Efficiently Learning Linear System Solvers for Fast Numerical Simulation

Misha Khodak

UW-Madison

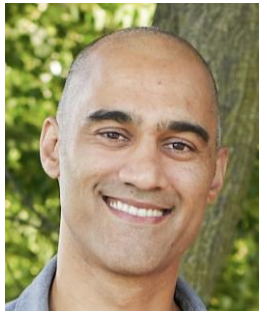
CS 839

19 February 2026

Collaborators

Edmond Chow

Nina Balcan



Ameet Talwalkar

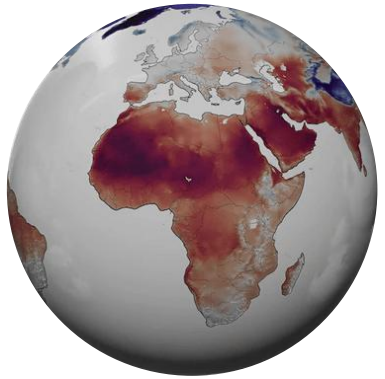
Min Ki Jung

Brian Wynne

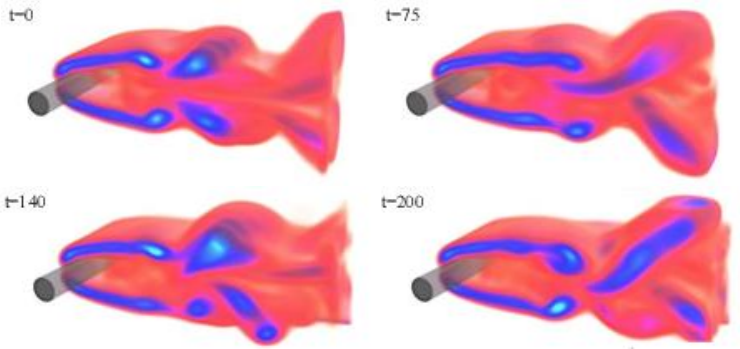
Egemen Kolemen

The promise of using **machine learning** for **scientific computing**

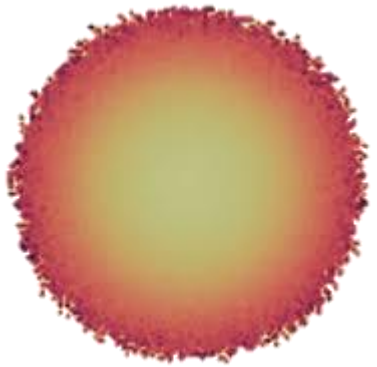
- real-time weather prediction
- fast inverse solvers using differentiable forward-pass simulation
- avoiding exponential scaling in many-particle systems



NVIDIA FourCastNet

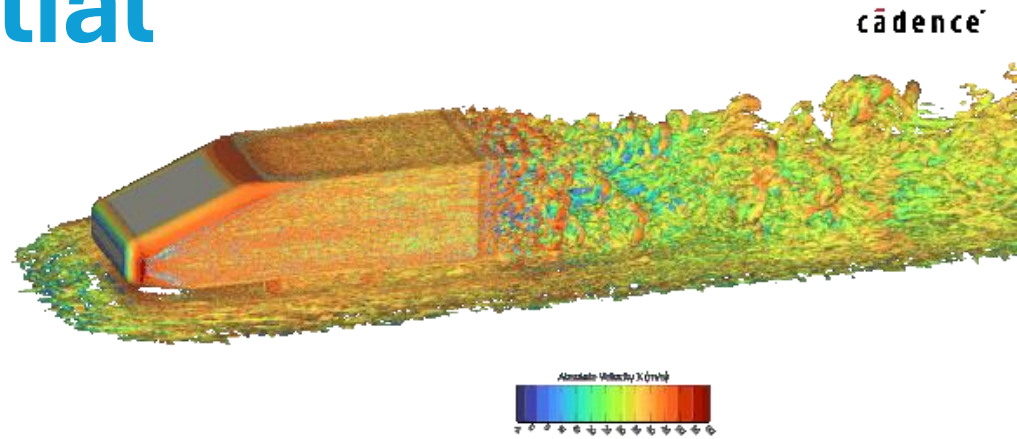


Um, Brand, Fei, Holl, & Thuerey



Boffi & Vanden-Eijnden

This talk: Solving **transient partial differential equations (PDEs)**



Goal: predict the evolution of a physical system from some initial conditions

Usually governed by a PDE relating different quantities across time and space

$$\nabla \cdot \overline{u} = 0$$

$$\rho \frac{D\overline{u}}{Dt} = -\nabla p + \mu \nabla^2 \overline{u} + \rho \overline{F}$$

Why?

- Scientists use simulation to gain understanding without running experiments
- Engineers use computer-assisted engineering (CAE) software for development
- Weather forecasting
- Physics visualization in movies/games
- Data-generation

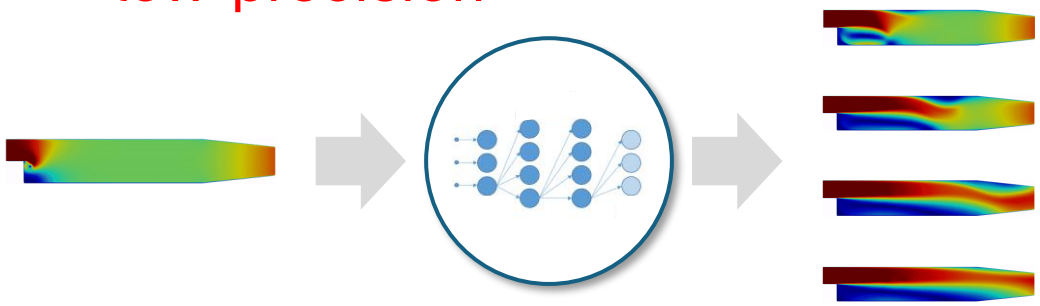
The Complicated Computer Modeling Behind an Animated Snowball

BY AISHA HARRIS NOV 26, 2013 • 12:02 PM

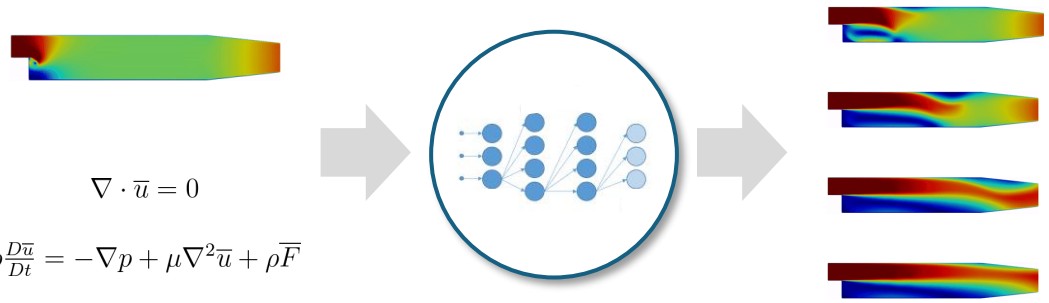
How can **machine learning** be used to **simulate transient PDEs** ?

Full replacement approaches (learning **instead of** simulating)

- neural operators
  [Lu et al., 2021; Li et al., 2021]
  - fast inference
  - needs classical training data
  - may not transfer to new settings
  - low-precision



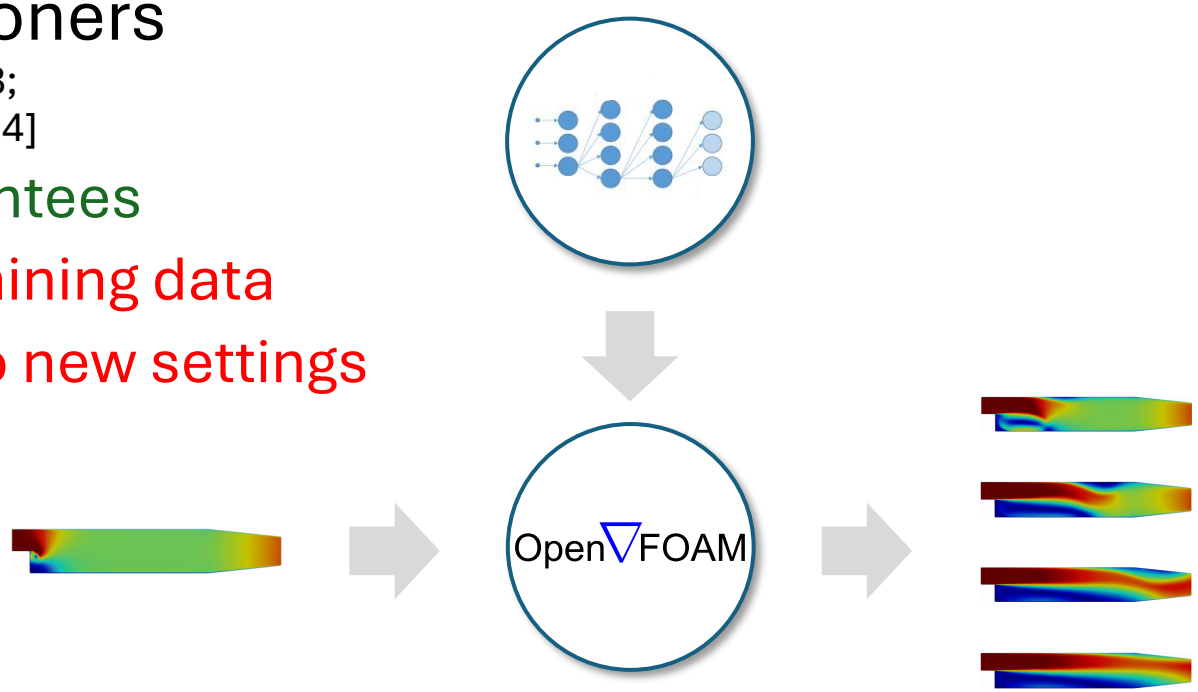
- physics-informed neural networks
  [Raissi et al., 2019]
  - can combine with experimental data
  - hard-to-train
  - local optima



How can **machine learning** be used to **simulate transient PDEs** ?

Integrationist approaches (learning **before** simulating)

- reduced-order models / learned preconditioners
  [Fresca et al., 2021; Li et al., 2023; Kopaničáková & Karniadakis, 2024]
  - correctness guarantees
  - needs classical training data
  - may not transfer to new settings



How can **machine learning** be used to **simulate transient PDEs** ?

Thus far:

- successful **training** of neural solvers
- no correctness guarantees, especially when transferring to new settings
- unclear compute improvement

Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations

Nick McGreivy & Ammar Hakim

Nature Machine Intelligence volume **6**, pages 1256–1269 (2024)

This talk: Learning **while** simulating

- integrates learning into classical solvers **without pre-training**
- correctness guarantees
- efficiency guarantees in model settings
- wallclock improvements **demonstrated within a popular OSS framework**

Outline

1. **PCGBandit:** integrating online learning into numerical simulation

2. **Learning guarantees** in a model setting

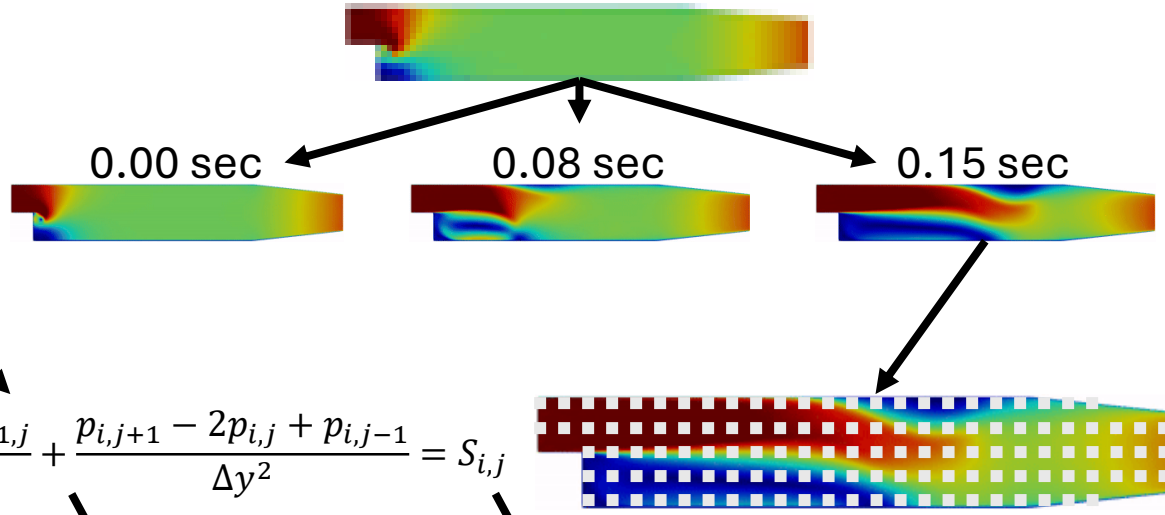
3. **Open directions**

The typical approach to solving a PDE

- discretize in time
- discretize in space
- pose the solution at timestep $t+1$ as the solution of a linear system
- solve the linear system and advance to the next timestep

$$\nabla^2 p = S$$

$$\nabla^2 p_{t+1} = S_t$$



$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} = S_{i,j}$$

$$\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} p_{1,1} \\ \vdots \\ p_{n_x,n_y} \end{bmatrix} = \begin{bmatrix} S_{1,1} \\ \vdots \\ S_{n_x,n_y} \end{bmatrix}$$

$$\mathbf{A}_t \ \mathbf{x}_{t+1} = \mathbf{b}_t$$

The typical approach to solving a linear system

- $\mathbf{A}$ usually too large for a direct solve ($n = \#\text{grid points}$)

- iterative methods get (very) close to the solution using matrix-vector products
  - achieve accuracy $\varepsilon = \frac{\|\mathbf{b}-\mathbf{A}\hat{\mathbf{x}}\|}{\|\mathbf{b}\|} \ll 1$ in $O_{\mathbf{A}}\left(\log\frac{1}{\varepsilon}\right)$ steps
  - each step takes $O\big(\text{nnz}(\mathbf{A})\big) = O(n)$ FLOPs

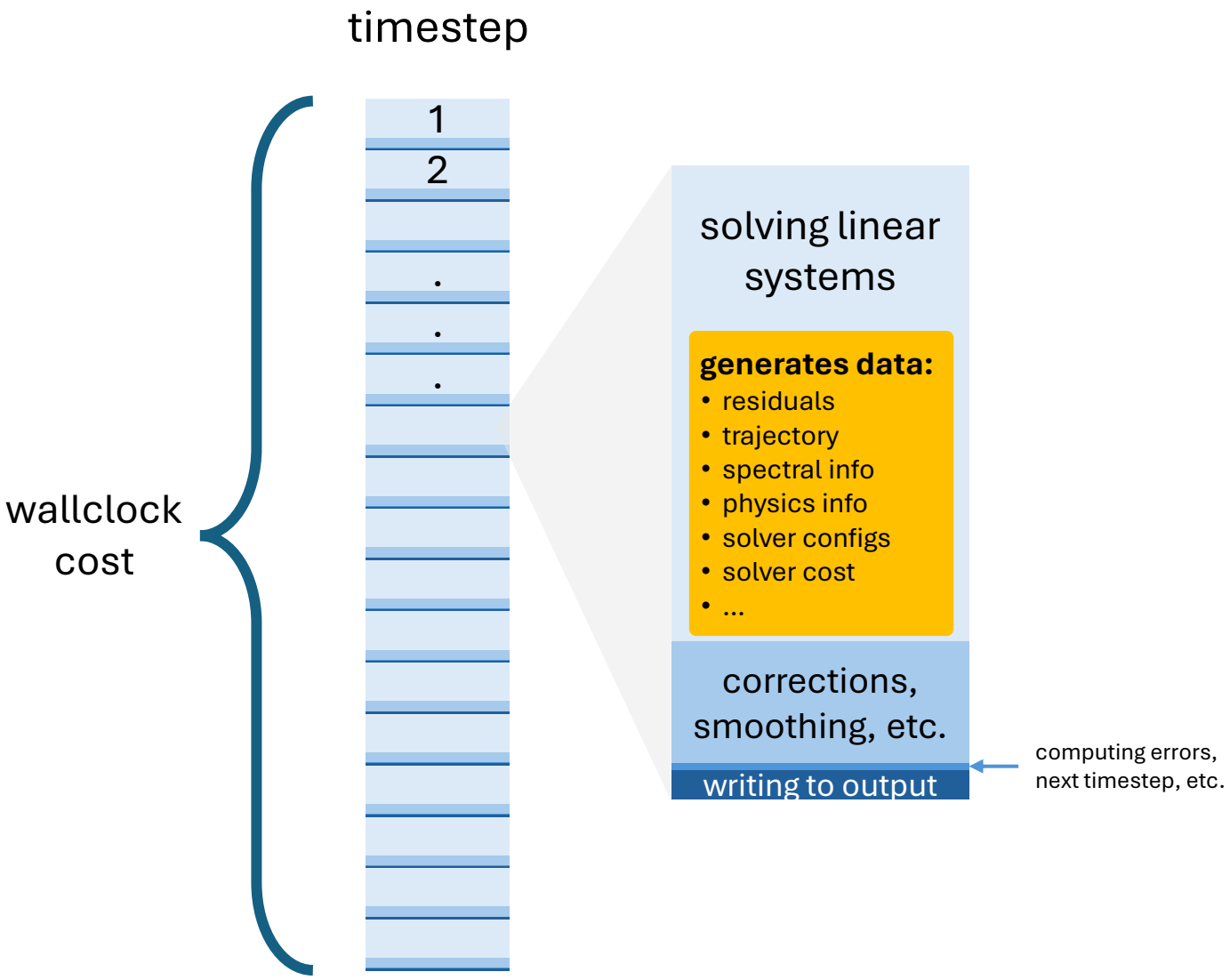
- most widely used algorithms are Krylov subspace methods
  - preconditioned conjugate gradient (**PCG**) for SPD matrices
  - GMRES for general matrices

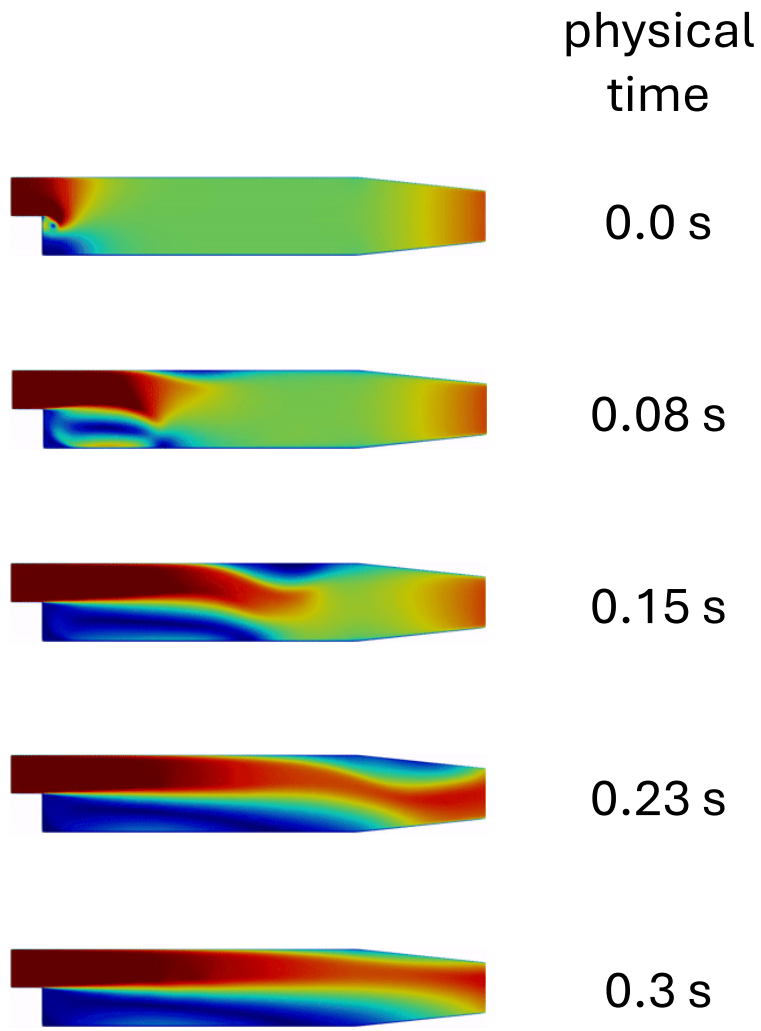
- in practice these **require a good preconditioner** to work well

What is a preconditioner?

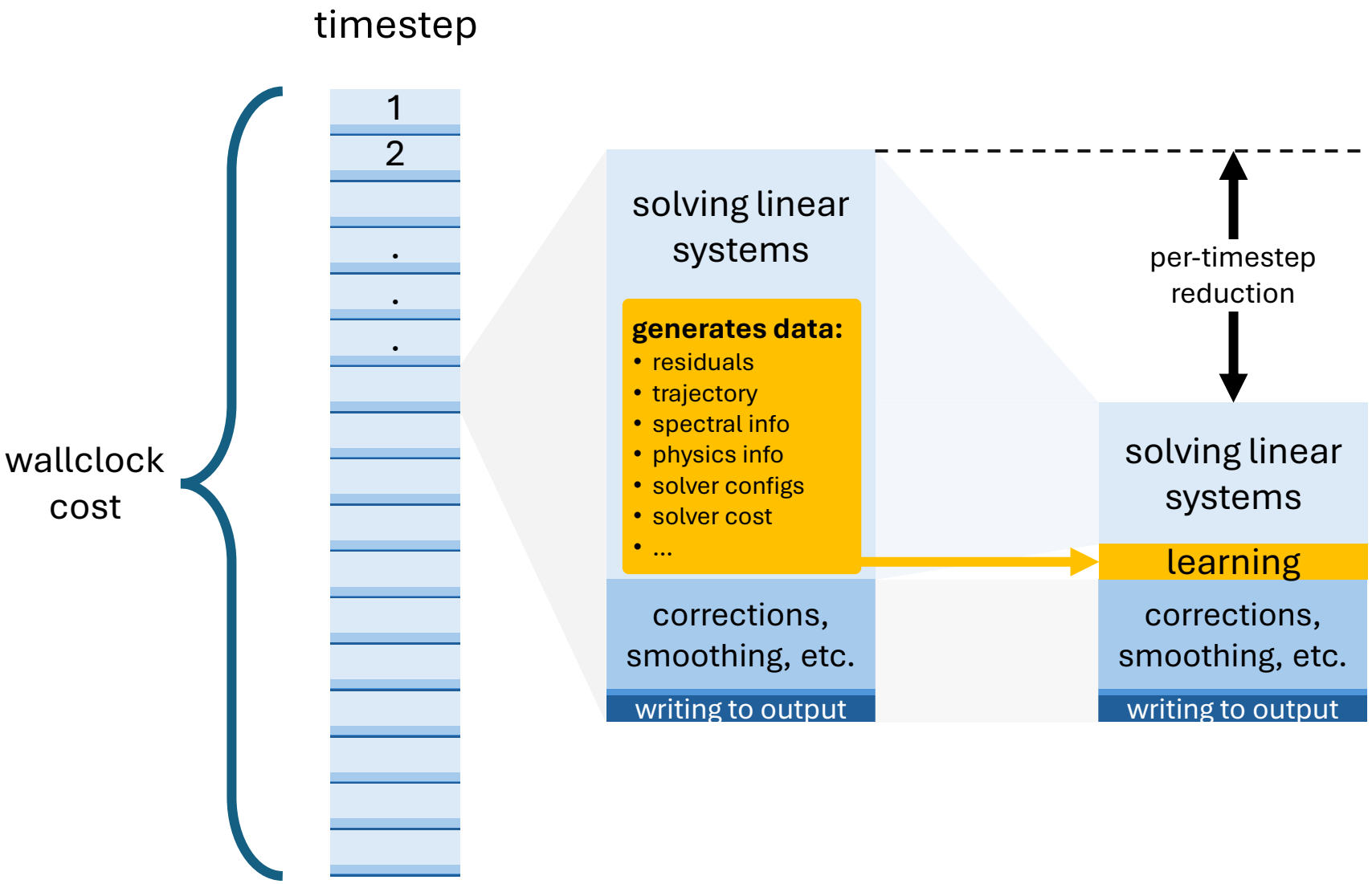
- a matrix $\mathbf{M}$ such that
  - $\mathbf{M} \approx \mathbf{A}$
  - multiplying $\mathbf{M}^{-1}$ by a vector is efficient
- many preconditioners have been developed
  - triangular products, low-rank, factorized, etc.
  - **performance varies significantly by problem**
- even when the asymptotically best preconditioner is known, **performance is highly dependent on its hyperparameters**
  - choice of multigrid smoothers, incomplete Cholesky drop tolerances, etc.
  - effect is hard to predict (due to physics, discretization, hardware, etc.)
- This talk: **how do we learn cost-optimal preconditioners?**

What is in the cost of a transient simulation?

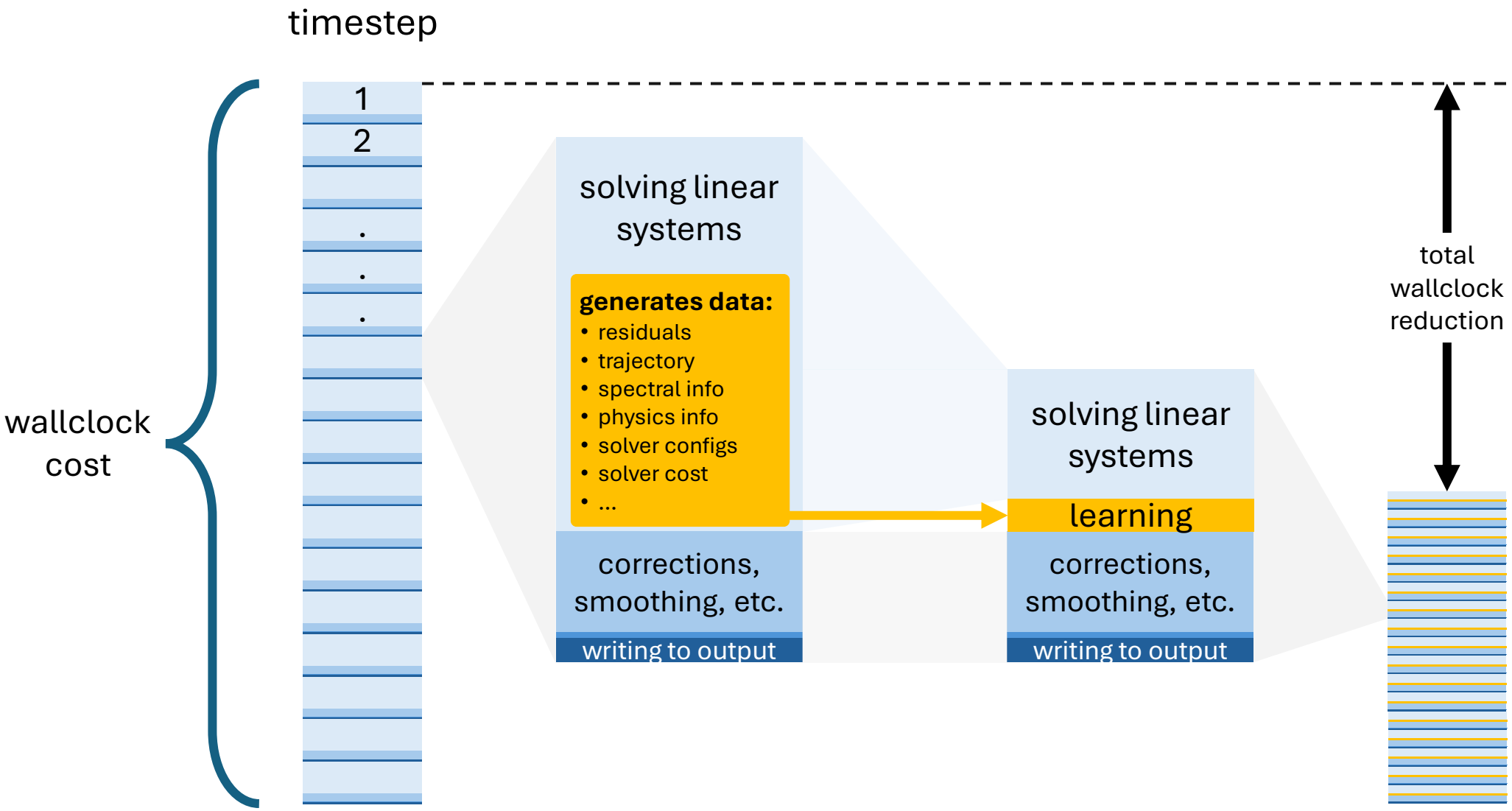




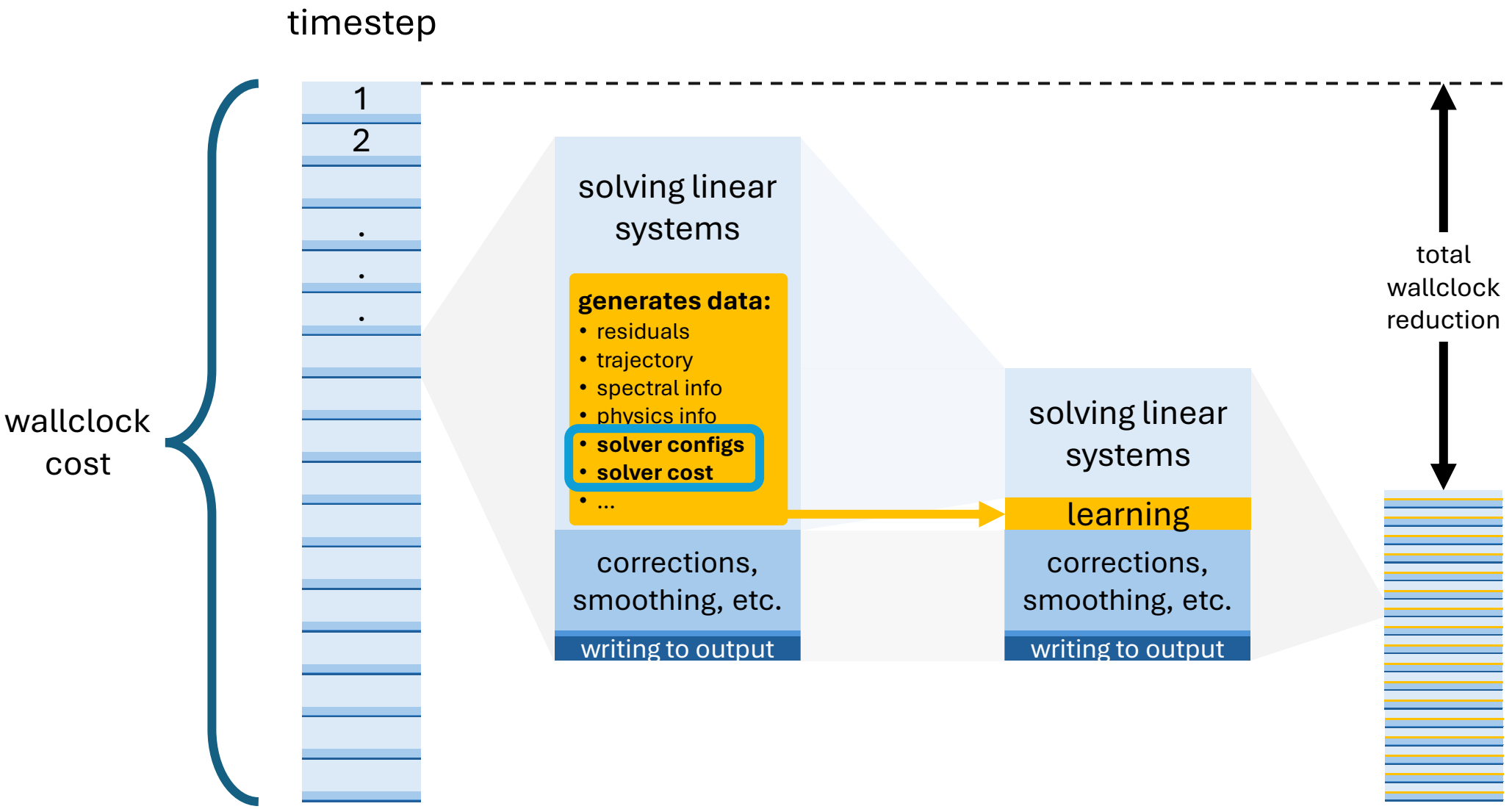
Reducing the cost with **in-simulation** data



Reducing the cost with **in-simulation** data



This talk: Learning from solver configs and costs



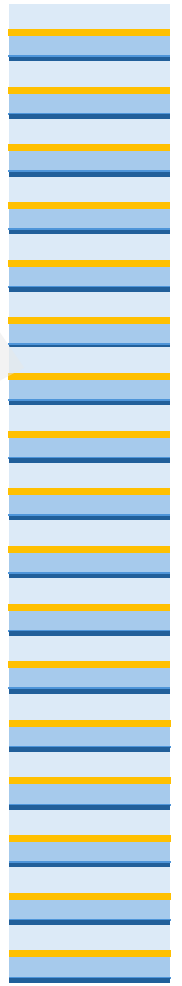
How are simulations usually configured?

At each timestep t :

- configure a solver, e.g. PCG
  - usually we just use the same config for every timestep of the simulation
- solve $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$ and record the cost in wallclock time

solving a linear system $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$

learning

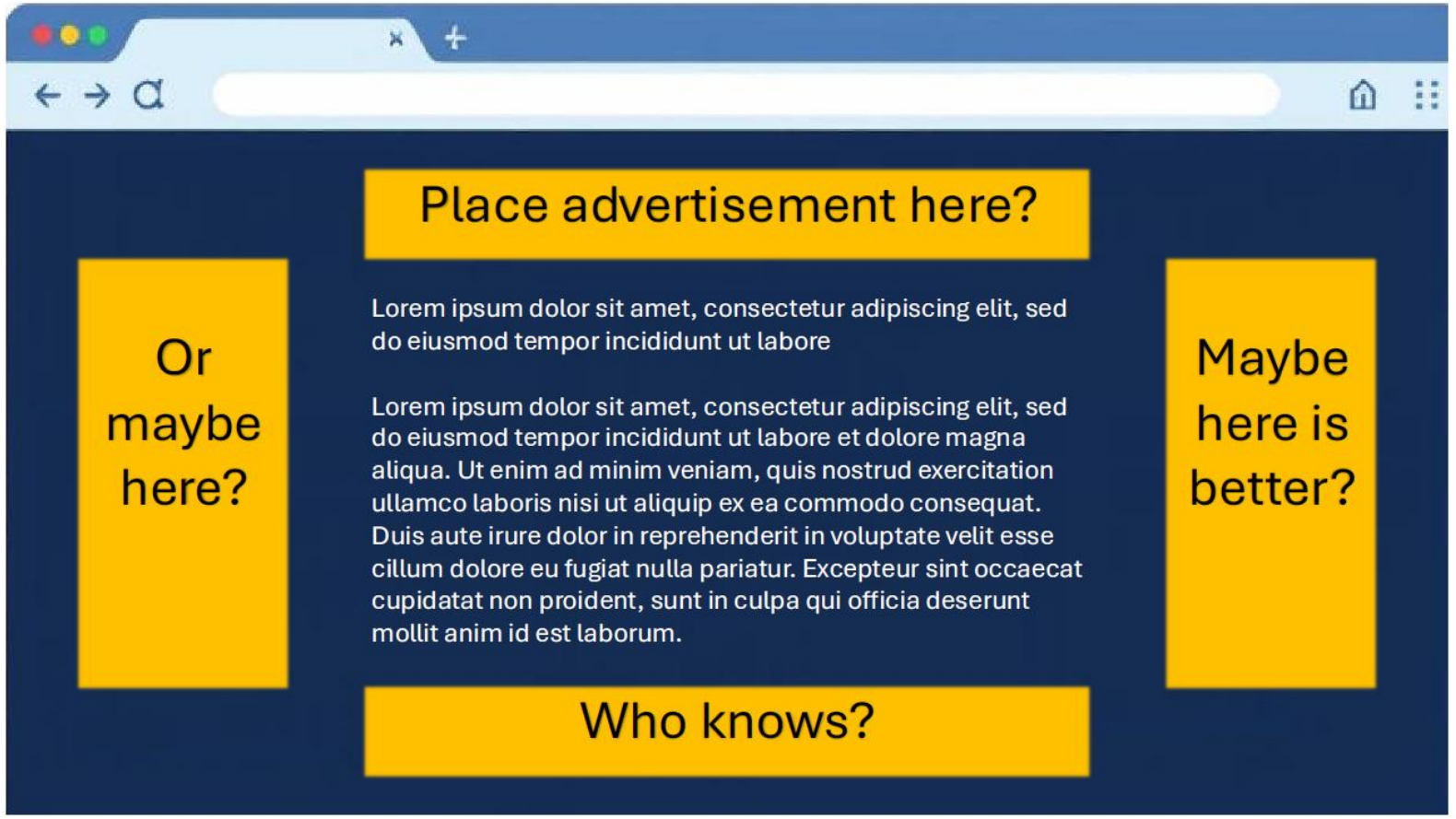


What if we have d different configurations to choose from?
How can we pick the best one without trying all of them?

A short detour into multi-armed bandits (MAB)

Original motivation:

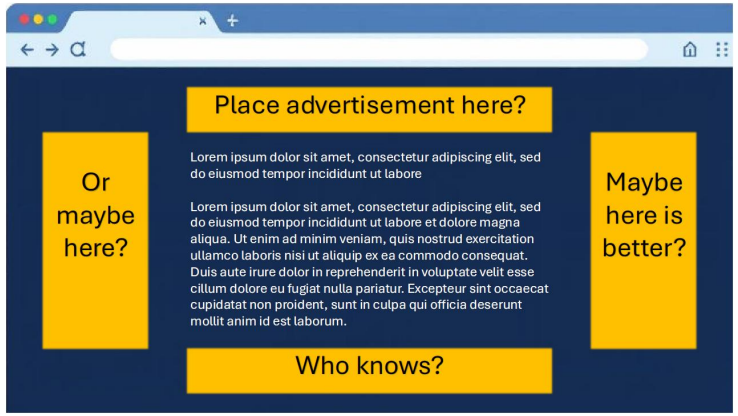
- a user clicks on my webpage
- **where to put an ad to maximize the probability of them clicking on it?**



Multi-armed bandits: Problem setup

Example: webpage ad placement

Four options: top, bottom, left, right



Modeled as a multi-round game:

for $t = 1, \ldots, T$ (new visits to webpage)
  choose action $i_t \in \{1, \ldots, d\}$ (d = 4)
  receive reward $\mathbf{r}_{t[i_t]} \in \mathbb{R}$ (1 if user clicks on ad, else 0)
    different actions lead to different rewards
    don't see rewards for actions not taken

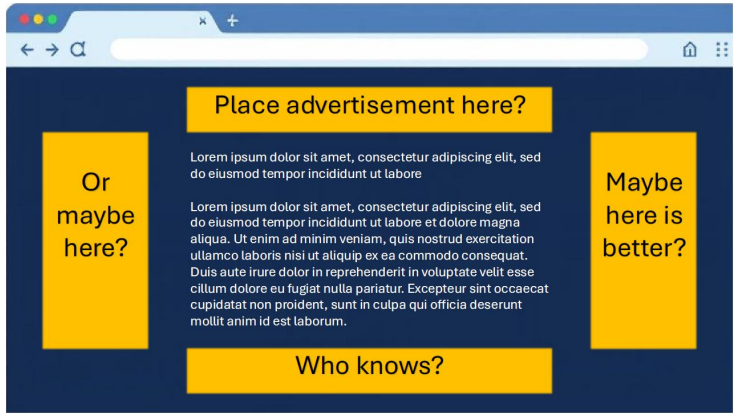
Goal: minimize the **cumulative regret**

$$\text{Regret} = \underbrace{\max_i \sum_{t=1}^{T} \mathbf{r}_{t[i]}}_{\text{reward of best fixed arm}} - \underbrace{\sum_{t=1}^{T} \mathbf{r}_{t[i_t]}}_{\text{our total reward}}$$

Multi-armed bandits: Basic **stochastic** model

Often assumes each action has an underlying reward distribution, e.g.

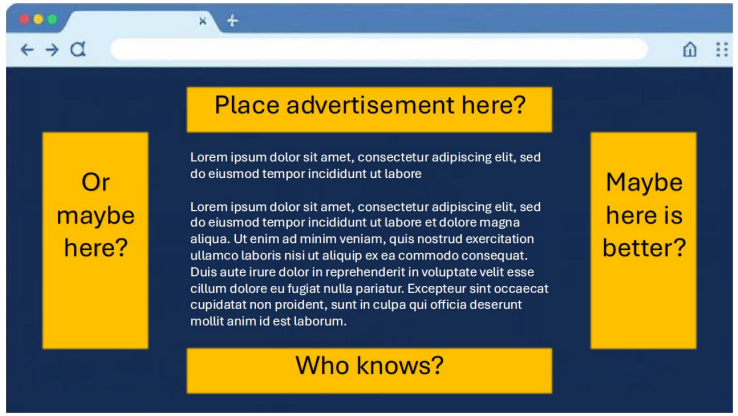
1. top: 0.06 probability of click ($\mathbb{E}\mathbf{r}_{t[\text{top}]} = 0.06$)
2. bottom: 0.02
3. left: 0.04
4. right: 0.07



Reduces goal to figuring out which arm has the best expected reward

Key concept: to maximize total reward over time we must trade-off exploration and exploitation

Exploration vs. exploitation



Depends on time horizon

- exploit if one step left
- exploring may produce better reward in the long run

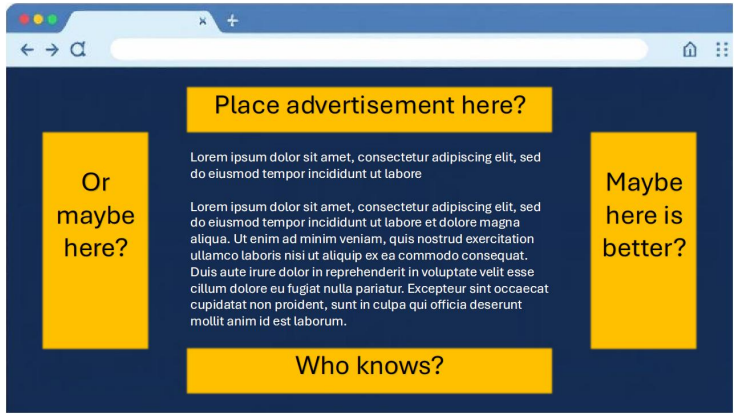
Depends on uncertainty/noise in the reward

- explore more if distribution has large variance

Simplest algorithm: ε-greedy

- choose best arm seen so far, *except*
- with probability ε, choose arm uniformly at random

Non-stationary problems



What if the reward distributions change over time?

- e.g. the distribution of webpage users changes

Then the basic model is **adversarial**:

- adversary designs sequence of rewards $\mathbf{r}_1, \ldots, \mathbf{r}_T$ to make the algorithm do poorly

Another model is **contextual**:

- algorithm has access to context $\mathbf{c}_t$ that informs the non-stationarity of the rewards $\mathbf{r}_1, \ldots, \mathbf{r}_T$
- e.g. user information

Multi-armed bandits: Algorithm design

Want algorithms that achieve **sublinear** regret:

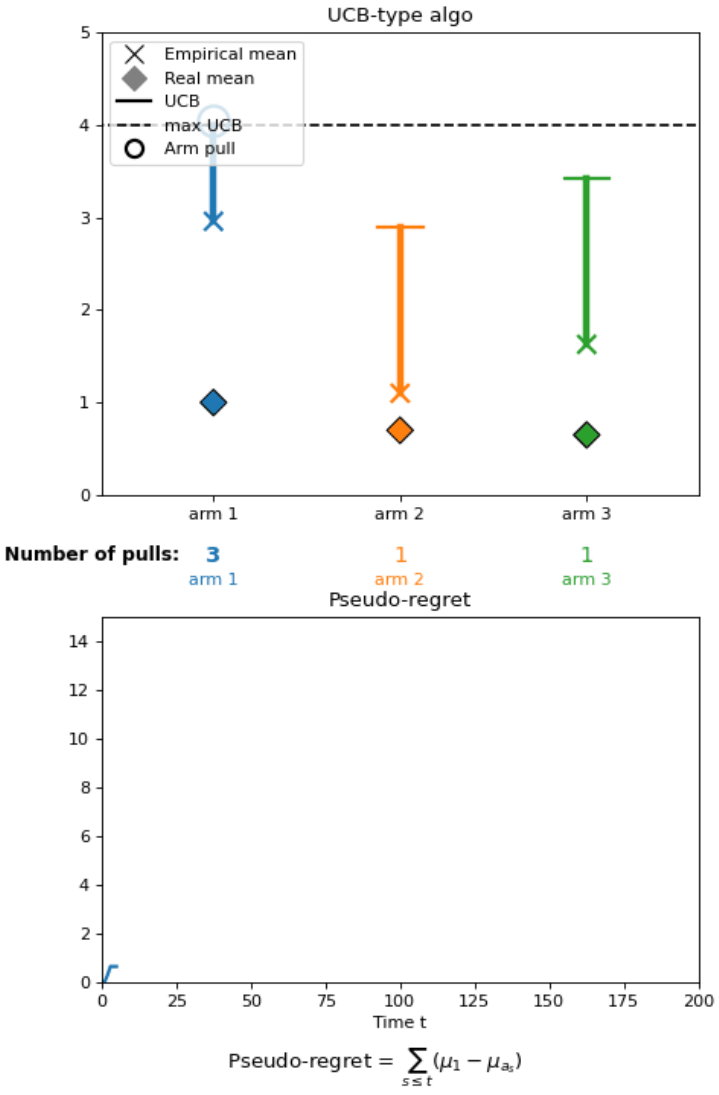
$$\text{Regret} \quad = \quad \max_i \sum_{t=1}^{T} \mathbf{r}_{t[i]} \quad - \quad \sum_{t=1}^{T} \mathbf{r}_{t[i_t]} \quad = \quad o(T)$$

- sublinear regret implies improvement in performance over time
- successful strategies balance exploration and exploitation

Stochastic bandit algorithm: UCB-1

[Auer-Cesa-Bianchi-Fischer, MLJ 2002]

- maintain estimate of an upper bound on the mean reward of each arm (similar to a confidence interval)
- at each round, optimistically pick the action with the highest upper bound
- $O(\log T)$ regret in the stochastic case ☺
- linear regret in adversarial case ☹



Wikipedia

Adversarial bandit algorithm: Exp3

[Auer-Cesa-Bianchi-Freund-Schapire, SIAM J. Computing 2002]

set a step-size $\eta > 0$

uniformly initialize probability distribution $\mathbf{p}$ over the d arms

for each round $t = 1, \dots, T$

select action i_t randomly according to $\mathbf{p}$

receive reward $\mathbf{r}_{t[i_t]}$

update $\mathbf{p}_{[i_t]} \leftarrow \mathbf{p}_{[i_t]} \cdot \exp\left(\eta \frac{\mathbf{r}_{t[i_t]}}{\mathbf{p}_{[i_t]}}\right)$

normalize $\mathbf{p} \leftarrow \mathbf{p}/\|\mathbf{p}\|_1$

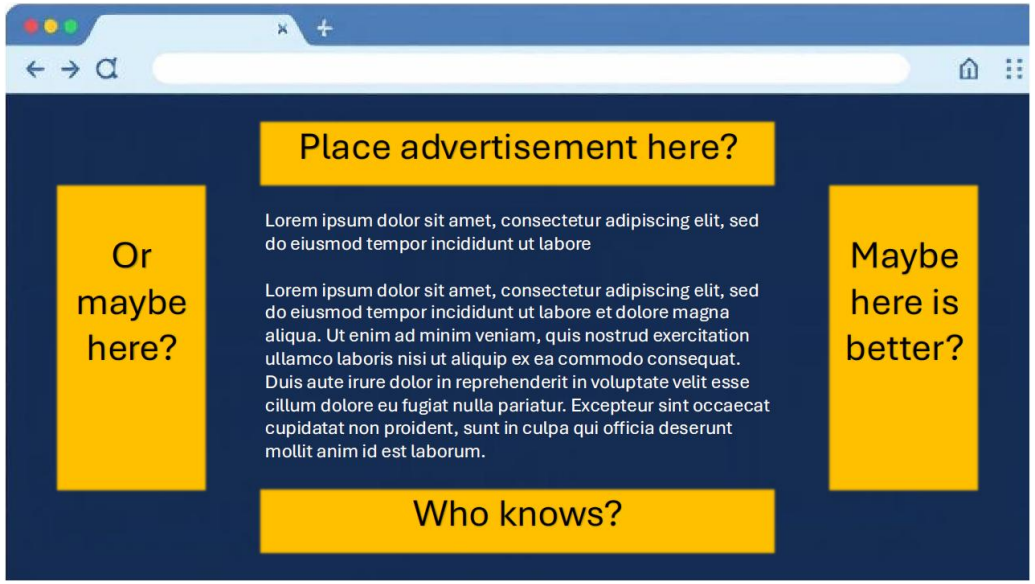
Exp3 ensures $O(\sqrt{dT \log d})$ regret ☺

Often viewed as a pessimistic algorithm ☹

Multi-armed bandits: Summary

Very popular area of research last decade:

- lots of algorithms developed
- made Google a lot of money
- **why do we care?**



Multi-armed bandits for numerical simulation

Example: ~~webpage ad placement~~ **linear solver selection**

Four options: ~~top, bottom, left, right~~ **preconditioner configurations**

Modeled as a multi-round game:

for $t = 1, \dots, T$ ~~(new visits to webpage)~~ **simulation timesteps**
  choose action $i_t \in \{1, \dots, d\}$ ~~(d = 4)~~ **(d=number of configuration options)**
  receive ~~reward~~ $\mathbf{cost}_t(\mathrm{i_t}) \in \mathbb{R}$ ~~(1 if user clicks on ad, else 0)~~ **linear solve time**

Goal: minimize the **cumulative regret**

$$\text{Regret} = \underbrace{\sum_{t=1}^{T} \text{cost}_t(i_t)}_{\text{total wallclock spent solving linear systems}} - \underbrace{\min_i \sum_{t=1}^{T} \text{cost}_t(i)}_{\text{total wallclock of best config}}$$

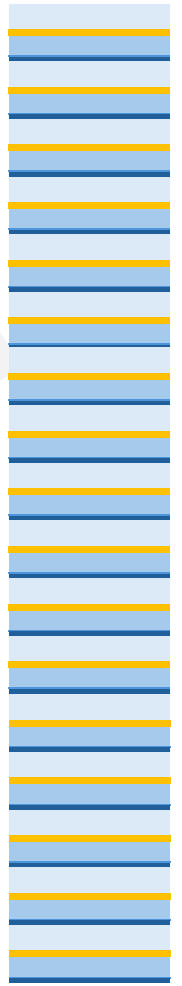
A learning approach: Maintaining a **distribution over solver configs**

At each timestep t :

- configure a solver, e.g. PCG
  - sample configuration $i_t \sim \mathbf{p}_t$ from a distribution $\mathbf{p}_t \in [0,1]^d$ over d different configurations
- solve $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$ and record $\text{cost}_t(i_t)$ in wallclock time

solving a linear system $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$

learning



timestep

$t = 1$

$t = 2$

$\vdots$

$t = T$

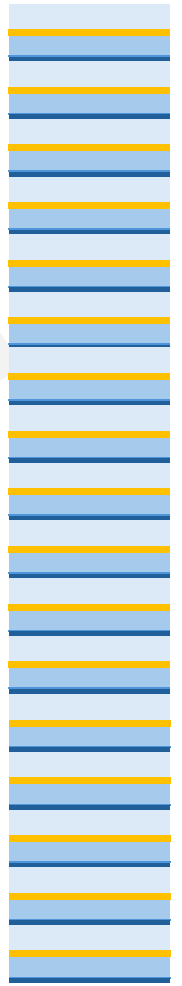
A learning approach: Maintaining a **distribution over solver configs**

At each timestep t :

- configure a solver, e.g. PCG
  - sample configuration $i_t \sim \mathbf{p}_t$ from a distribution $\mathbf{p}_t \in [0,1]^d$ over d different configurations
- solve $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$ and record $\text{cost}_t(i_t)$ in wallclock time
- use the cost to update the distribution to $\mathbf{p}_{t+1}$

solving a linear system $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$

learning



timestep

$t = 1$

$t = 2$

$\vdots$

$t = T$

Updating probabilities using a **bandit** method

At each timestep t :

- configure a solver, e.g. PCG
  - sample configuration $i_t \sim \mathbf{p}_t$ from a distribution $\mathbf{p}_t \in [0,1]^d$ over d different configurations
- solve $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$ and record $\text{cost}_t(i_t)$ in wallclock time
- use the cost to update the distribution to $\mathbf{p}_{t+1}$

Tsallis-INF

[Abernethy-Lee-Tewari, 2015; Zimmert-Seldin, 2022]

- construct an estimator $\mathbf{c}_t \in \mathbb{R}^d$ of the cost of **each** config $i \in [d]$
  - how: $\mathbf{c}_{t[i]} = \frac{\text{cost}_t(i_t)}{\mathbf{p}_{t[i_t]}}$ if $i = i_t$ else 0
  - why: $\mathbb{E}_{\mathbf{p}_t}[\mathbf{c}_{t[i]}] = \text{cost}_t(i)$
- set cumulative cost estimates

learning costs:
- $O(\#\text{configs})$ memory
- $O(\#\text{configs})$ compute

$$\mathbf{C}_t = \sum_{s=1}^{t} \mathbf{c}_s$$

- update

$$\mathbf{p}_{t+1} = \text{argmin}_{\mathbf{p}} \underbrace{\frac{\mathbf{C}_t^\top \mathbf{p}}{2\sqrt{t}}}_{\text{minimize expected cumulative cost ...}} - \underbrace{\sum_{i=1}^{d} \sqrt{\mathbf{p}_{[i]}}}_{\text{... regularized by the negative Tsallis entropy}}$$

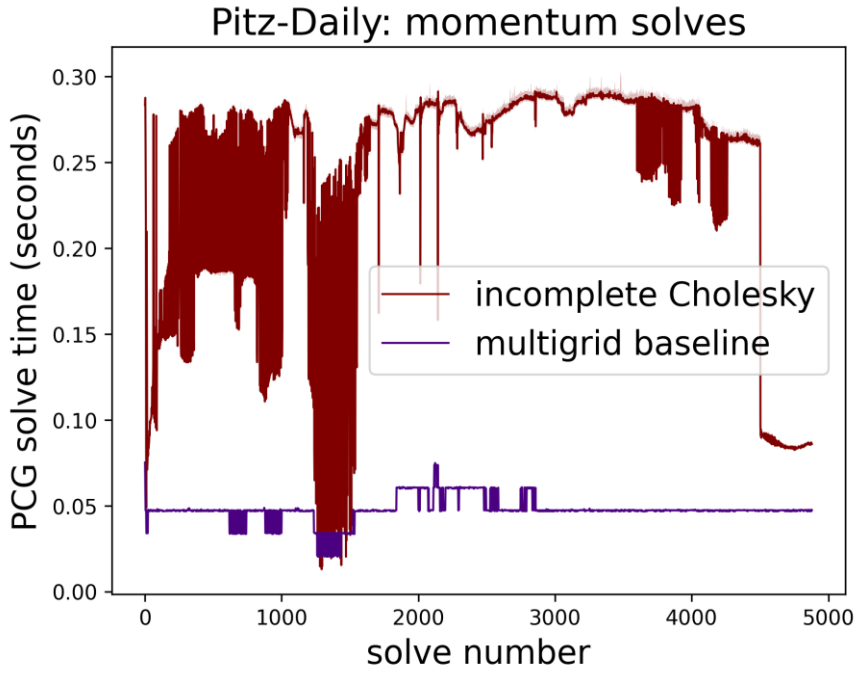
Aside: Why Tsallis-INF?

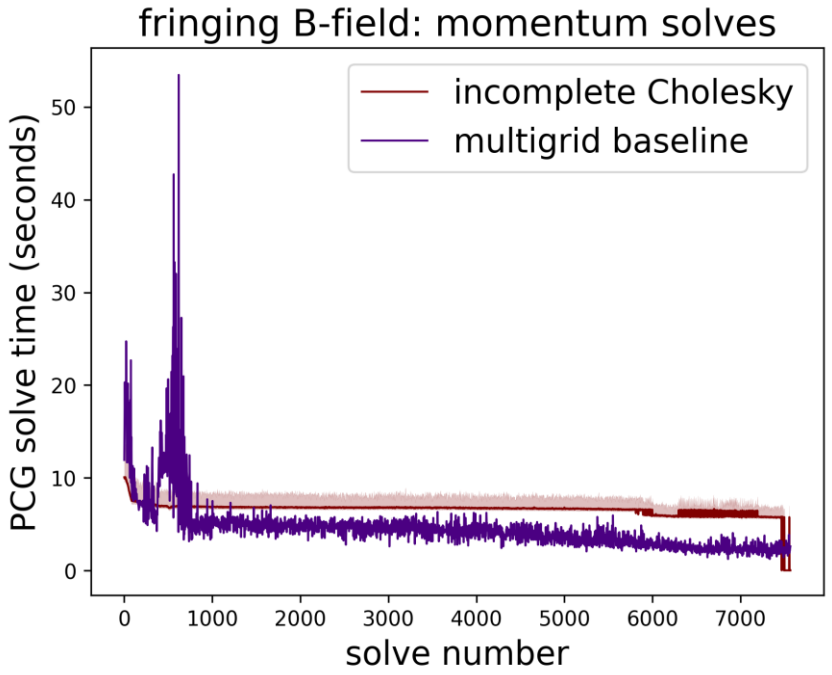
Unlike Exp3, it

- guarantees **optimal regret** $O(\sqrt{dT})$ in the adversarial setting
  [Abernethy-Lee-Tewari, 2015]

- also has **best-of-both-worlds** stochastic guarantees
  [Zimmert-Seldin, 2022]

- which setting are we in?

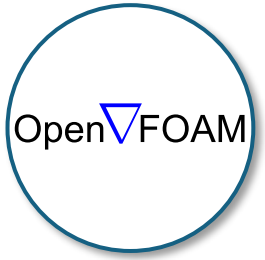




What might this look like in practice?

Integrating learning into PDE codes

[**K**-Chow-Jung-Wynne-Kolemen, 2025]



We develop a code for **OpenFOAM**, a leading OSS toolbox for solving transient fluid PDEs (among others)

PCGBandit: github.com/the-lens-project/PCGBandit

1. runs PCG with preconditioners sampled from a learned distribution
2. algorithm updates distribution using PCG runtime alone
3. no other packages and **no pre-training**: just replace the PCG solver specification in any OpenFOAM config with PCGBandit

Evaluation setup

[**K**-Chow-Jung-Wynne-Kolemen, 2025]

Four simulations:

- four OpenFOAM tutorials at 2x resolution
- two problems from the FreeMHD code
  - magnetohydrodynamic software developed at the Princeton Plasma Physics Lab: github.com/PlasmaControl/FreeMHD

33 preconditioners:

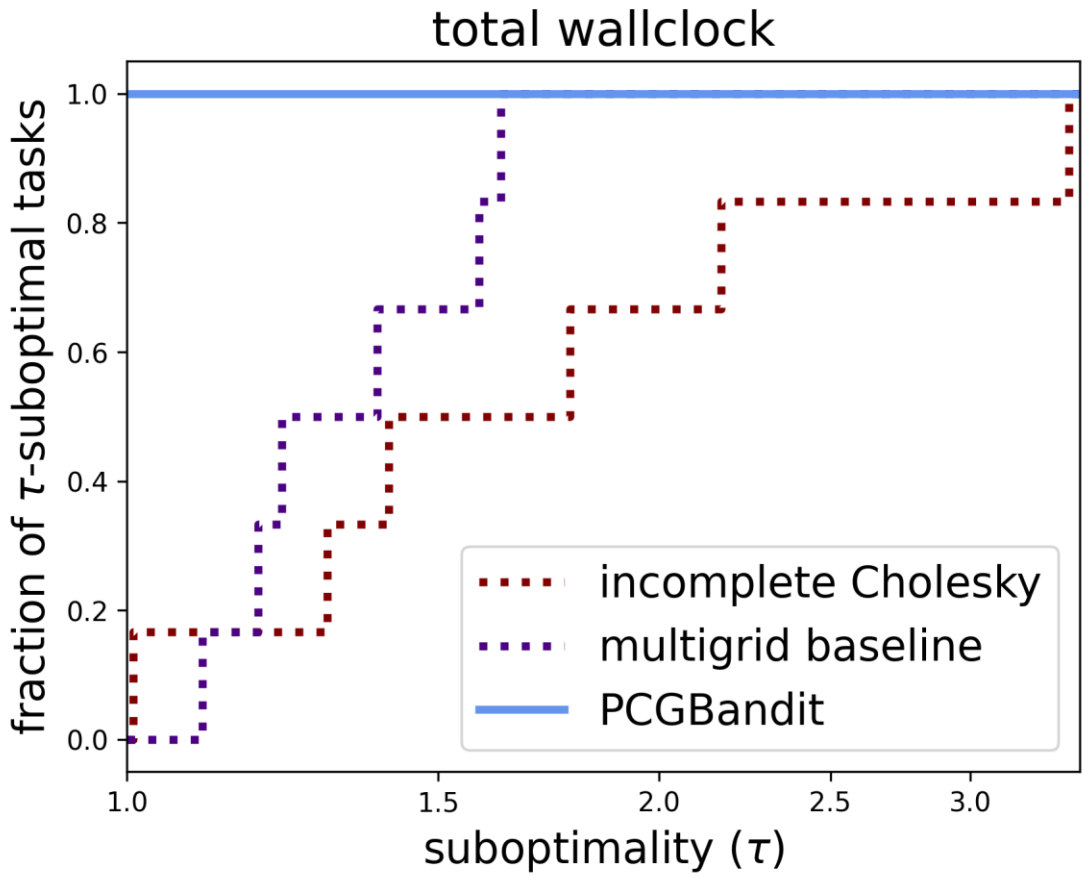
- IC(0)
- ICT with eight thresholds $(10^{-4}, \ldots, 10^{-0.5})$
- multigrid with 4 smoothers, 3 coarsest cell numbers, and 2 level-merging options

Performance profiles: How much better is PCGBandit?

[**K**-Chow-Jung-Wynne-Kolemen, 2025]

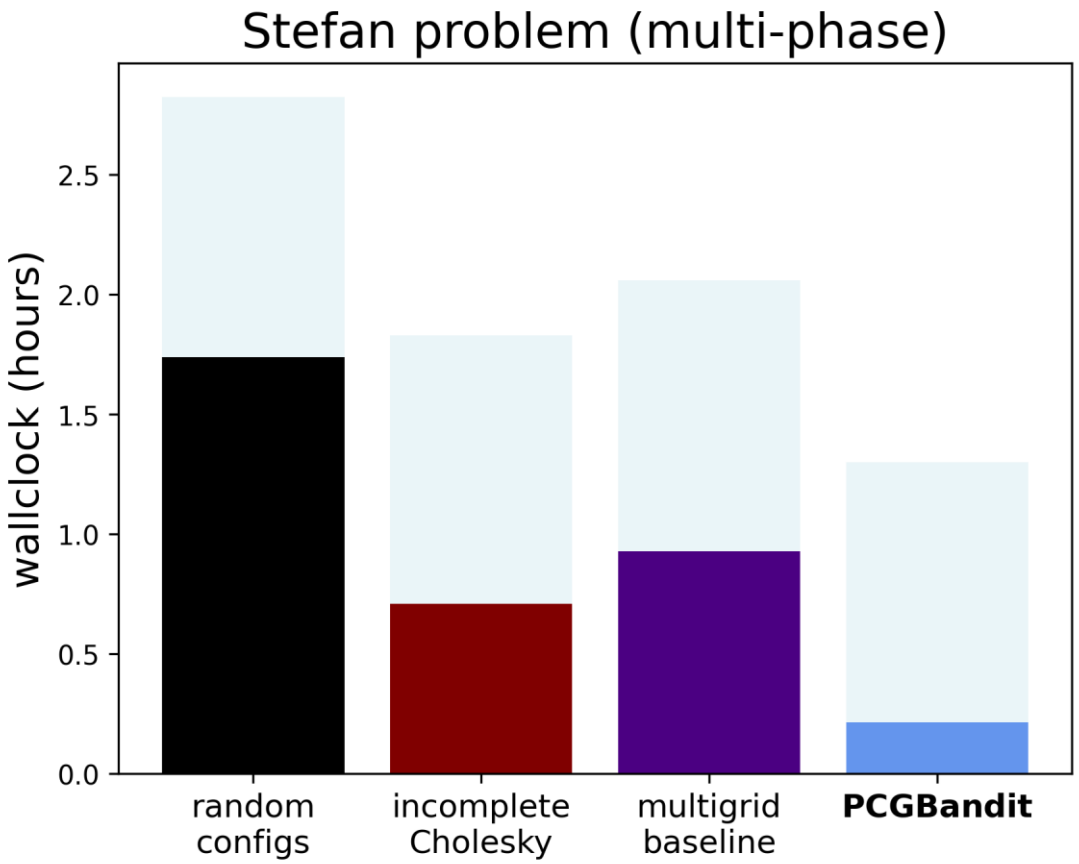
each curve plots the fraction of tasks on which a method is at most τ-suboptimal (in terms of wallclock)

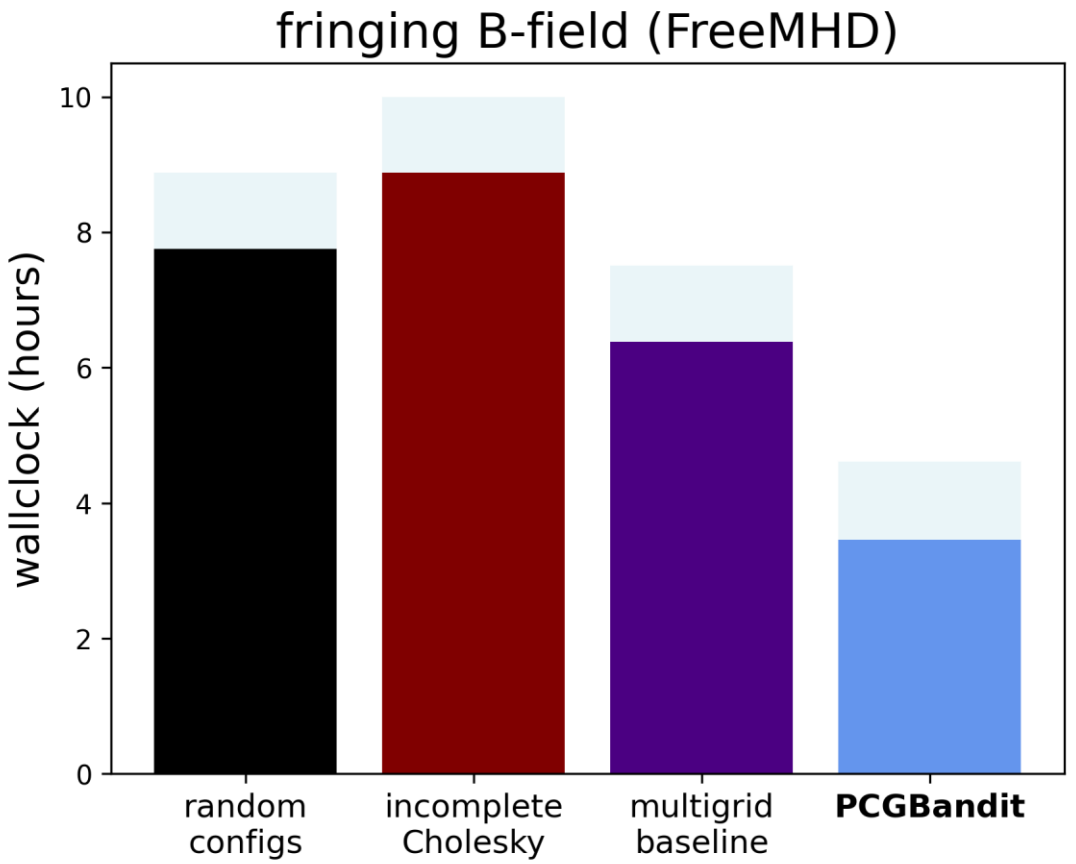
best static baseline (multigrid) can take more than 1.5x longer to run



Improvement on specific tasks: Phase change and magnetohydrodynamics

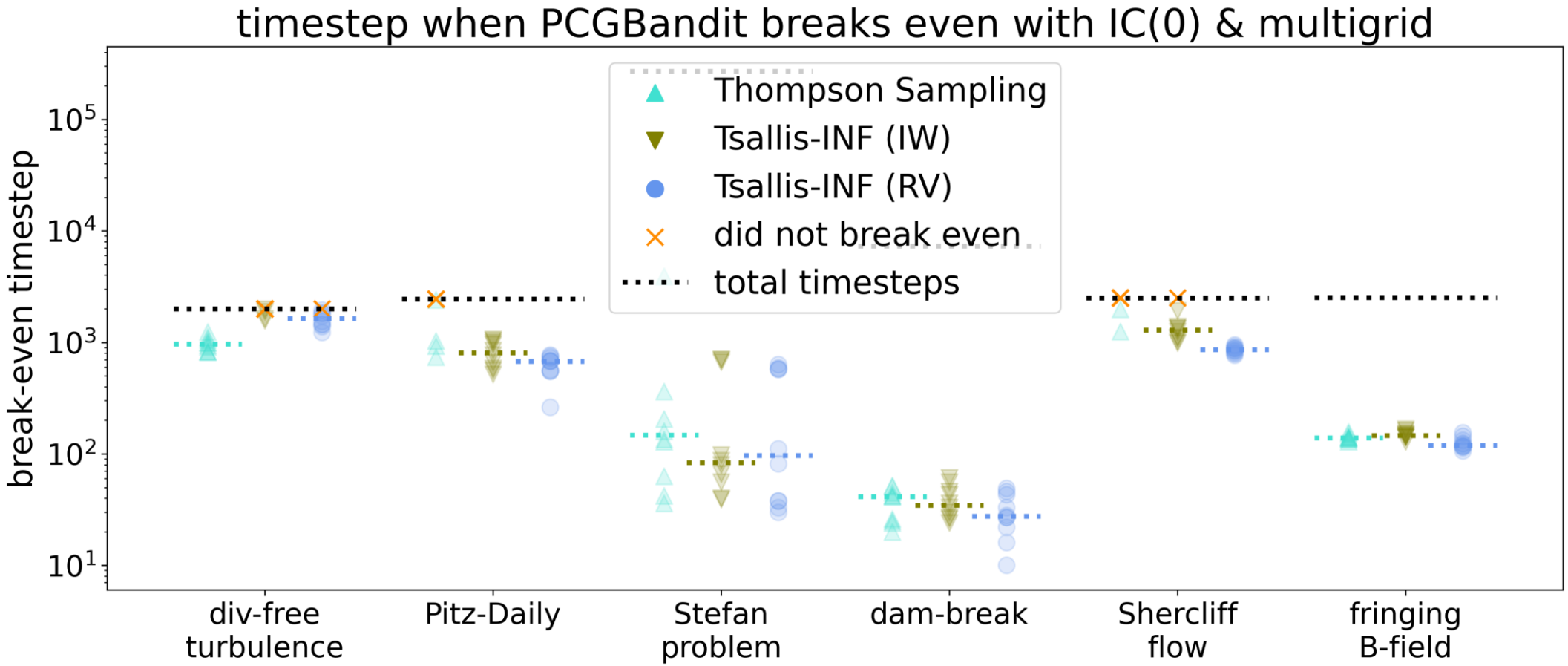
[**K**-Chow-Jung-Wynne-Kolemen, 2025]





Breakeven results:
How many timesteps is needed for utility?

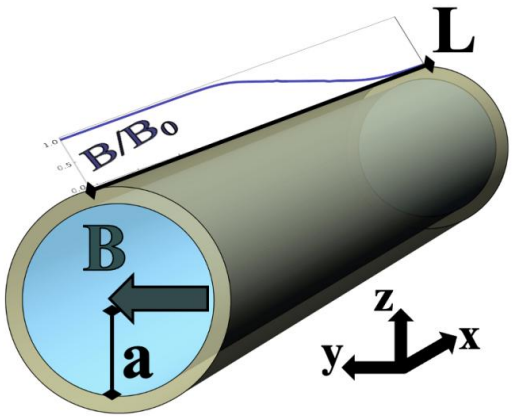
[**K**-Chow-Jung-Wynne-Kolemen, 2025]



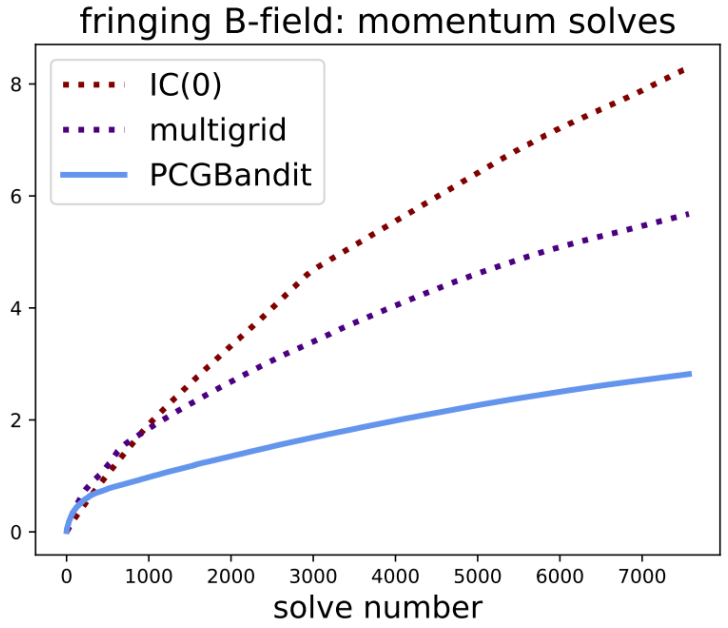
A glance at what PCGBandit learns

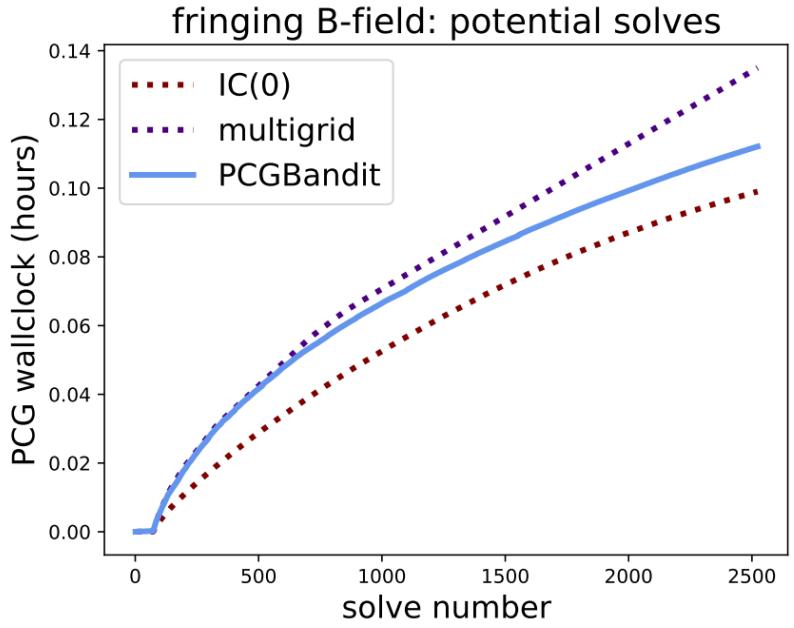
[**K**-Chow-Jung-Wynne-Kolemen, 2025]

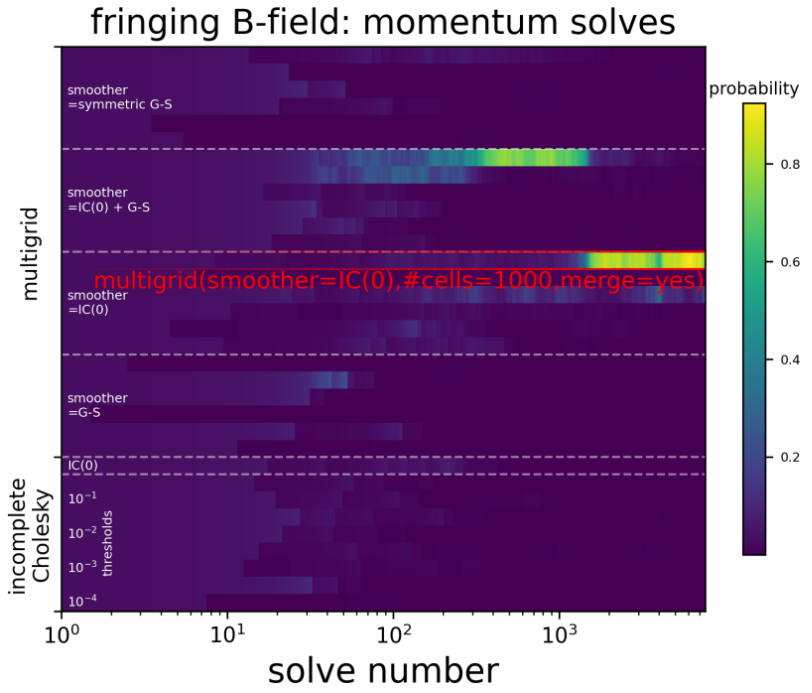
Setup: NaK flow through a dipole magnet in a closed pipe with conducting walls

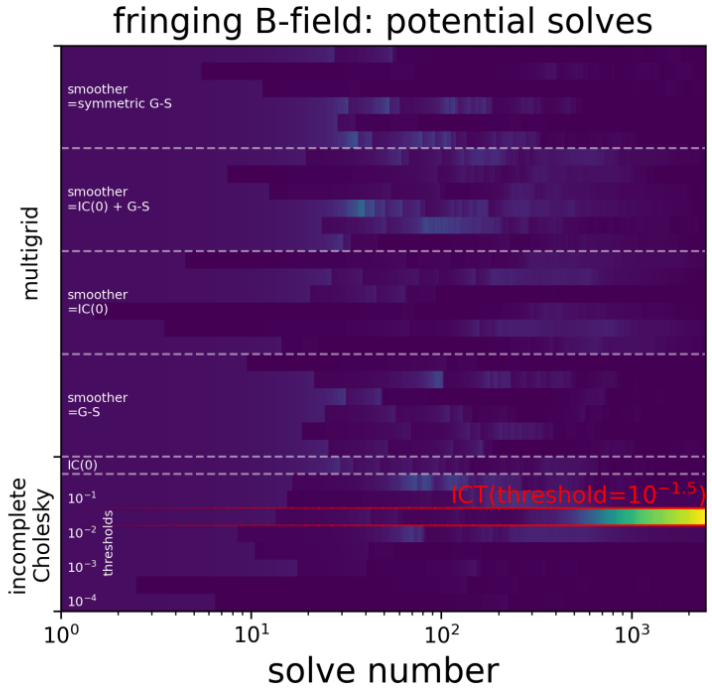


Have to solve multiple linear systems across simulation regions









Outline

1. **PCGBandit:** integrating online learning into numerical simulation

2. **Learning guarantees** in a model setting

3. **Open directions**

Why should we expect this to work? Analyzing a **model setting**

[**K**-Chow-Balcan-Talwalkar, ICLR 2024]

At each timestep t :

- configure a solver, e.g. ~~PCG~~
  - sample configuration $i_t \sim \mathbf{p}_t$ from a distribution $\mathbf{p}_t \in [0,1]^d$ over d different ~~configurations~~
- solve $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$ and record $\text{cost}_t(i_t)$ in ~~wallclock time~~
- use the cost to update the distribution to $\mathbf{p}_{t+1}$

successive over-relaxation (SOR)

relaxation parameters $\omega \in [1,2)$

number of iterations

Learning to relax: A model setting for learning-enhanced numerical simulation

[**K**-Chow-Balcan-Talwalkar, ICLR 2024]

Goal: show that using feedback from solving $(\mathbf{A}_1, \mathbf{b}_1), \ldots, (\mathbf{A}_t, \mathbf{b}_t)$ we can set a good SOR parameter at step $t+1$

More formally:

online learn a sequence of parameters $\omega_1, \ldots, \omega_T \in \Omega$

Approach:

1. online bandit optimization
2. a new **regularity result** about SOR's dependence on ω

cost of solving $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$ using parameter ω_t

$$\frac{1}{T}\sum_{t=1}^{T} \text{cost}_t(\omega_t) \leq o_T(1) + \min_{\omega} \frac{1}{T}\sum_{t=1}^{T} \text{cost}_t(\omega)$$

avg. cost we incur

goes to 0 as $T \to \infty$ (**Regret**/T)

avg. cost of the best fixed parameter ω

How does SOR depend on the relaxation parameter?

[**K**-Chow-Balcan-Talwalkar, ICLR 2024]

What is successive over-relaxation (**SOR**)?

- converges assuming $\mathbf{A}_1, \ldots, \mathbf{A}_T$ are SPD
- cost (#iterations) depends strongly on the **relaxation parameter** $\omega \in [1,2)$
- we use its symmetric (SSOR) variant

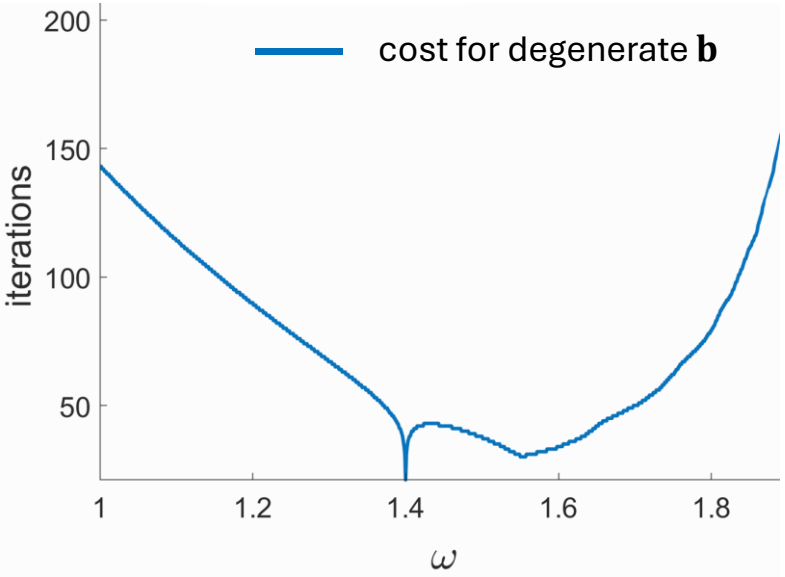
$\mathbf{D} + \mathbf{L} + \mathbf{L}^T \leftarrow \mathbf{A}$
$\mathbf{W}_\omega \leftarrow \mathbf{D}/\omega + \mathbf{L}$
$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$
for $k = 0, \ldots$ **do**
  if $\|\mathbf{r}_k\|_2 \leq \varepsilon \|\mathbf{r}_0\|_2$ **then**
    return k
  $\mathbf{x} = \mathbf{x} + \mathbf{W}_\omega^{-1} \mathbf{r}_k$
  $\mathbf{r}_{k+1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$

How does SOR depend on the relaxation parameter?

[**K**-Chow-Balcan-Talwalkar, ICLR 2024]

Recall: successive over-relaxation (**SOR**)

- converges assuming $\mathbf{A}_1, \ldots, \mathbf{A}_T$ are SPD
- cost (#iterations) depends strongly on the **relaxation parameter** $\omega \in [1,2)$
- we use its symmetric (SSOR) variant



Challenge: for degenerate $\mathbf{b}$, the cost can be a hard-to-optimize function of ω

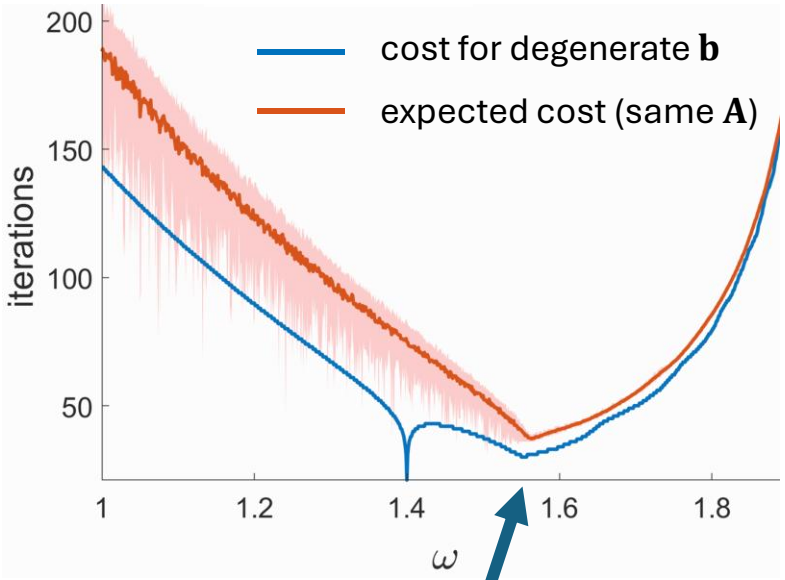
How does SOR depend on the relaxation parameter?

[**K**-Chow-Balcan-Talwalkar, ICLR 2024]

In practice: performance typically depends mostly on the spectrum of $\mathbf{A}$, *not* on properties of $\mathbf{b}$

Approach: model "typical" systems with a **distributional assumption** on $\mathbf{b_t}$ (no additional assumptions on $\mathbf{A_t}$)

Challenge: for degenerate $\mathbf{b}$, the cost can be a hard-to-optimize function of ω

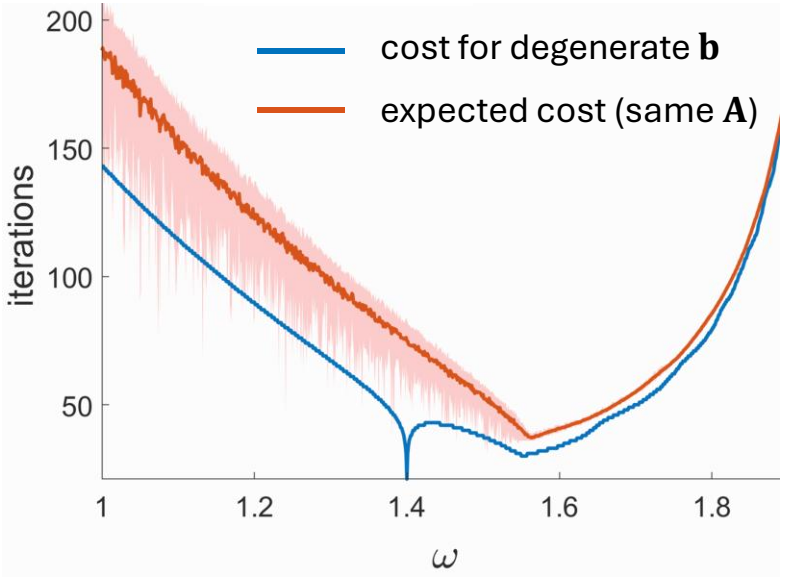


$$\omega^* = 1 + \left(\frac{\mu}{1+\sqrt{1-\mu^2}}\right)^2$$

$$\mu = \rho(\mathbf{I_n} - \mathbf{D^{-1}A})$$

How does SOR depend on the relaxation parameter?

[**K**-Chow-Balcan-Talwalkar, ICLR 2024]



In practice: performance typically depends mostly on the spectrum of $\mathbf{A}$, *not* on properties of $\mathbf{b}$

Approach: model "typical" systems with a **distributional assumption** on $\mathbf{b_t}$ (no additional assumptions on $\mathbf{A_t}$)

Lemma [KCBT'24]: $\mathbb{E}_{\mathbf{b}}[\text{cost}(\omega)]$ is $O\left(\sqrt{n}\log^4\frac{n}{\varepsilon}\right)$-Lipschitz-continuous in ω

- follows from Gaussian anti-concentration
- enables bandit algorithms like Tsallis-INF to match the optimal $\omega \in \Omega$ by maintaining a distribution over a **discretization of the domain** Ω

Learning to relax: A model setting for learning-enhanced numerical simulation

[**K**-Chow-Balcan-Talwalkar, 2024]

Theorem [**K**CBT'24]: For *any* sequence of SPD matrices $\mathbf{A}_1, \ldots, \mathbf{A}_T$ and i.i.d. vectors $\mathbf{b}_t$, Tsallis-INF sequentially sets $\omega_1, \ldots, \omega_T$ such that in-expectation

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T} \text{cost}_t(\omega_t)}_{\text{avg. cost incurred}} \leq \underbrace{O\left(\frac{\sqrt[6]{n}}{\sqrt[3]{T}} \log^2 \frac{n}{\varepsilon}\right)}_{\substack{\text{goes to zero as} \\ T \to \infty}} + \underbrace{\min_{\omega \in \Omega} \frac{1}{T}\sum_{t=1}^{T} \text{cost}_t(\omega)}_{\substack{\text{avg. cost of the best} \\ \text{fixed parameter } \omega}}$$

Can we do better than just a **static** config?

Attaining **instance-optimality** using extra simulation information

[**K**-Chow-Balcan-Talwalkar, 2024]

Suppose the linear systems have the structure $\mathbf{A}_t = \mathbf{A} + c_t \mathbf{I}_n$

- e.g. solving the heat equation with **time-varying diffusion**: $\partial_t u = \alpha(t)\left(\partial_{xx} u + \partial_{yy} u\right)$

Then we can achieve the performance of the **optimal sequence** of SOR configurations using **contextual** bandit algorithms:

Theorem [**K**CBT'24]: if $\mathbf{A}_t = \mathbf{A} + c_t \mathbf{I}_n$ then there exists an algorithm for sequentially setting $\omega_1, \ldots, \omega_T$ such that in-expectation

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T} \text{cost}_t(\omega_t)}_{\text{avg. cost we incur}} \quad \leq \quad o_T(1) \quad + \quad \underbrace{\frac{1}{T}\sum_{t=1}^{T} \text{cost}_t(\omega_t^*)}_{\text{avg. cost of always using the instance-optimal configuration } \omega_t^*}$$

Outline

1. **PCGBandit:** integrating online learning into numerical simulation

2. **Learning guarantees** in a model setting

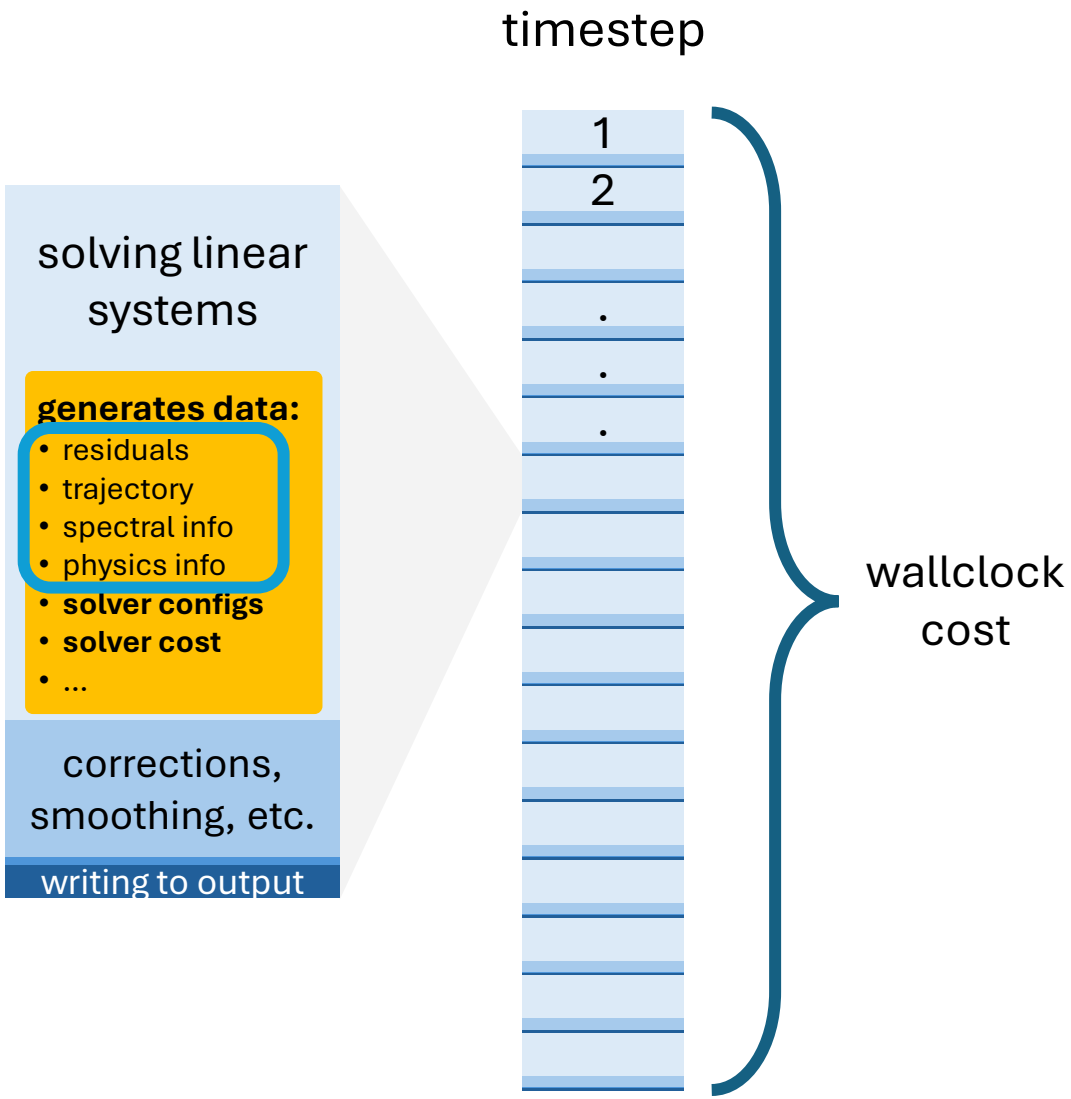
3. **Open directions**

Open directions for **learning-augmented simulation**

- PCGBandit is using minimal information (solve time) to learn a minimal model class (1 of k solver configs)
- in practice we have
  - much more information
  - many more interesting things to learn
- we also have numerous simulation codes to get data
  - OpenFOAM tutorials
  - other OSS code
  - Github
  - LLMs that can assemble all of this

Methodological directions

- opportunities:
  - preconditioners are matrices and so can be **parameterized much more richly** than via a few configs
  - besides solver costs, we have significant other info about individual systems
- challenges:
  - the objective is non-differentiable; we need **proxy losses** to do full information online learning
  - high dimensional parameterizations require more data to learn; perhaps we can **pretrain offline** ?



Summary

Learning can be done:

- instead of simulating
- before simulating
- **while simulating**

Thank you!

pages.cs.wisc.edu/~khodak
khodak@wisc.edu

Demonstrated computational usefulness of the latter in a widely-used OSS

- **K**-Jung-Wynne-Chow-Kolemen. *PCGBandit: One-shot acceleration of transient PDE solvers via online-learned preconditioners*. 2025.
- https://github.com/the-lens-project/PCGBandit

Learning-theoretic correctness in a model setting:

- **K**-Chow-Balcan-Talwalkar. *Learning to relax: Setting solver parameters across a sequence of linear system instances*. ICLR 2024.

Open directions:

1. making use of richer feedback, complex preconditioner parameterizations, multi-simulation pretraining
2. augmenting other simulation software in new applications