Non-stochastic Best Arm Identification and Hyperparameter Optimization

AISTATS 2026 Test of Time Award

Kevin Jamieson
Associate Professor
University of Washington

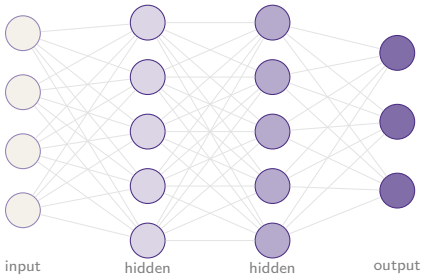
Ameet Talwalkar
Associate Professor
Carnegie Mellon University

April 29, 2026

What are hyperparameters?

regularization
dropout, weight decay

optimizer
lr, momentum, schedule



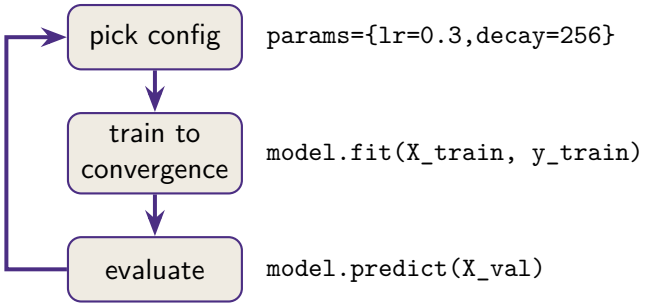
data pipeline
batch size, augmentation

depth & width
(layers, neurons per layer)

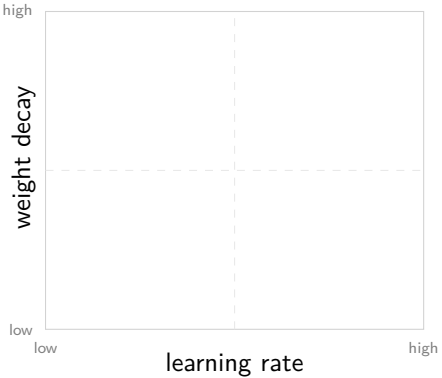
Hyperparameters are inputs to the learning algorithm — **not** learned by gradient descent. A modern model can have dozens of them, and getting them right matters enormously.

Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

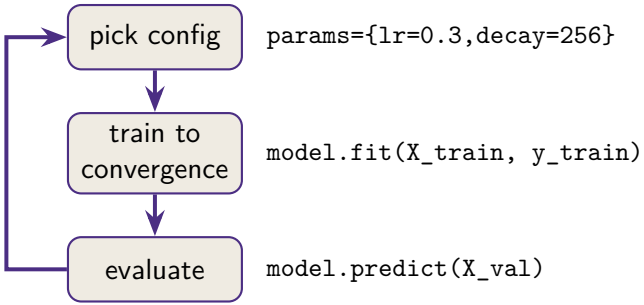


Each point = one full training run.

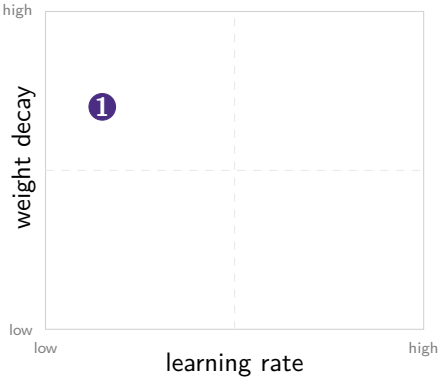
With a small budget, you try very few configs.

Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

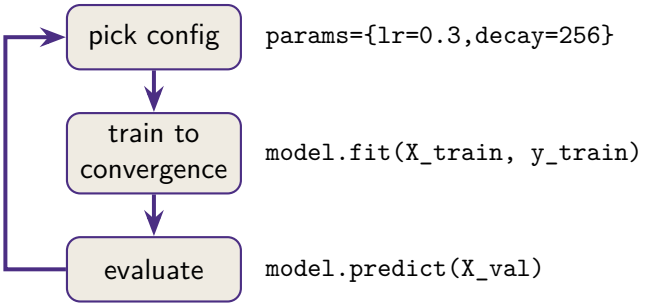


Each point = one full training run.

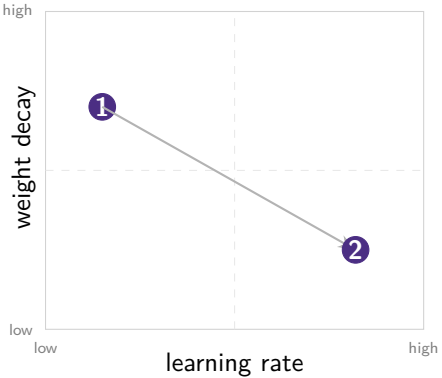
With a small budget, you try very few configs.

Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

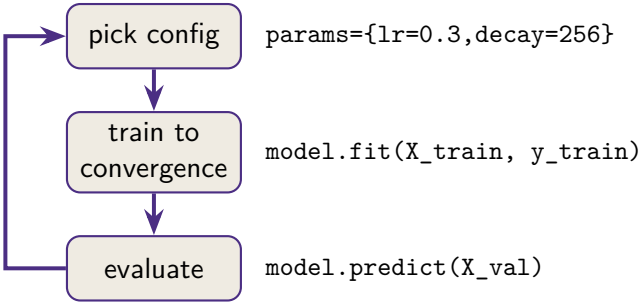


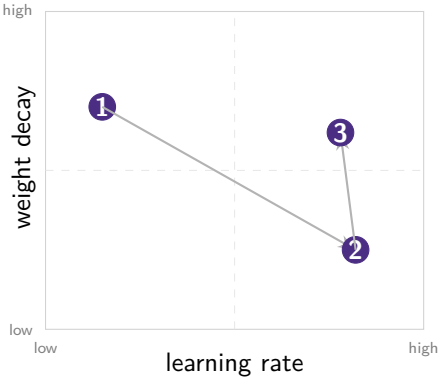
Each point = one full training run.

With a small budget, you try very few configs.

Hyperparameter tuning, traditionally

The standard paradigm:





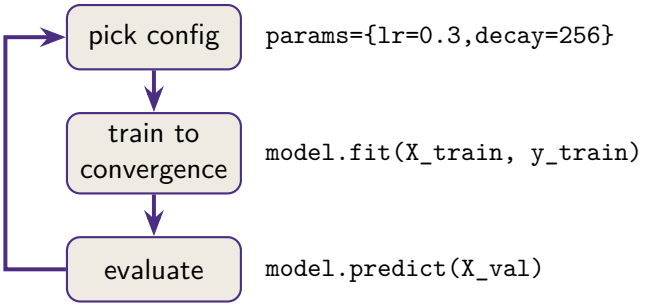
Each config requires a **full run** before you learn anything.

Each point = one full training run.

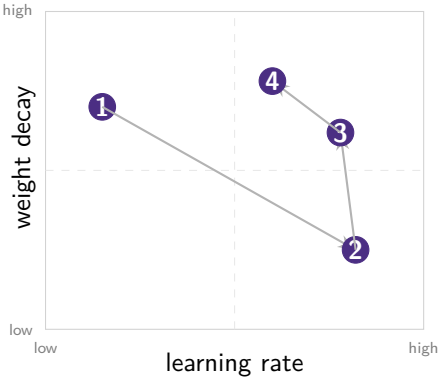
With a small budget, you try very few configs.

Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

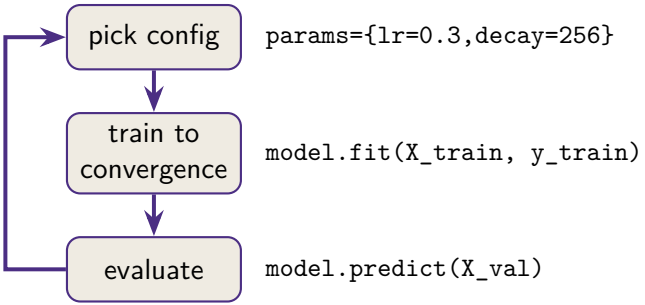


Each point = one full training run.

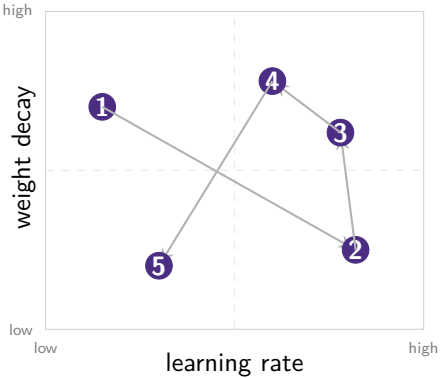
With a small budget, you try very few configs.

Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

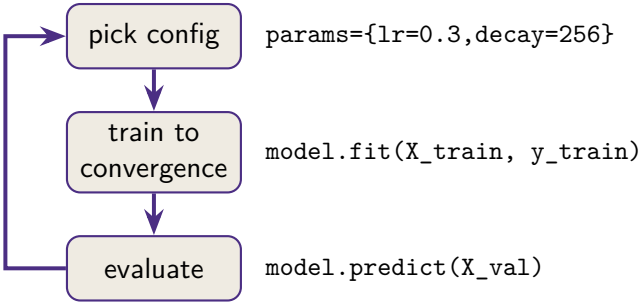


Each point = one full training run.

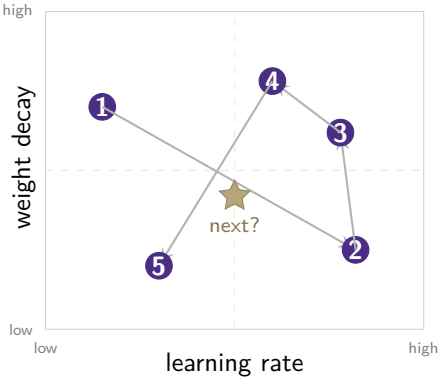
With a small budget, you try very few configs.

Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

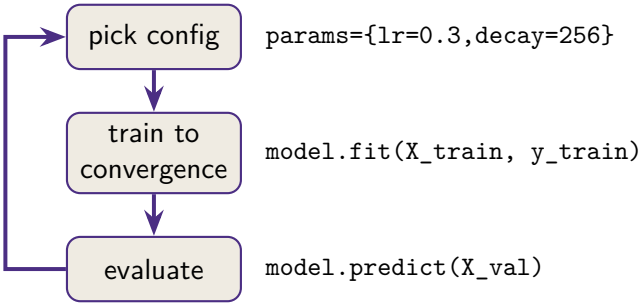


Each point = one full training run.

With a small budget, you try very few configs.

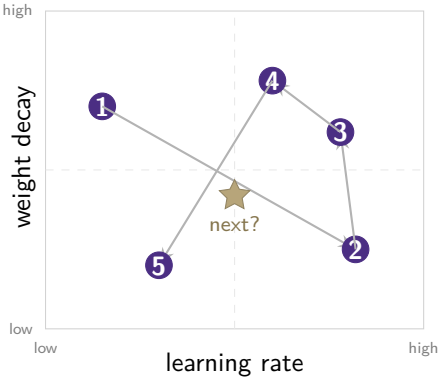
Hyperparameter tuning, traditionally

The standard paradigm:



Each config requires a **full run** before you learn anything.

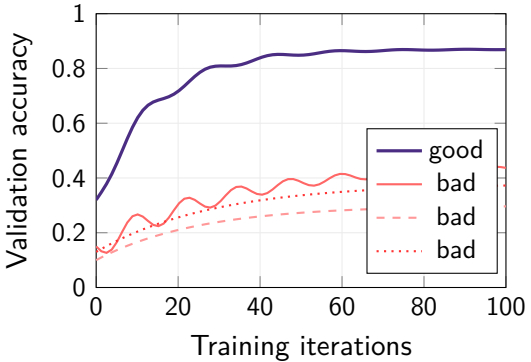
Grad Student Descent



Each point = one full training run.

With a small budget, you try very few configs.

Our insight: training is iterative

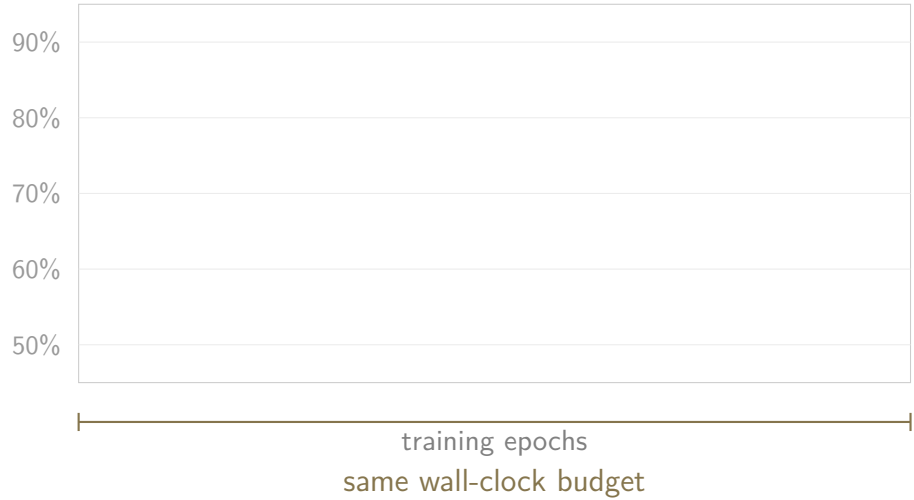


Bad configurations look bad early.

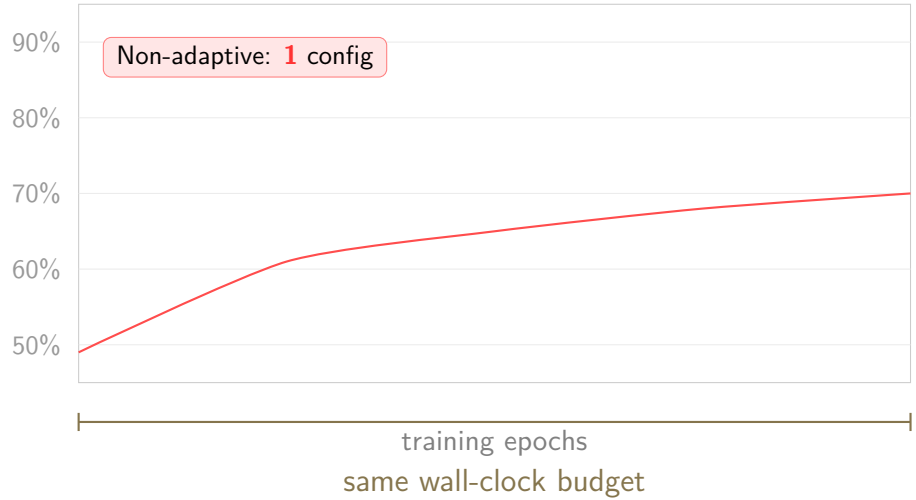
So: **kill them early** and spend that compute on new configs instead.

See far more configs in the same budget.

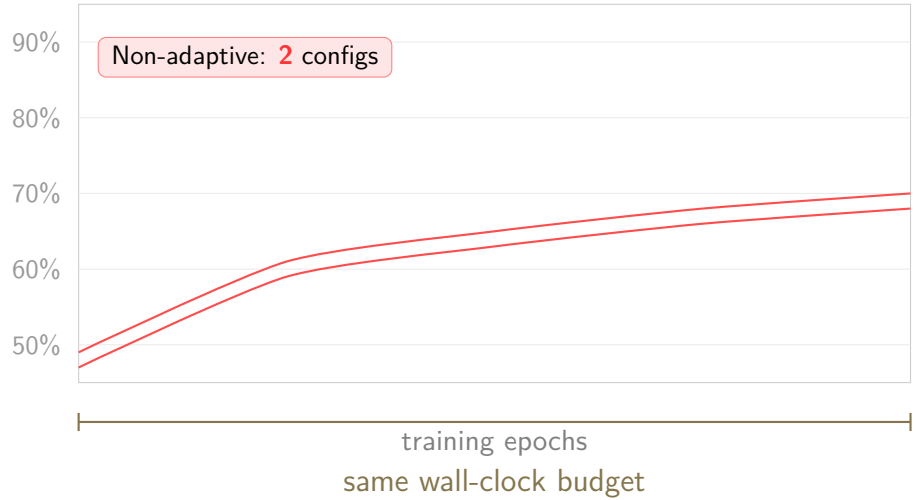
Same budget — many more configs



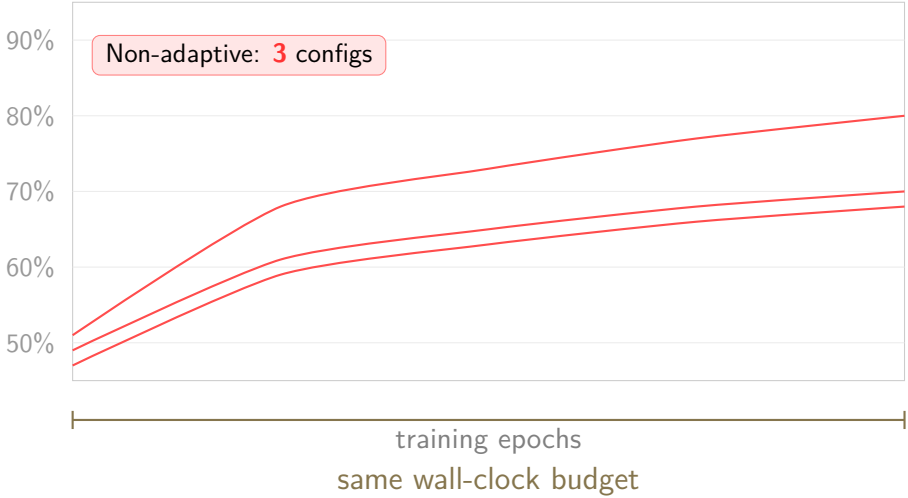
Same budget — many more configs



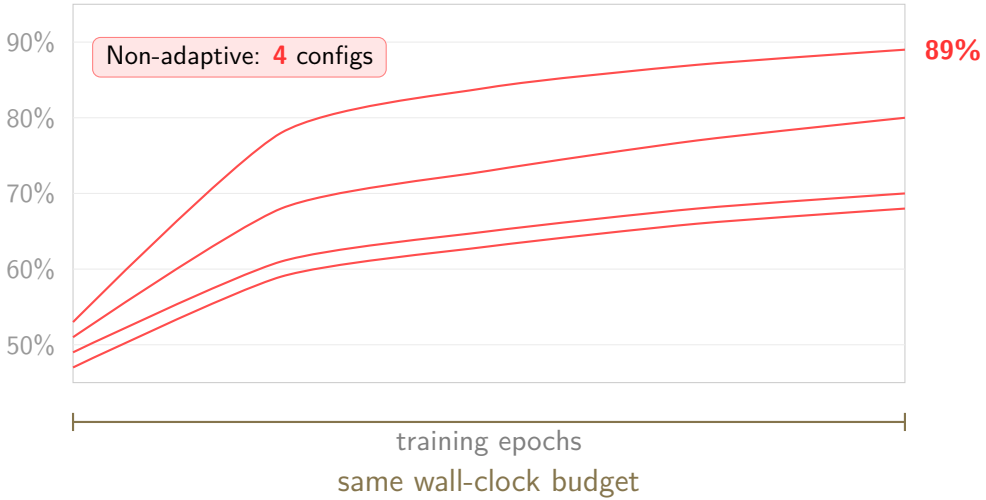
Same budget — many more configs



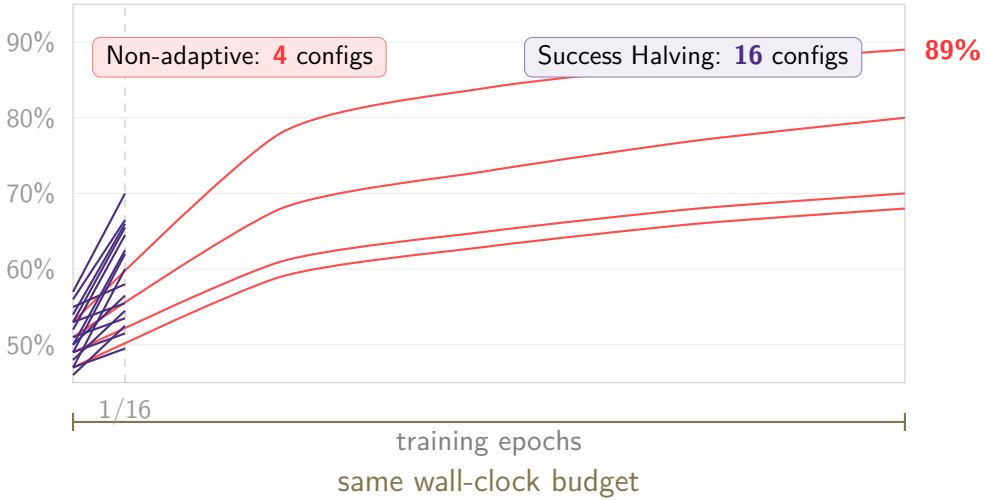
Same budget — many more configs



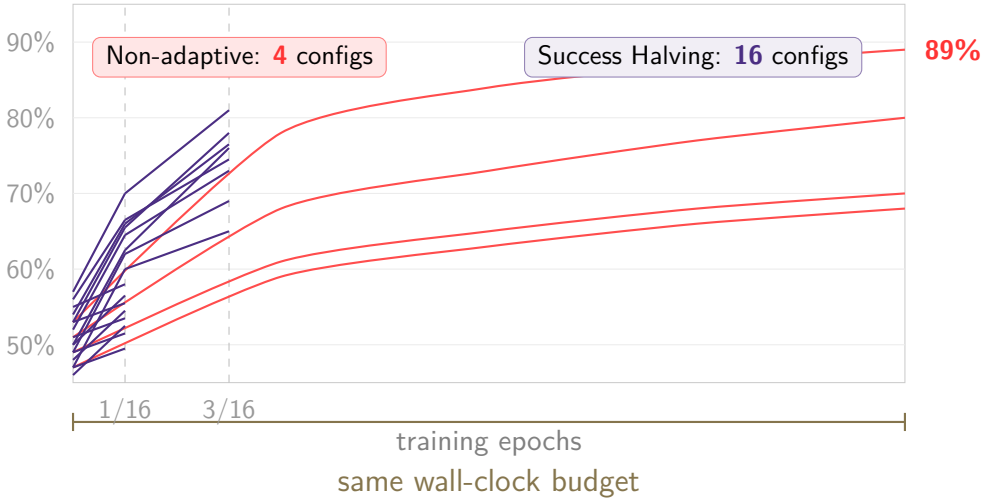
Same budget — many more configs



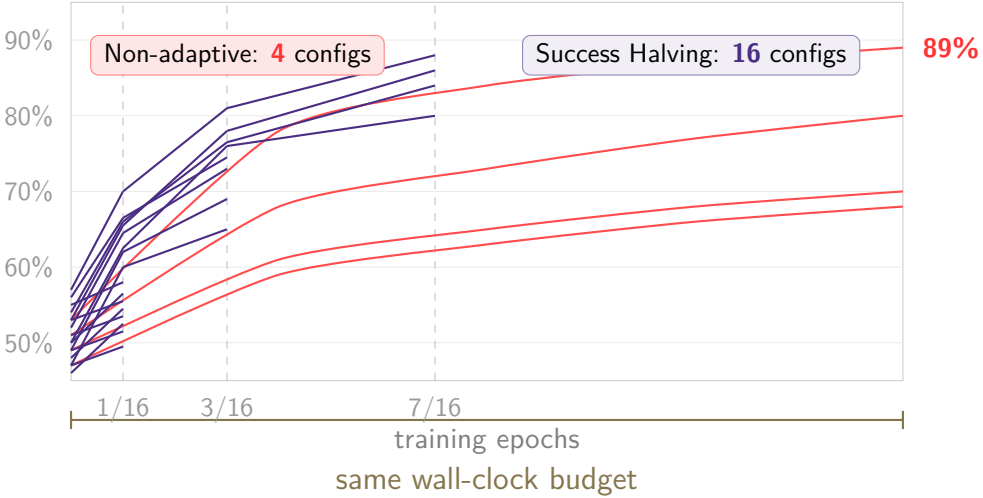
Same budget — many more configs



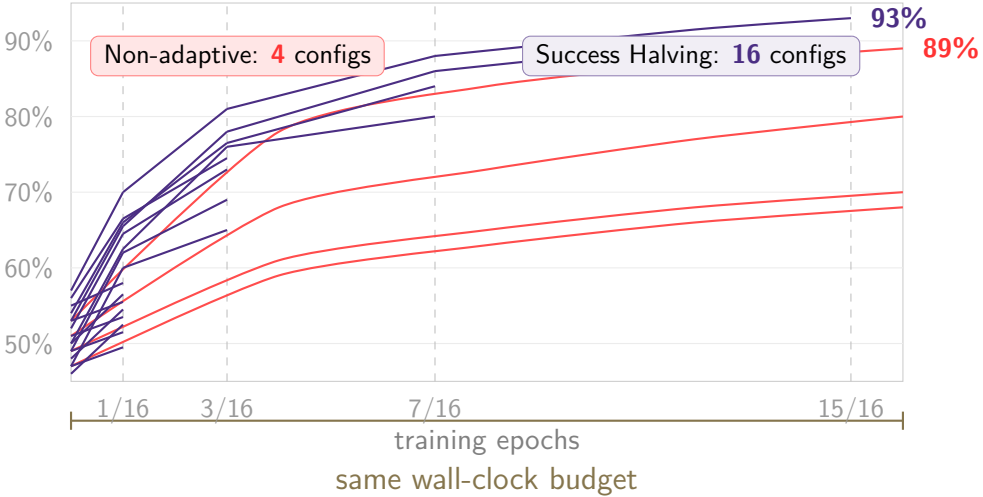
Same budget — many more configs



Same budget — many more configs



Same budget — many more configs



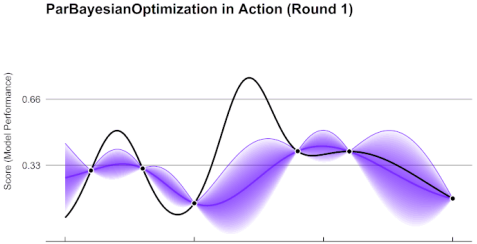
Roadmap for today

1. State of hyperparameter tuning in 2016
2. Inspiration for the algorithm
3. Hyperband and extensions
4. Final thoughts: LLMs

The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat



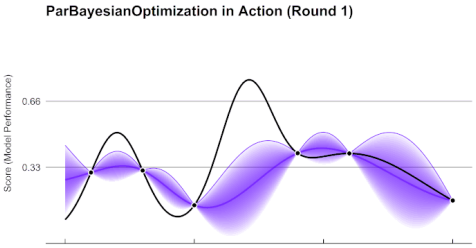
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



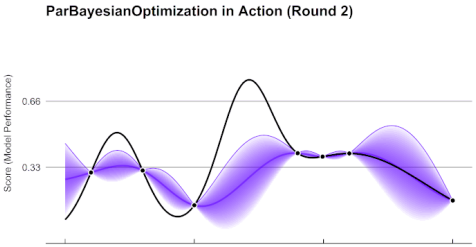
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



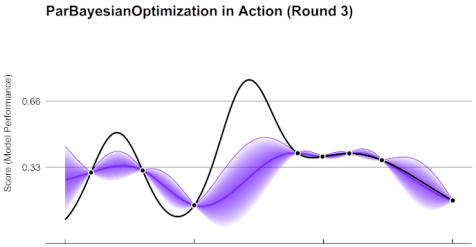
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



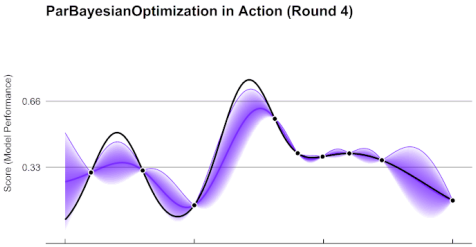
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



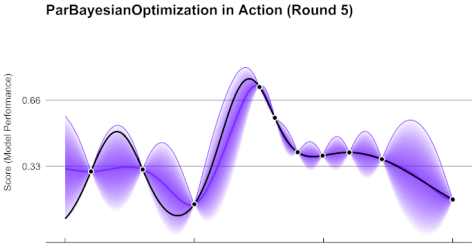
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



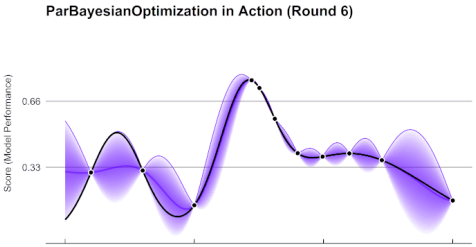
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



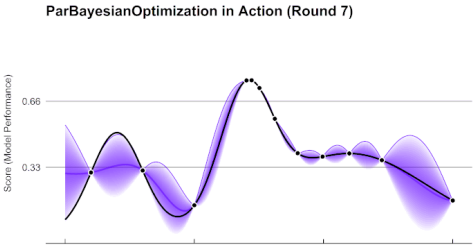
The world in 2016: Bayesian optimization reigns

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML



Bayesian optimization: a genuinely powerful idea

The core framework:

- Fit a **Gaussian process** surrogate to observed (config, loss) pairs
- Query the point that maximizes an **acquisition function**; observe loss; update the GP; repeat

Landmark papers:

Srinivas et al. (2010) — GP-UCB, theoretical regret bounds
Bergstra et al. (2011) — TPE, tree-structured search
Hutter et al. (2011) — SMAC, random forests as surrogate
Snoek et al. (2012) — Spearmint, practical BO for ML
Swersky et al. (2014) — *Freeze-Thaw*: Learning curves

Why it's appealing

- Principled uncertainty quantification, known sample complexity guarantees
- Balances exploration & exploitation
- Works well in low dimensions ($d \lesssim 5$) with smooth, stationary objectives

Where Bayesian optimization struggles

1. Unknown smoothness

GP performance depends critically on the *kernel* and *prior* that encodes smoothness of the *true* objective. This is itself unknown—and must be estimated.

Where Bayesian optimization struggles

1. Unknown smoothness

GP performance depends critically on the *kernel* and *prior* that encodes smoothness of the *true* objective. This is itself unknown—and must be estimated.

2. Warm-up cost

The first $\sim$10–20 evaluations are quasi-random to fit kernel hyper-hyperparameters.

Where Bayesian optimization struggles

1. Unknown smoothness

GP performance depends critically on the *kernel* and *prior* that encodes smoothness of the *true* objective. This is itself unknown—and must be estimated.

2. Warm-up cost

The first $\sim$10–20 evaluations are quasi-random to fit kernel hyper-hyperparameters.

3. Curse of dimensionality

Hyperparameter spaces for modern models can have dozens of dimensions. Sample complexity of any non-trivial GP scales exponentially in dimension.

Where Bayesian optimization struggles

1. Unknown smoothness

GP performance depends critically on the *kernel* and *prior* that encodes smoothness of the *true* objective. This is itself unknown—and must be estimated.

2. Warm-up cost

The first $\sim$10–20 evaluations are quasi-random to fit kernel hyper-hyperparameters.

3. Curse of dimensionality

Hyperparameter spaces for modern models can have dozens of dimensions. Sample complexity of any non-trivial GP scales exponentially in dimension.

4. Ignores iterative training

Standard BO treats each evaluation as a unit: start training, wait for convergence, observe final loss.

Freeze-Thaw (Swersky et al. 2014)

Bayesian optimization over parameterized *learning curve extrapolations*. A beautiful idea but also another parameter to tune.

Where Bayesian optimization struggles

1. Unknown smoothness

GP performance depends critically on the *kernel* and *prior* that encodes smoothness of the *true* objective. This is itself unknown—and must be estimated.

2. Warm-up cost

The first ∼10–20 evaluations are quasi-random to fit kernel hyper-hyperparameters.

3. Curse of dimensionality

Hyperparameter spaces for modern models can have dozens of dimensions. Sample complexity of any non-trivial GP scales exponentially in dimension.

4. Ignores iterative training

Standard BO treats each evaluation as a unit: start training, wait for convergence, observe final loss.

Freeze-Thaw (Swersky et al. 2014)

Bayesian optimization over parameterized *learning curve extrapolations*. A beautiful idea but also another parameter to tune.

5. Acquisition function optimisation

Maximising EI / UCB is itself a **non-convex global optimisation** problem. You replaced one hard problem with another.

Where Bayesian optimization struggles

1. Unknown smoothness

GP performance depends critically on the *kernel* and *prior* that encodes smoothness of the *true* objective. This is itself unknown—and must be estimated.

2. Warm-up cost

The first $\sim$10–20 evaluations are quasi-random to fit kernel hyper-hyperparameters.

3. Curse of dimensionality

Hyperparameter spaces for modern models can have dozens of dimensions. Sample complexity of any non-trivial GP scales exponentially in dimension.

4. Ignores iterative training

Standard BO treats each evaluation as a unit: start training, wait for convergence, observe final loss.

Freeze-Thaw (Swersky et al. 2014)

Bayesian optimization over parameterized *learning curve extrapolations*. A beautiful idea but also another parameter to tune.

5. Acquisition function optimisation

Maximising EI / UCB is itself a **non-convex global optimisation** problem. You replaced one hard problem with another.

But it works in practice, right?

How it started

Evan Sparks found SOTA hyperparameter packages to be **ineffective**. And moreover, *early* SGD loss was surprisingly predictive of final performance.

Co-author **Ameet Talwalkar** found this curious and recognized it as a potential bandit problem. He asked me what I thought.

Sparks, Talwalkar, Franklin, Jordan, Kraska. *TuPAQ: An Efficient Planner for Large-Scale Predictive Analytic Queries.* SoCC 2015.

We ran our own experiments

We benchmarked SMAC, TPE, and random search on **117 real hyperparameter datasets** from Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In NIPS, 2015.

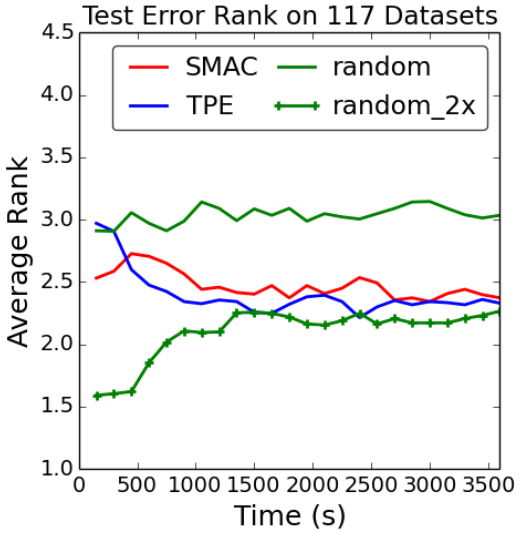
- Intrinsic dimensionality ≈ 15
- 200–400+ evaluations — well into the regime where BO should shine
- Metric: **average rank** on test error over time

We ran our own experiments

We benchmarked SMAC, TPE, and random search on **117 real hyperparameter datasets** from Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In NIPS, 2015.

- Intrinsic dimensionality ≈ 15
- 200–400+ evaluations — well into the regime where BO should shine
- Metric: **average rank** on test error over time

The punchline: BO beats Random. But `random_2x` — random search at twice speed — beats BO methods.



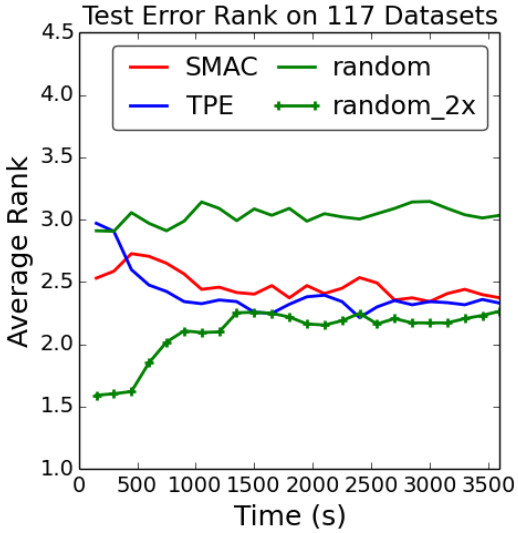
We ran our own experiments

We benchmarked SMAC, TPE, and random search on **117 real hyperparameter datasets** from Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In NIPS, 2015.

- Intrinsic dimensionality ≈ 15
- 200–400+ evaluations — well into the regime where BO should shine
- Metric: **average rank** on test error over time

The punchline: BO beats Random. But `random_2x` — random search at twice speed — beats BO methods.

Can we speed up random search?



How it started

Evan Sparks found SOTA hyperparameter packages to be **ineffective**. And moreover, *early* SGD loss was surprisingly predictive of final performance.

Co-author **Ameet Talwalkar** found this curious and recognized it as a potential bandit problem. He asked me what I thought.

Evan's discard heuristic

Evan's heuristic discarded half configs at equally spaced intervals. Reminded me of **Successive Halving** (Karnin, Koren, Somekh, 2013)—an algorithm from the *stochastic* best-arm identification literature.

Sparks, Talwalkar, Franklin, Jordan, Kraska. *TuPAQ: An Efficient Planner for Large-Scale Predictive Analytic Queries.* SoCC 2015.

How it started

Evan Sparks found SOTA hyperparameter packages to be **ineffective**. And moreover, *early* SGD loss was surprisingly predictive of final performance.

Co-author **Ameet Talwalkar** found this curious and recognized it as a potential bandit problem. He asked me what I thought.

Sparks, Talwalkar, Franklin, Jordan, Kraska. *TuPAQ: An Efficient Planner for Large-Scale Predictive Analytic Queries.* SoCC 2015.

Evan's discard heuristic

Evan's heuristic discarded half configs at equally spaced intervals. Reminded me of **Successive Halving** (Karnin, Koren, Somekh, 2013)—an algorithm from the *stochastic* best-arm identification literature.

The open question

Can we guarantee anything about the performance of Successive Halving on non-stochastic training curves?

Stochastic best-arm identification

The problem

n **arms.** Pulling arm i yields an i.i.d. reward $X_{i,t}$ with *unknown* mean μ_i.

Goal: identify $i^* = \arg\max_i \mu_i$ using as few total pulls as possible.

The problem

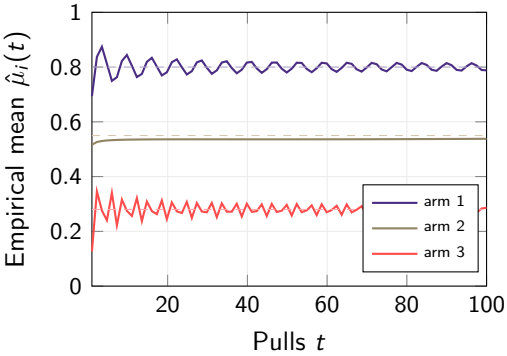
n **arms.** Pulling arm i yields an i.i.d. reward $X_{i,t}$ with *unknown* mean μ_i.

Goal: identify $i^* = \arg\max_i \mu_i$ using as few total pulls as possible.

Empirical means converge **predictably**:

$$\hat{\mu}_i(t) = \frac{1}{t}\sum_{s=1}^{t} X_{i,s} \longrightarrow \mu_i \quad \text{at rate } 1/\sqrt{t}.$$

Once $\hat{\mu}_i(t)$ is close enough to μ_i, we can **reliably rank and eliminate** the worst arms.



Unknown convergence envelopes γ_i

Loss $\ell_{i,t}$ after t epochs is **deterministic** — no randomness, no i.i.d. assumption.

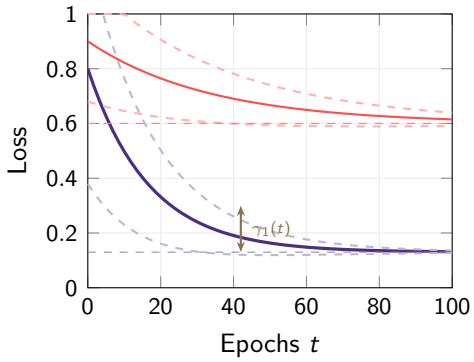
It does converge: $\ell_{i,t} \to \nu_i$ as $t \to \infty$.

Define the **tightest** envelope $\gamma_i(t)$ such that

$$|\ell_{i,t} - \nu_i| \;\leq\; \gamma_i(t) \quad \forall\, t.$$

$\gamma_i(t)$ shrinks as $\ell_{i,t}$ approaches ν_i.

ν_i and γ_i are both completely unknown.



The same algorithm — a very different analysis

Theorem (Jamieson & Talwalkar, 2016)

Define the problem hardness

$$H = \sum_{i=2}^{n} \gamma_i^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right).$$

If $B \gtrsim \lceil \log_2 n \rceil \cdot H$, Successive Halving returns the best arm.

The algorithm is identical to the stochastic setting. Very different analysis.

The same algorithm — a very different analysis

Theorem (Jamieson & Talwalkar, 2016)

Define the problem hardness

$$H = \sum_{i=2}^{n} \gamma_i^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right).$$

If $B \gtrsim \lceil \log_2 n \rceil \cdot H$, Successive Halving returns the best arm.

The algorithm is identical to the stochastic setting. Very different analysis.

Uniform allocation can be far worse

Sample complexity of uniform allocation is

$$n \max_{i>1} \gamma_i^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right)$$

The catch: when do you start discarding?

Discard too early

Good configs haven't had time to separate from bad ones. You may eliminate the true best arm before it reveals itself.

Discard too late

Everyone runs to near-convergence before any cut. No better than running each config fully. Budget wasted.

SH requires choosing when to make the first cut.

Can we avoid this choice entirely?

The catch: when do you start discarding?

Discard too early

Good configs haven't had time to separate from bad ones. You may eliminate the true best arm before it reveals itself.

Discard too late

Everyone runs to near-convergence before any cut. No better than running each config fully. Budget wasted.

Solution: Run several SH brackets in parallel, each with a *different* aggressiveness.

This is Hyperband (2017).

Liam Li

Kevin Jamieson

Giulia DeSalvo

Afshin Rostamizadeh

Ameet Talwalkar

Extensions

BOHB (Falkner et al., 2018)

- Replace random sampling in Hyperband with Bayesian optimization (TPE-style) for config *selection*.
- Retains the SH early-stopping backbone.
- I was resistant because of all the issues with BO, but empirically it works very well.

ASHA (Li et al., ICLR 2020 / Determined AI)

- Synchronous SH stalls on stragglers in distributed settings.
- **Asynchronous SHA:** promote configurations as soon as they hit a rung, without waiting for peers.
- Scales **linearly** with workers; tested up to 500.
- Outperforms PBT, BOHB, and Google Vizier in production benchmarks.

Where does Hyperband fit with LLMs?

What I was told

"We train several **small models** on modest data. The best ones we **make bigger** and train longer on more data. The best of those we scale up again."

Where does Hyperband fit with LLMs?

What I was told

"We train several **small models** on modest data. The best ones we **make bigger** and train longer on more data. The best of those we scale up again."

I make **no claim of influence**—scaling laws and the "small-to-large" paradigm emerged organically in the LLM community.

Where does Hyperband fit with LLMs?

What I was told

"We train several **small models** on modest data. The best ones we **make bigger** and train longer on more data. The best of those we scale up again."

I make **no claim of influence**—scaling laws and the "small-to-large" paradigm emerged organically in the LLM community.

But at least for now, it appears that resource optimization, not hyperparameter selection, remains the dominant paradigm.

Thanks

Special thanks to Evan Sparks, Ameet Talwalkar, Liam Li, Afshin Rostamizadeh, Giulia DeSalvo, Ben Recht, Moritz Hardt, Ekaterina Gonina, Yoram Singer.

Papers

Non-stochastic Best Arm Identification and Hyperparameter Optimization
Jamieson & Talwalkar (2016)

Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization
Li, Jamieson, DeSalvo, Rostamizadeh, Talwalkar (JMLR 2018)

A System for Massively Parallel Hyperparameter Tuning (ASHA) Li, Jamieson, Rostamizadeh, Gonina, Hardt, Recht, Talwalkar, (2020)

Try it

`ray.tune`
`optuna`
`keras-tuner`
`Scikit-learn`
`AWS SageMaker`
`Azure Machine Learning`
`Weights & Biases (W&B)`

Kevin Jamieson
University of Washington
`jamieson@cs.washington.edu`

Backup: Successive Halving pseudocode

Algorithm: Successive Halving

Input: budget B, arms $[n]$, losses $\ell_{i,k}$

$S_0 \leftarrow [n]$

for $k = 0, 1, \ldots, \lceil \log_2 n \rceil - 1$ **do**

  $r_k \leftarrow \lfloor B/(|S_k| \lceil \log_2 n \rceil) \rfloor$

  Pull each arm $i \in S_k$ for r_k steps; observe ℓ_{i,r_k}

  $S_{k+1} \leftarrow$ top $\lfloor |S_k|/2 \rfloor$ arms by ℓ_{i,r_k}

Output: singleton of $S_{\lceil \log_2 n \rceil}$

Total observed losses: $\displaystyle\sum_{k=0}^{\lceil \log_2 n \rceil - 1} |S_k| = 2n + 1$