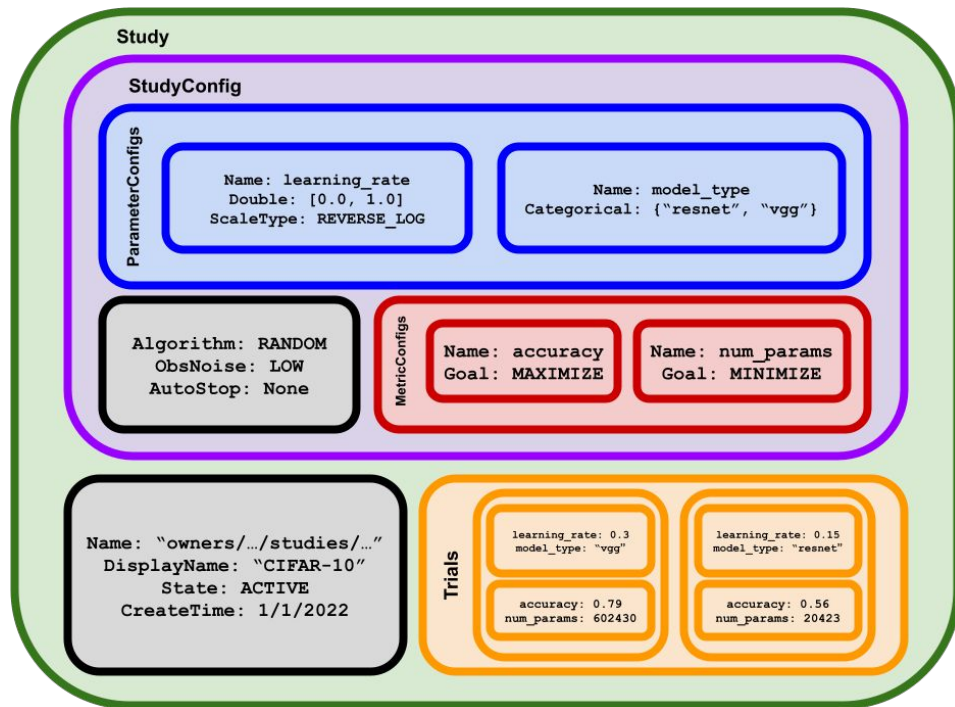
 docs **passing**

[Google AI Blog](#) | [Getting Started](#) | [Documentation](#) | [Installation](#) | [Citing Vizier](#)

# Hands-On 1: Basics (Colab, ReadTheDocs)

# Definitions

- **Study:**
  - Entire Optimization Run
- **StudyConfig: Configuration**
  - Search Space
  - Algorithm
  - Noise
  - ...
- **ParameterConfig: Parameter Specification**
- **MetricConfig: Metric Specification**



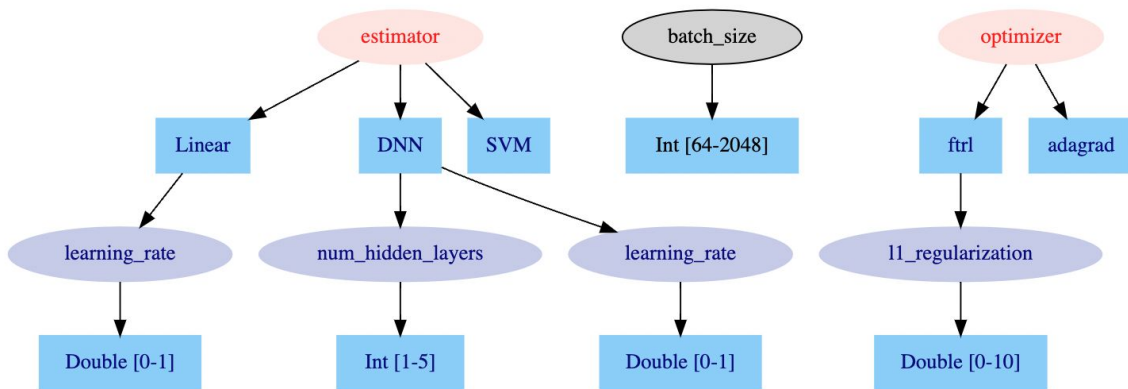
# Search Space Construction: ParameterConfigs

Core:

- Double: Continuous range [a,b]
- Integer: Integer range [a,b]
- Discrete: Finite set of floats.
- Categorical: Finite set of strings.

Each ParameterConfig also contains:

- Scaling Type (uniform, log)
- Child/Conditional Parameters

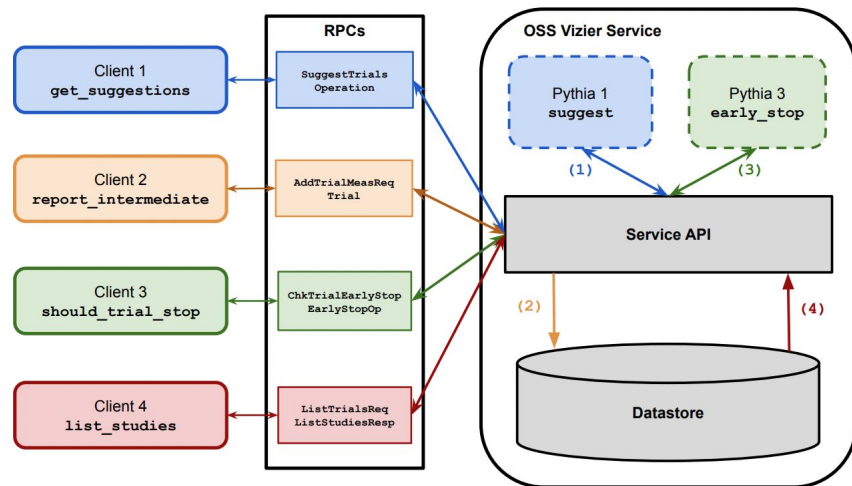




**(Optional) Deep Dive: What's under the hood?**

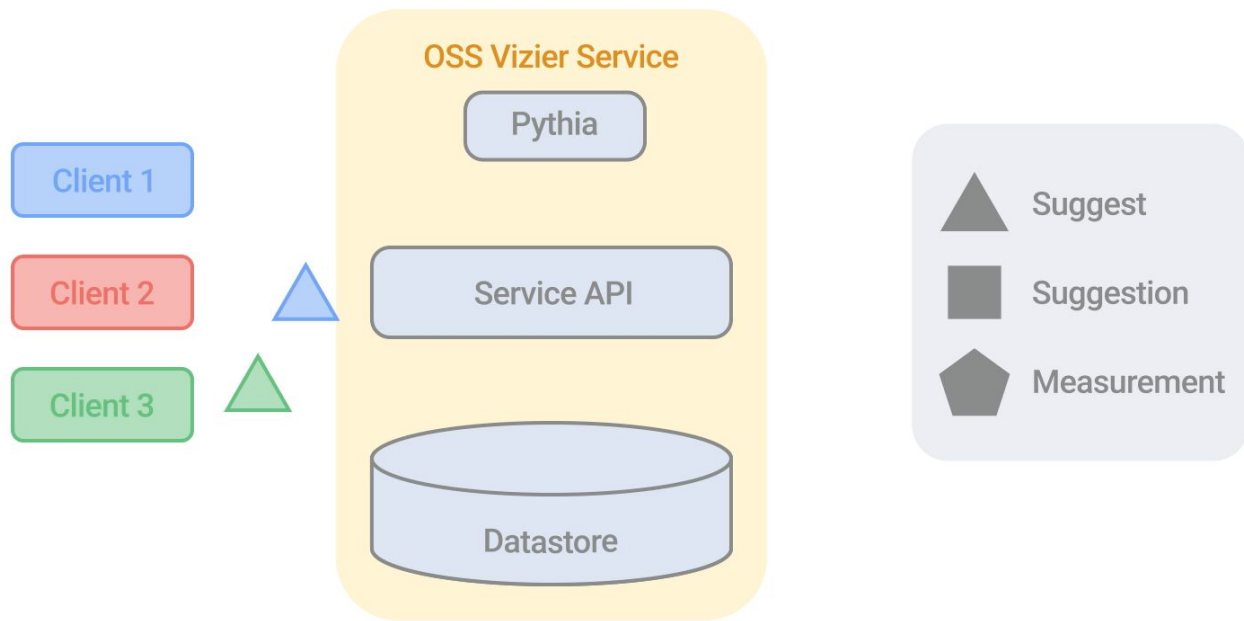
# Core Server-Client Procedure

- Client sends `SuggestTrials` RPC to Server.
- Server starts Pythia policy
  - `Operation` protobuf to keep track of everything
- Client repeatedly pings server on status of `Operation`
- Client finally receives suggestion



**All transactions + operations are stored in server datastore!**

# Suggestion Animation



# Distributed Communication

- Remote Procedure Calls (RPCs) formatted as Protocol Buffers (protobufs)
- Server + Client classes based on gRPC

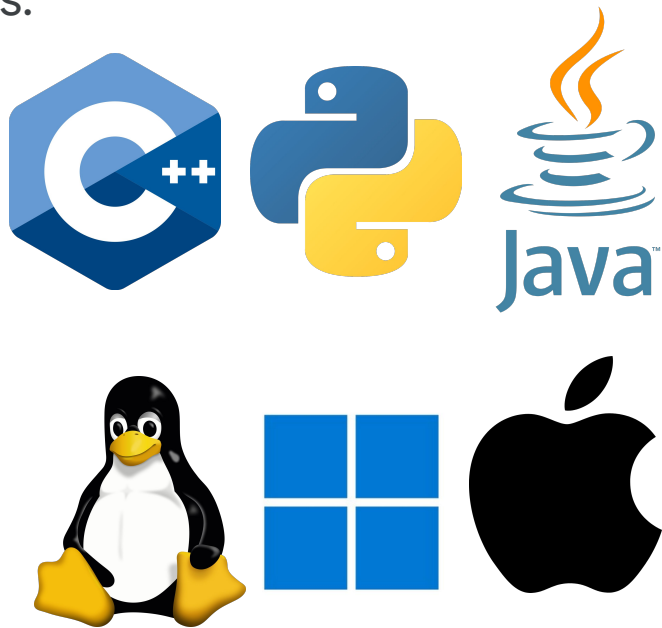


A high performance, open source universal RPC framework

# Language + Platform Independence

Protobufs (for RPC Backend) are ubiquitous across:

- Languages
  - C++, Python, Java, and many more
- Platforms
  - Linux, Windows, Mac



# PyVizier: Abstracting away Protobufs

- Hides away RPC protobufs from user + algorithms
- Use same Python libraries across all Vizier variants
- More Pythonic data structures

```
1 from vizier.service import study_pb2
2 from google.protobuf import struct_pb2
3
4 param_1 = study_pb2.Trial.Parameter(parameter_id='learning_rate', value=struct_pb2.
5     Value(number_value=0.4))
6 param_2 = study_pb2.Trial.Parameter(parameter_id='model_type', value=struct_pb2.
7     Value(string_value='vgg'))
8 metric_1 = study_pb2.Measurement.Metric(metric_id='accuracy', value=0.4)
9 metric_2 = study_pb2.Measurement.Metric(metric_id='num_params', value=20423)
10 final_measurement = study_pb2.Trial.Measurement(metrics=[metric_1, metric_2])
11 trial = study_pb2.Trial(parameters=[param_1, param_2], final_measurement=
12     final_measurement)
```

**Original Protobuf: Verbose + Complex**

```
1 from vizier.pyvizier import ParameterDict, ParameterValue, Measurement, Metric,
2     Trial
3
4 params=ParameterDict()
5 params['learning_rate'] = ParameterValue(0.4)
6 params['model_type'] = ParameterValue('vgg')
7 final_measurement = Measurement()
8 final_measurement.metrics['accuracy'] = Metric(0.7)
9 final_measurement.metrics['num_params'] = Metric(20423)
10 trial = pv.Trial(parameters=params, final_measurement=final_measurement)
```

**PyVizier: More Pythonic!**

# Hands-On 2: Algorithm API

# Typical Algorithm Design: “Designer”

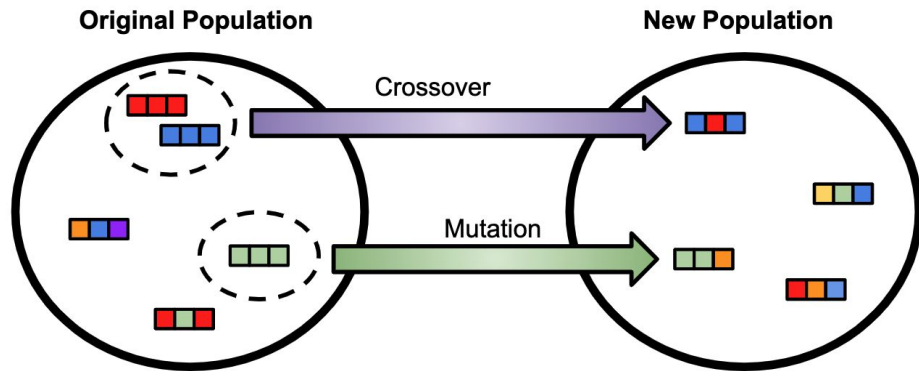
- Very typical API for writing an algorithm:

```
class Designer(...):  
    """Suggestion algorithm for sequential usage."""  
  
    @abc.abstractmethod  
    def update(self, completed: CompletedTrials, all_active: ActiveTrials) -> None:  
        """Updates recently completed and ALL active trials into the designer's state."""  
  
    @abc.abstractmethod  
    def suggest(self, count: Optional[int] = None) -> Sequence[vz.TrialSuggestion]:  
        """Make new suggestions."""
```



# Service Requirements

- Ensure fault-tolerance on algorithms:
  - Fresh algorithm can recover when needed
  - Use historical trials as “algorithm state”!
- Querying the history:
  - Algorithm can query whichever trials they need.
  - Very useful for algorithms which work in batches/populations
    - e.g. Genetic Algorithms



# Hosted Algorithm: “Policy”

- **PolicySupporter**: Query previous trials to recover state.
- **stateless\_algorithm**: Stateless algorithm or Designer

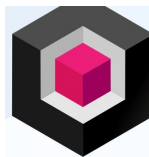
```
class TypicalPolicy(Policy):  
  
    def __init__(self, policy_supporter: PolicySupporter):  
        self._policy_supporter = policy_supporter  
  
    def suggest(self, request: SuggestRequest) -> SuggestDecision:  
        all_completed = policy_supporter.GetTrials(status_matches=COMPLETED)  
        all_active = policy_supporter.GetTrials(status_matches=ACTIVE)  
        suggestions = stateless_algorithm(all_completed, all_active)  
        return SuggestDecision(suggestions)
```

# Algorithms Included ([Reference](#))

- **Classic:** Random, Grid, Shuffled-Grid, Quasi-Random
- **Evolution:** CMA-ES, NSGA2, EagleStrategy
- **Boolean:** BOCS, Harmonica
- **Bayesian:** [GP-Bandit](#)

# Hands-On 3: Benchmarks API

# Benchmarking Goal



black  
box

OBJECTIVE

Find the best black-box optimizer for machine learning.

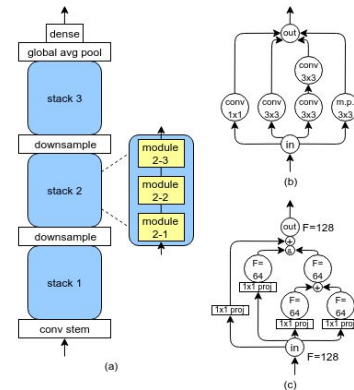
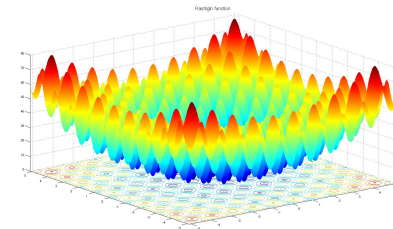
Many papers and competitions (like BBO Challenge at [NeurIPS](#)) require good benchmarking to **discover and research SOTA algorithms**:

- Find generally useful and robust set of diverse metric/benchmarks
- Allow for easy comparison with other competitive algorithms
- Reduce human interpretation biases from metric/benchmark design
- Remove confusion or possible misinterpretation from plots

# Benchmarks

## Included:

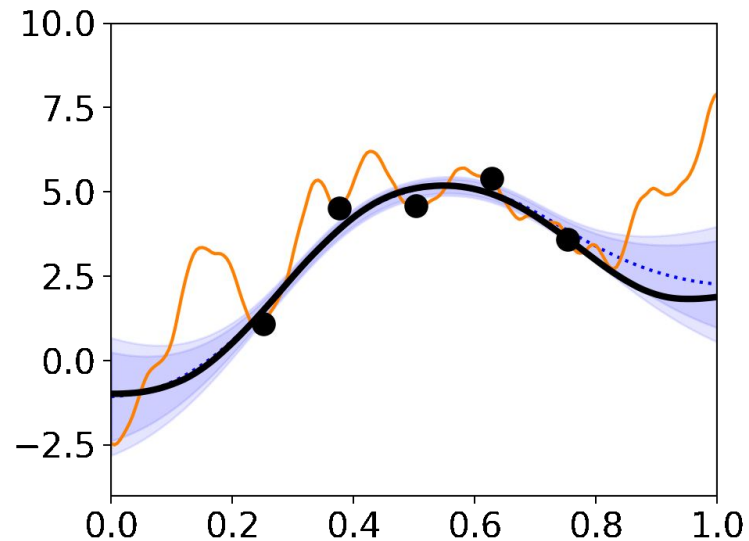
- [BBOB](#), [COMBO](#)
- NASBENCH ([101](#) + [201](#))
- [HPOB](#)
- [Atari100K](#)
- Utilities (Noise, Shifting, Sparsifying, etc.)



# Vizier Default: GP-Bandit

Inspired by original 2015 C++ implementation **(before AutoDiff)**

- Gaussian Process Kernel
  - Matern-5/2
  - Optional Linear Kernel Addition
- UCB + Trust Region Acquisition
  - Evolutionary Optimizer [“Eagle Strategy”](#)
- Input and Objective Warping
  - Outlier removal + Gaussian-fitting + Log warping
- ARD + Priors Optimization
  - JAX-based LBFGS-B for constrained optimization
  - Priors are tuned and ensembled.



# Algorithmic Advantages

- AutoDiff + GPU support via JAX + Tensorflow Probability
  - Most other packages only use NumPy or Sklearn
  - Can easily do quick experimentation (priors, hparams, optimizers, acquisitions...)
- “Advanced” Tricks
  - Trust Region, Warping, ARD-optimization, Self-Tuning
- Many Features
  - Categorical/Discrete, Batched/Masked (UCB-PE), Multi-metric (HV Scalarization)





# Running Benchmarks

Benchmark protocols can be specified and then parallelized using Runners.

```
class BenchmarkState:  
    """State of a benchmark run. It is altered via benchmark protocols."""  
  
    experimenter: Experimenter  
    algorithm: PolicySuggester
```

**Maximal Flexibility:** Runner are mutators on **BenchmarkState**, which wraps the **PolicySupporter**, **Policy**, and **Experimenter**.

# Ray Integration

For easy algorithmic imports, Vizier supports integration into the [RayTune API](#):

- Search space and config conversions
- Vizier Searcher (to integrate with Ray)
- [Tutorial Colab](#) in RTD
- Parallelization support in Ray on Cloud

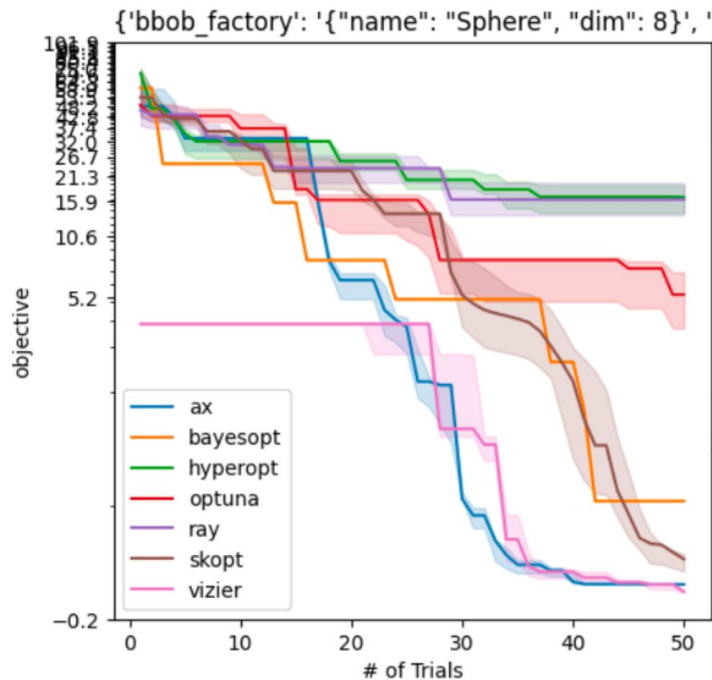


# Performance on BBOB

**Vizier (default) is competitive:**

from best objective plots across multiple benchmarks.

- How to quantify performance?
- How to aggregate across benchmarks?
- How to diversify across benchmarks and time horizon?



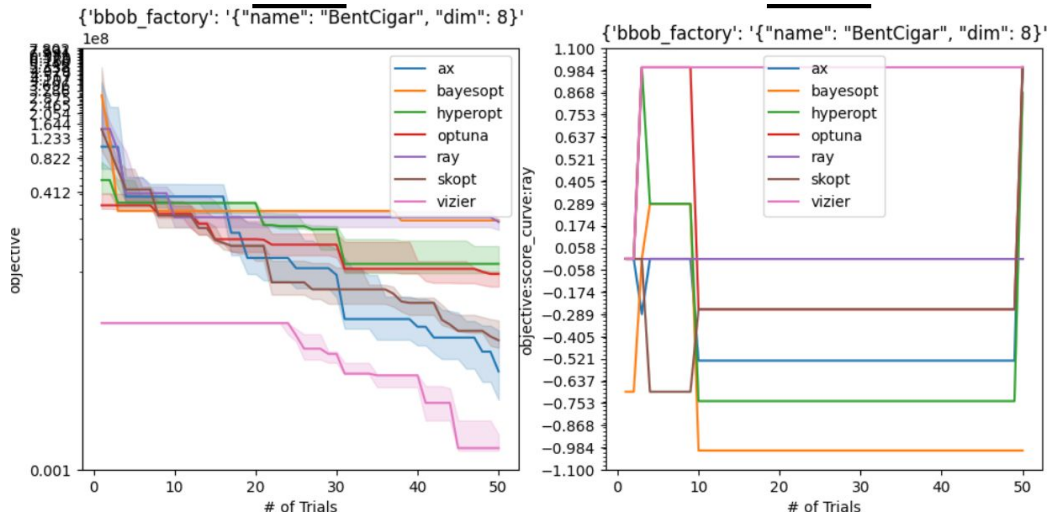
# Analysis Metrics

**Issue:** Performance can be subjective, especially across multiple benchmarks and plots.

**(Partial) Solution:** We support easy addition of normalized metrics, such as [performance profile](#), a gold standard in optimization.

## Example Metrics:

- LogEfficiency
- PercentageBetter
- Custom



# Try Out OSS Vizier!



For your BBO algorithmic, benchmarking, and analysis needs!

## Additional Links:

**Code:** <https://github.com/google/vizier>

**Documentation:** <https://oss-vizier.readthedocs.io/en/latest/index.html>

**AI Blog:**

<https://ai.googleblog.com/2023/02/open-source-vizier-towards-reliable-and.html>

**Paper:** <https://arxiv.org/abs/2207.13676>

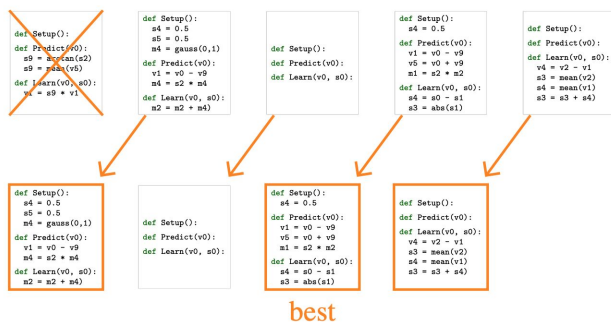
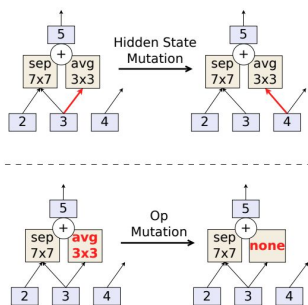
Google Research

**Thank you! Questions?**

# Extras

# PyGlove: Evolutionary + Combinatorial Computation

- OSS Vizier only supports flat search spaces + conditionals
  - Lacks choice function:  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- Integrate Vizier backend w/ PyGlove!
  - Vizier handles distributed system ([Tutorial](#))
  - Ex: Evolution for [NAS](#), [Genetic Programming](#)







# Vertex/Cloud Vizier



- Prod service for external users / businesses
- Shared client API: Easily switch b/w OSS or Cloud

Vertex AI > Documentation > Guides

Was this helpful?  

## Vertex AI Vizier overview

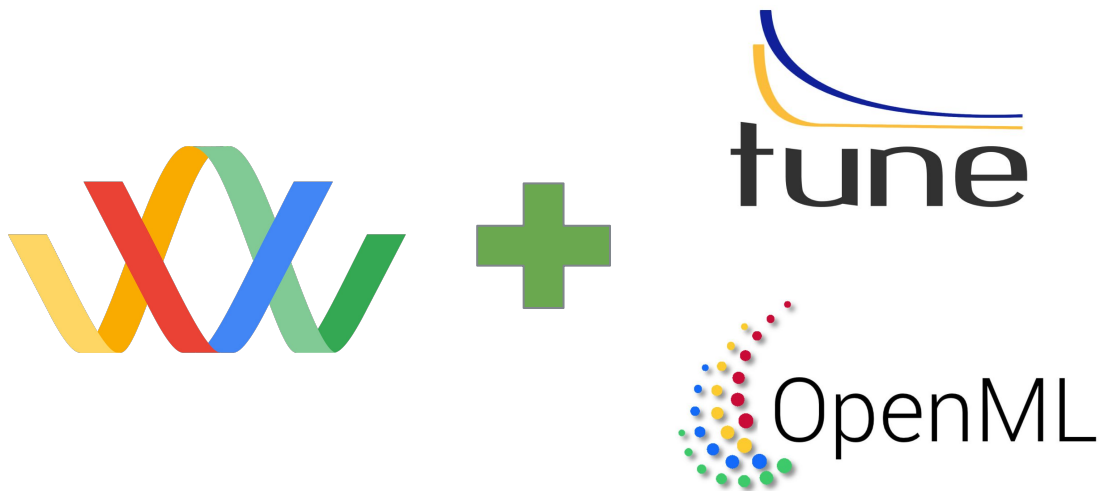
Send feedback

Vertex AI Vizier is a black-box optimization service that helps you tune hyperparameters in complex machine learning (ML) models. When ML models have many different hyperparameters, it can be difficult and time consuming to tune them manually. Vertex AI Vizier optimizes your model's output by tuning the hyperparameters for you.

# Future

# Potential + On-going External Integrations

- [Cross-study integration w/ OpenML](#)



# Algorithms

- Upcoming GP-UCB-PE algorithm
  - PE = “Pure Exploration”
- Baseline reimplementations
  - Ex: [HEBO](#), [TuRBO](#)
- Upcoming whitepaper on Vizier’s GP algorithms
  - Exact descriptions to allow reproducibility
  - Comparisons to existing packages

Pure Exploration in Finitely-Armed and Continuous-Armed Bandits

Sébastien Bubeck\*

*INRIA Lille – Nord Europe, Sequel project,  
40 avenue Halley, 59650 Villeneuve d’Ascq, France*

Rémi Munos\*

*INRIA Lille – Nord Europe, Sequel project,  
40 avenue Halley, 59650 Villeneuve d’Ascq, France*

Gilles Stoltz\*

*Ecole Normale Supérieure, CNRS  
75005 Paris, France  
et  
HEC Paris, CNRS,  
78351 Jouy-en-Josas, France*



*Heteroscedastic Evolutionary Bayesian Optimisation*

