Predicting & Optimizing ML Training

Simons “Future of LLMs” Workshop

April 3, 2025

Collaborators

Logan Engstrom

Ben Chen

Axel Feldmann

Billy Moses

Aleksander Mądry

Collaborators

Logan Engstrom

Ben Chen

Axel Feldmann

Billy Moses

Aleksander Mądry

Graduating now!

Collaborators

Graduating now (from *undergrad*)!

Logan Engstrom

Ben Chen

Axel Feldmann

Billy Moses

Aleksander Mądry

Graduating now!

Collaborators

Graduating now (from *undergrad*)!

Logan Engstrom

Ben Chen

Axel Feldmann

Billy Moses

Aleksander Mądry

Graduating now!

Optimizing ML Training with Metagradient Descent

Logan Engstrom*[1], Andrew Ilyas*[2†], Benjamin Chen*[1],
Axel Feldmann[1], William Moses[3], Aleksander Mądry[1]

*Equal contribution [1]MIT, [2]Stanford, [3]UIUC

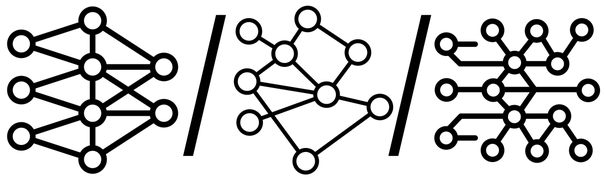
Motivation

Training requires many design choices. Making the right choice is hard!

Motivation

Training requires many design choices. Making the right choice is hard!

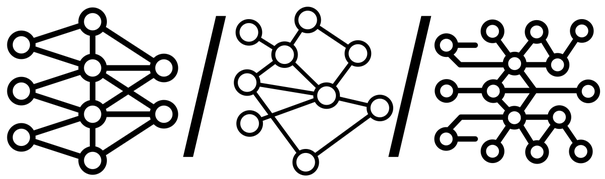
Model architecture



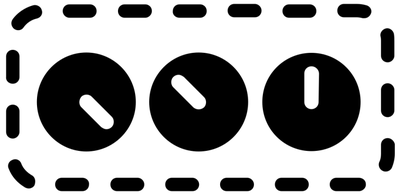
Motivation

Training requires many design choices. Making the right choice is hard!

Model architecture



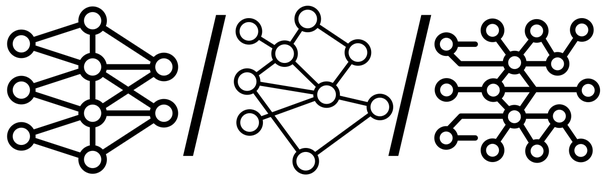
Hyperparameters



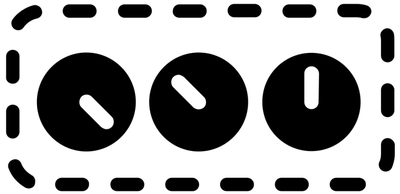
Motivation

Training requires many design choices. Making the right choice is hard!

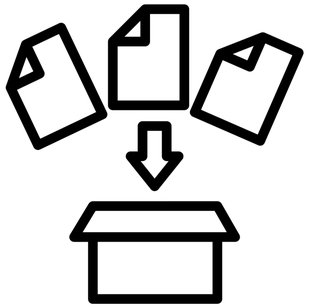
Model architecture



Hyperparameters



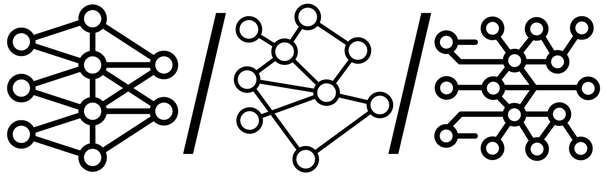
Training dataset



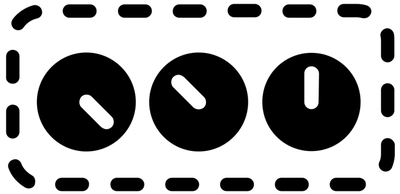
Motivation

Training requires many design choices. Making the right choice is hard!

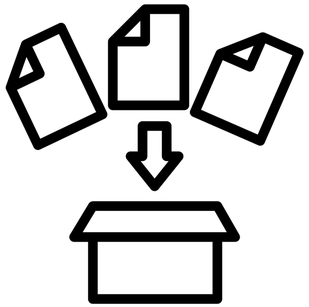
Model architecture



Hyperparameters



Training dataset



Fundamental Q: Can we predict and optimize the effect of these choices?

E.g.: Architecture search, scaling laws, data curation

The dream

A framework for optimizing and predicting the effect of design choices on model training

Example: Training data

Optimizing: Training data curation

(Sorscher et al. 2022; Gadre et al. 2023; Xie et al. 2024; Xia et al. 2024; Engstrom et al. 2024; many others)

Optimizing: Training data curation

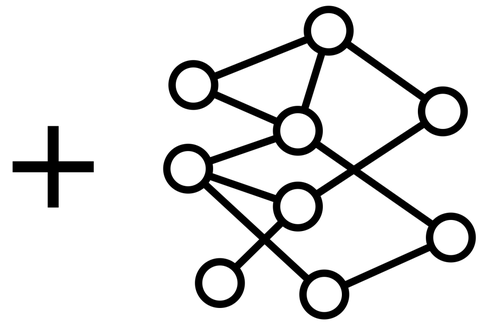
(Sorscher et al. 2022; Gadre et al. 2023; Xie et al. 2024; Xia et al. 2024; Engstrom et al. 2024; many others)

Scraped internet data

Optimizing: Training data curation

(Sorscher et al. 2022; Gadre et al. 2023; Xie et al. 2024; Xia et al. 2024; Engstrom et al. 2024; many others)

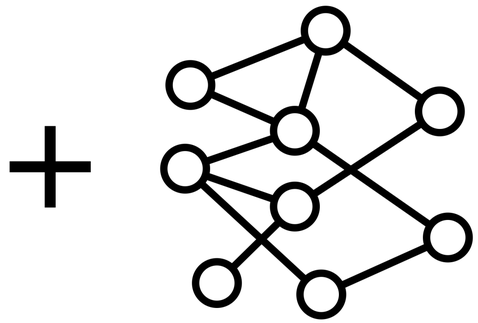
Scraped internet data



Optimizing: Training data curation

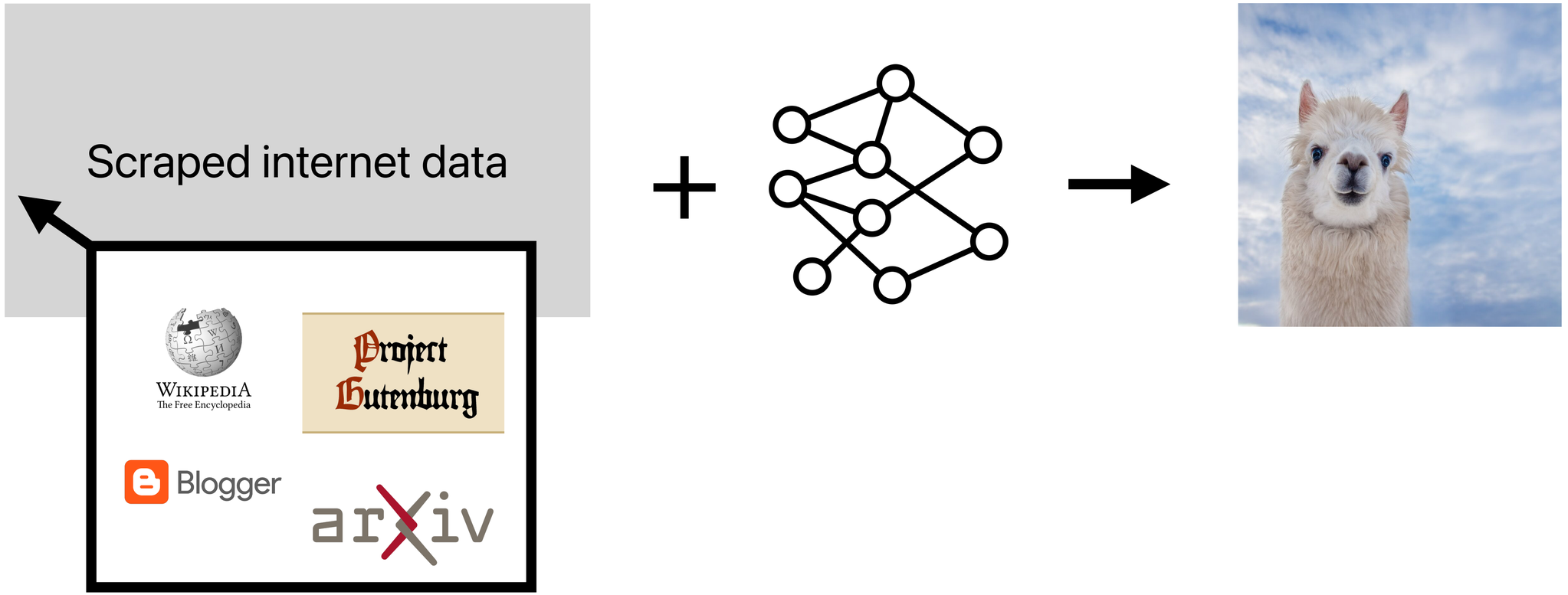
(Sorscher et al. 2022; Gadre et al. 2023; Xie et al. 2024; Xia et al. 2024; Engstrom et al. 2024; many others)

Scraped internet data



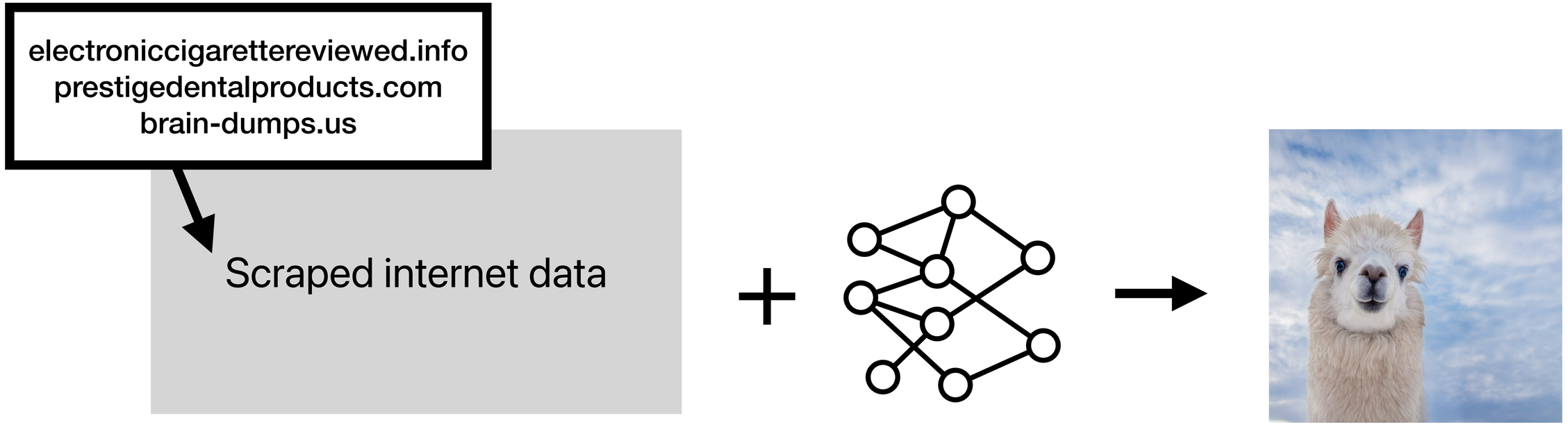
Optimizing: Training data curation

(Sorscher et al. 2022; Gadre et al. 2023; Xie et al. 2024; Xia et al. 2024; Engstrom et al. 2024; many others)



Optimizing: Training data curation

(Sorscher et al. 2022; Gadre et al. 2023; Xie et al. 2024; Xia et al. 2024; Engstrom et al. 2024; many others)



Optimizing: Training

(Sorscher et al. 2022; Gadre et al. 2023; Xie et ... ny others)

http://ufdc.ufl.edu/AA00010883/00095

ac omplishmnl 'o the mi alon and welR I the men. U (SR 600-17b-I) are clear enough and cover everybody. iM-be UeIrU d at home first. lDretewr.. NatM WLWr Of Panama. B Aleol B -w s sin,ambas to V '). .. &. .... iZ'." . '- .- '.'" ""5- ". ^ -*," ..^ -^:? ^~-. .... ; ,'- i ,? '. " ". 1 the eye to those Europe. like an orlontal: Mrt of the Ch-. te a ter ii wre n capw. to '. %e anti-trust sutt "I made let of money, but . mg C1 Y ay With ne shot of a Lot d Aieles mining and. petrlce Wymore is fac9 aur rtl 1 Dalton. Gr at gag tUdS 't come! Mand Jobaim AIMauccessful- M ." y uw? ie House ar&rkd, "Hold A mtllon yeas? A trillion? t Arrival." Or are they ageless?

electroniccigarettereviewed.info
prestigedentalproducts.com
brain-dumps.us

Scraped internet data

Optimizing: Training

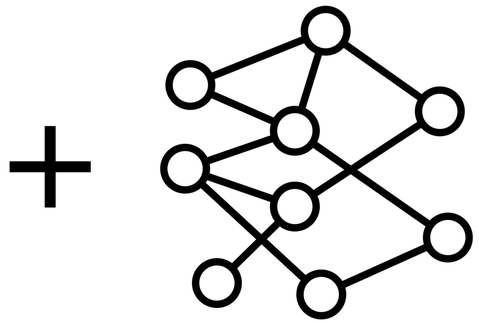
(Sorscher et al. 2022; Gadre et al. 2023; Xie et ... ny others)

http://ufdc.ufl.edu/AA00010883/00095

ac omplishmnl 'o the mi alon and welR I the men. U (SR 600-17b-I) are clear enough and cover everybody. iM-be UeIrU d at home first. lDretewr.. NatM WLWr Of Panama. B Aleol B -w s sin,ambas to V '). .. &. .... iZ'." . '- .- '.'" ""5- ". ^ -*," ..^ -^:? ^~-. .... ; ,'- i ,? '. " ". 1 the eye to those Europe. like an orlontal: Mrt of the Ch-. te a ter ii wre n capw. to '. %e anti-trust sutt "I made let of money, but . mg C1 Y ay With ne shot of a Lot d Aieles mining and. petrlce Wymore is fac9 aur rtl 1 Dalton. Gr at gag tUdS 't come! Mand Jobaim AIMauccessful- M ." y uw? ie House ar&rkd, "Hold A mtllon yeas? A trillion? t Arrival." Or are they ageless?

electroniccigarettereviewed.info
prestigedentalproducts.com
brain-dumps.us

Scraped internet data



Much of the internet is low-quality: what data should we train on?

Equivalently: can we **optimize** the training data selection for target performance?

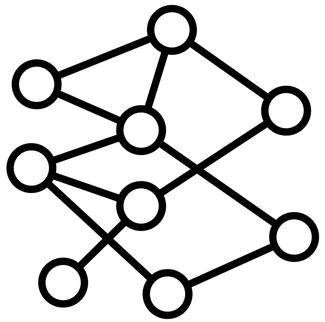
Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)

Predicting: Training data attribution

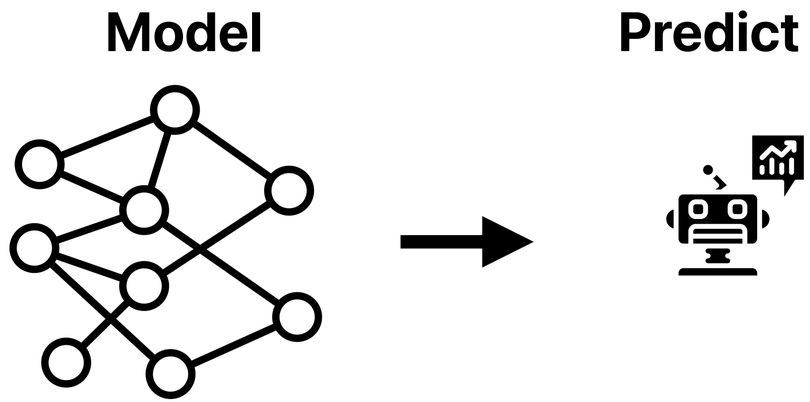
(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)

Model



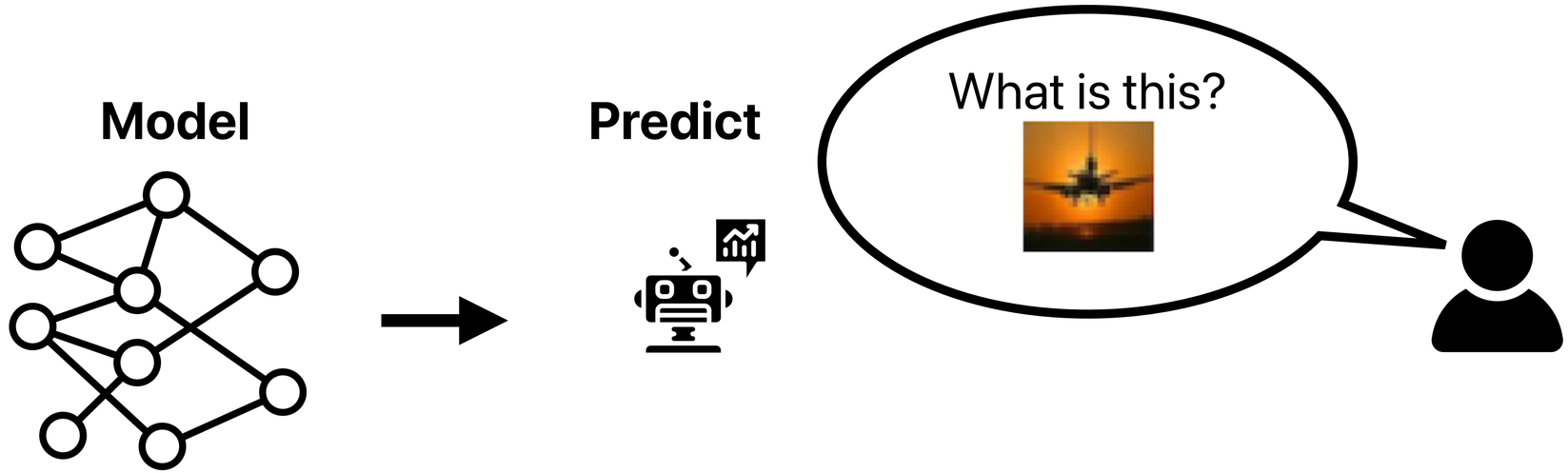
Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



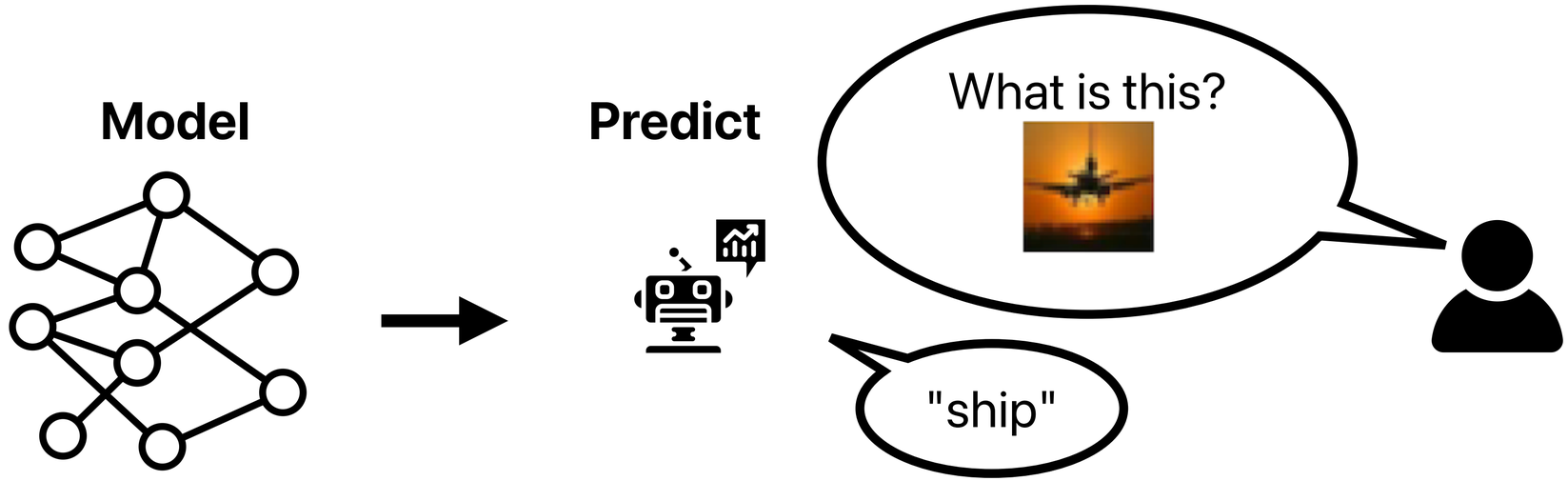
Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



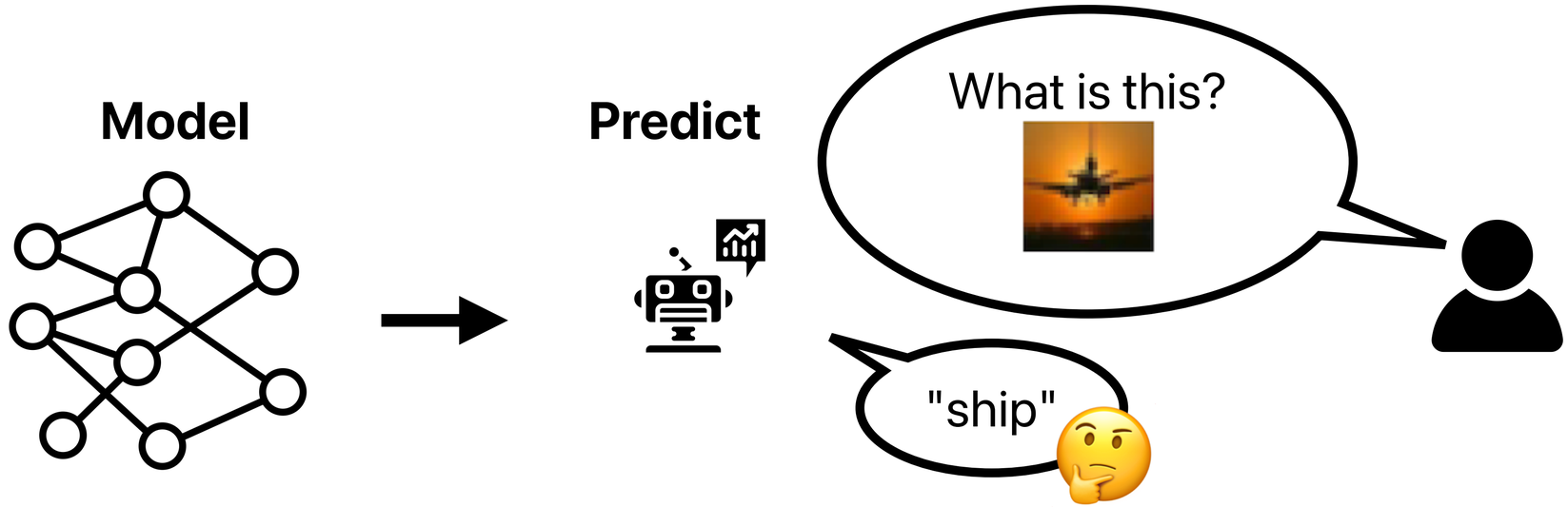
Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



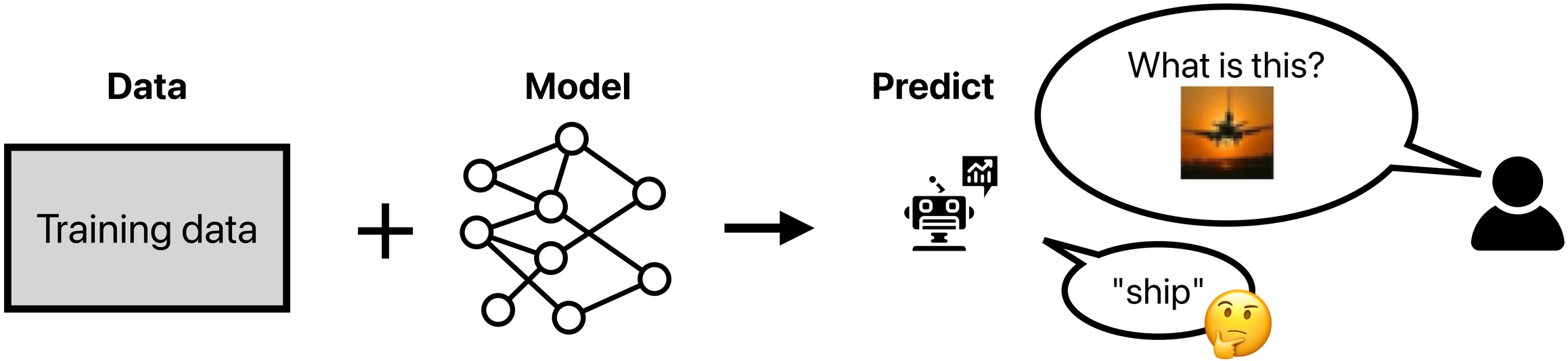
Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



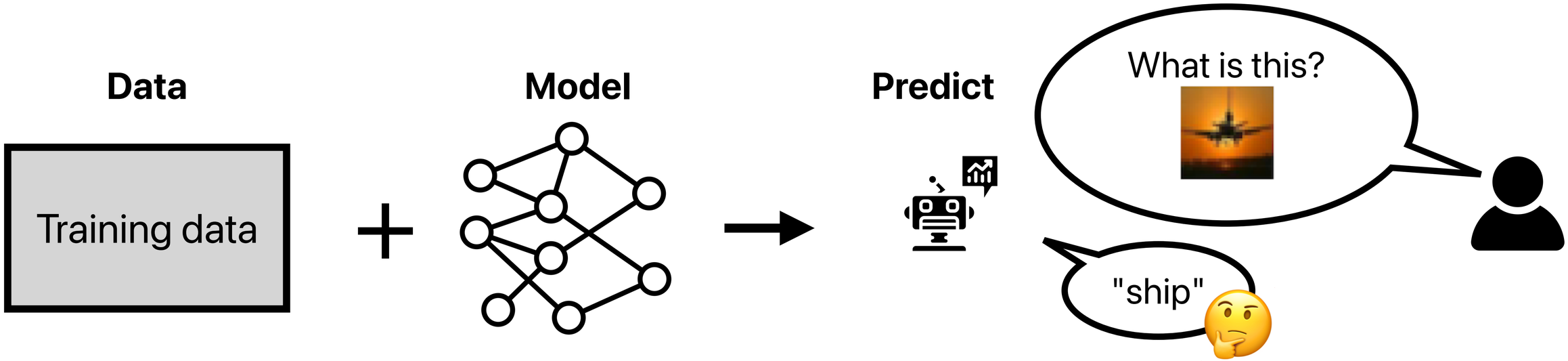
Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



Predicting: Training data attribution

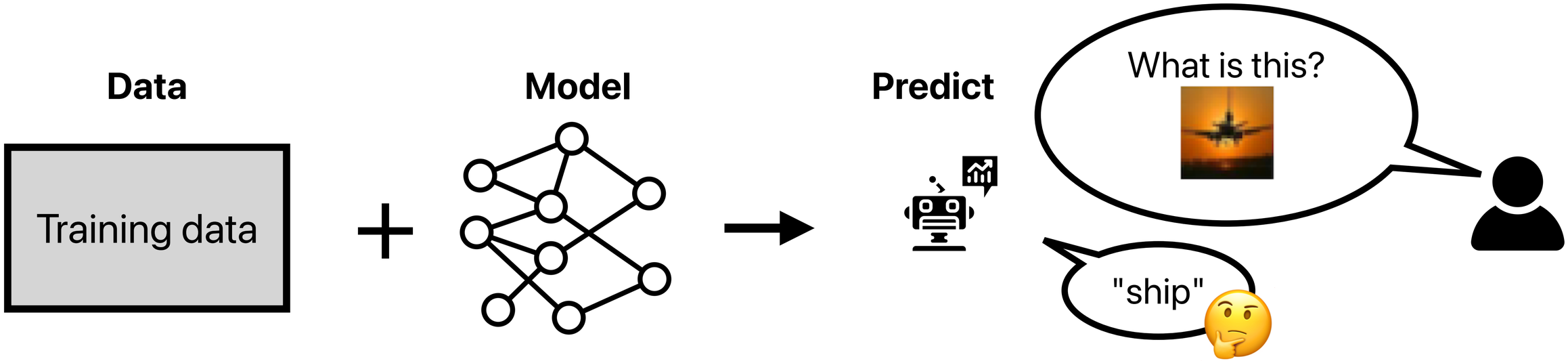
(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



"What examples in the training data caused this behavior?"

Predicting: Training data attribution

(Koh & Liang 2017; Feldman 2019; I et al. 2022; Bae et al. 2023, Park et al. 2024; many others)



"What examples in the training data caused this behavior?"

Equivalently: Can we **predict** change in this behavior as a function of training data?

Full problem statement

Problem: Metaparameter selection

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)

Metaparameter

$$\mathbf{z} \in \mathbf{Z} \subset \mathbb{R}^d$$

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)

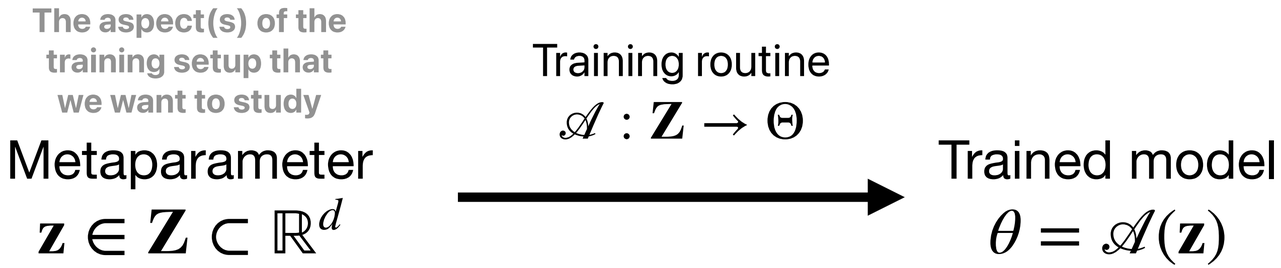
The aspect(s) of the training setup that we want to study

Metaparameter

$$\mathbf{z} \in \mathbf{Z} \subset \mathbb{R}^d$$

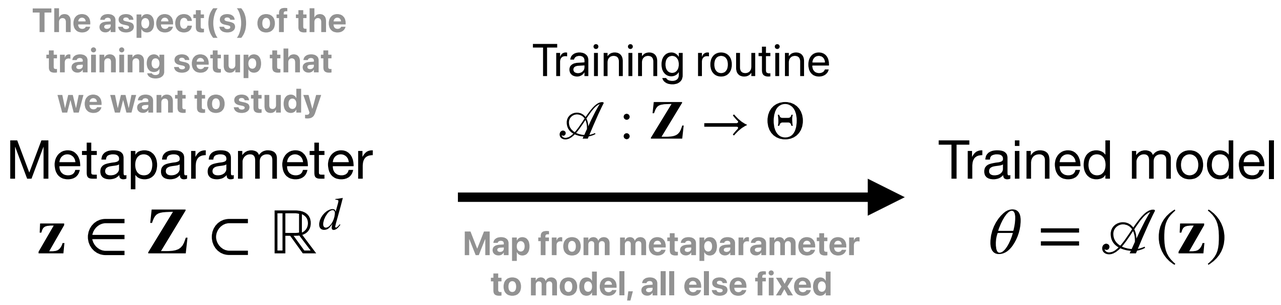
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



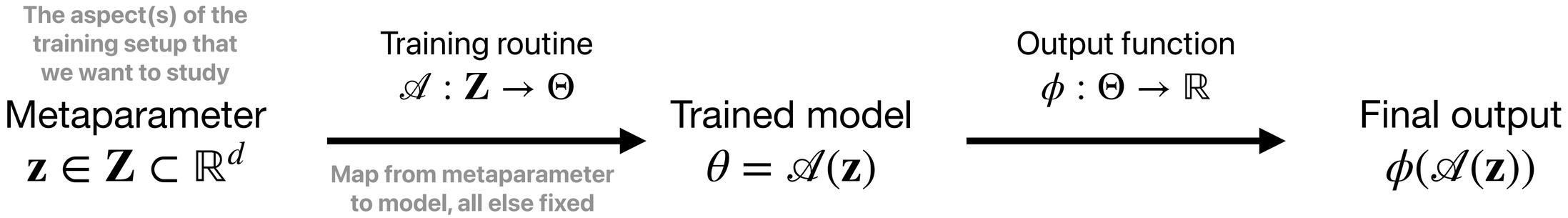
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



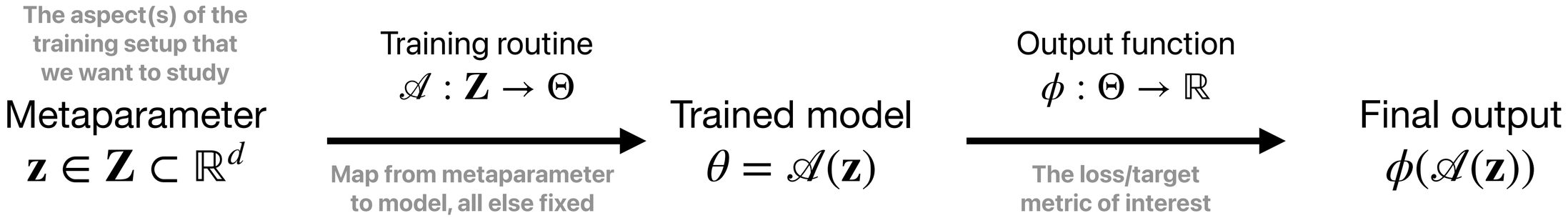
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



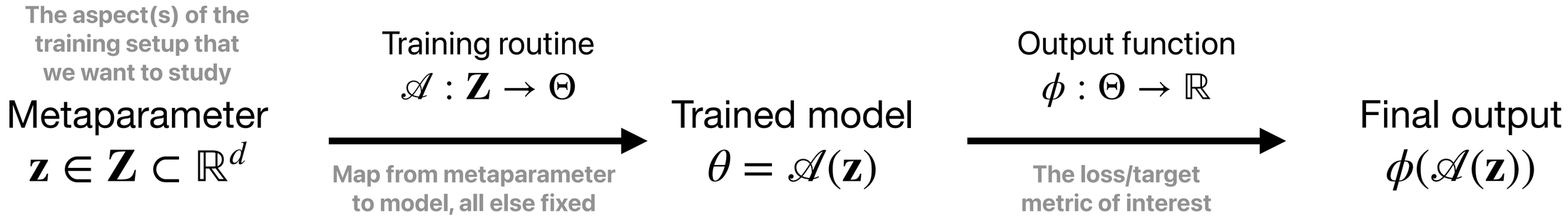
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



Problem: Metaparameter selection

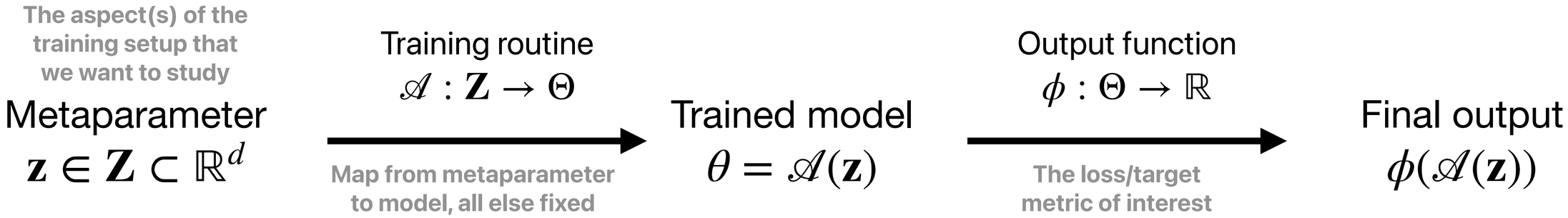
Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



Goals:

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)

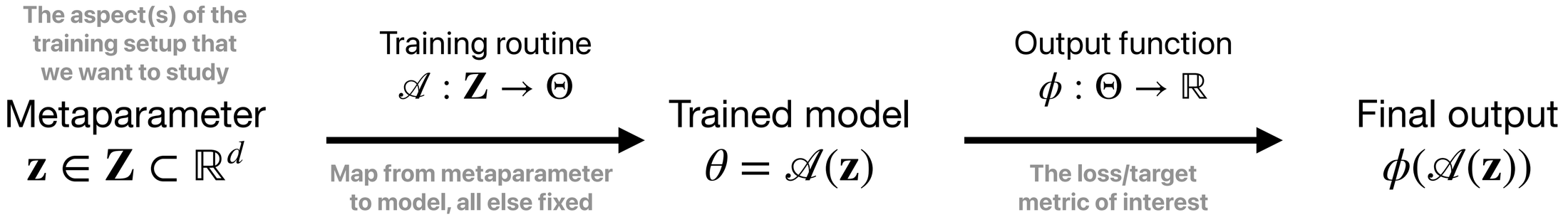


Goals:

Optimization: Find metaparameter $\mathbf{z}^* = \arg\min_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$ minimizing output

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



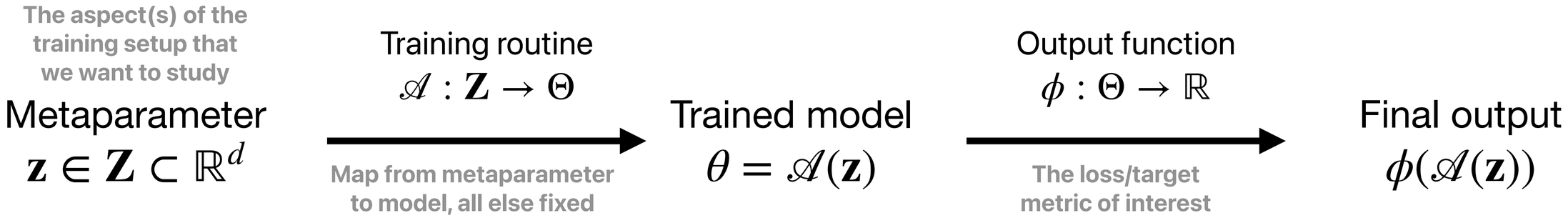
Goals:

Optimization: Find metaparameter $\mathbf{z}^* = \arg\min_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$ minimizing output

Prediction: Find $\hat{f}(\mathbf{z}) \approx \phi(\mathscr{A}(\mathbf{z}))$, direct model output estimator (does not invoke $\mathscr{A}$)

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



Goals:

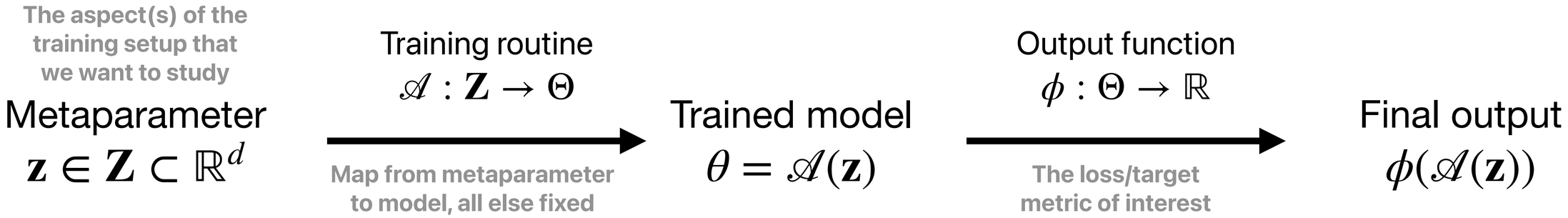
Optimization: Find metaparameter $\mathbf{z}^* = \arg\min_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$ minimizing output

Prediction: Find $\hat{f}(\mathbf{z}) \approx \phi(\mathscr{A}(\mathbf{z}))$, direct model output estimator (does not invoke $\mathscr{A}$)

Captures data curation & attribution but also <u>many</u> established ML challenges

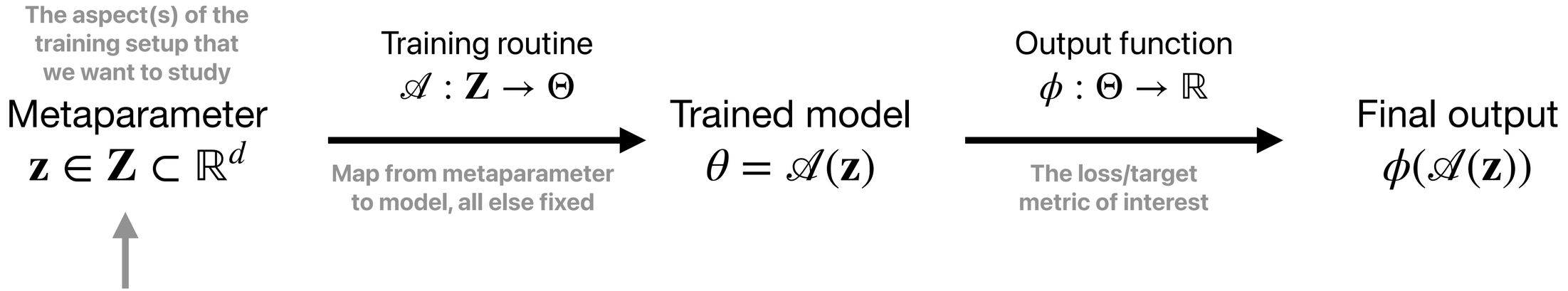
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



This will be our data selection:

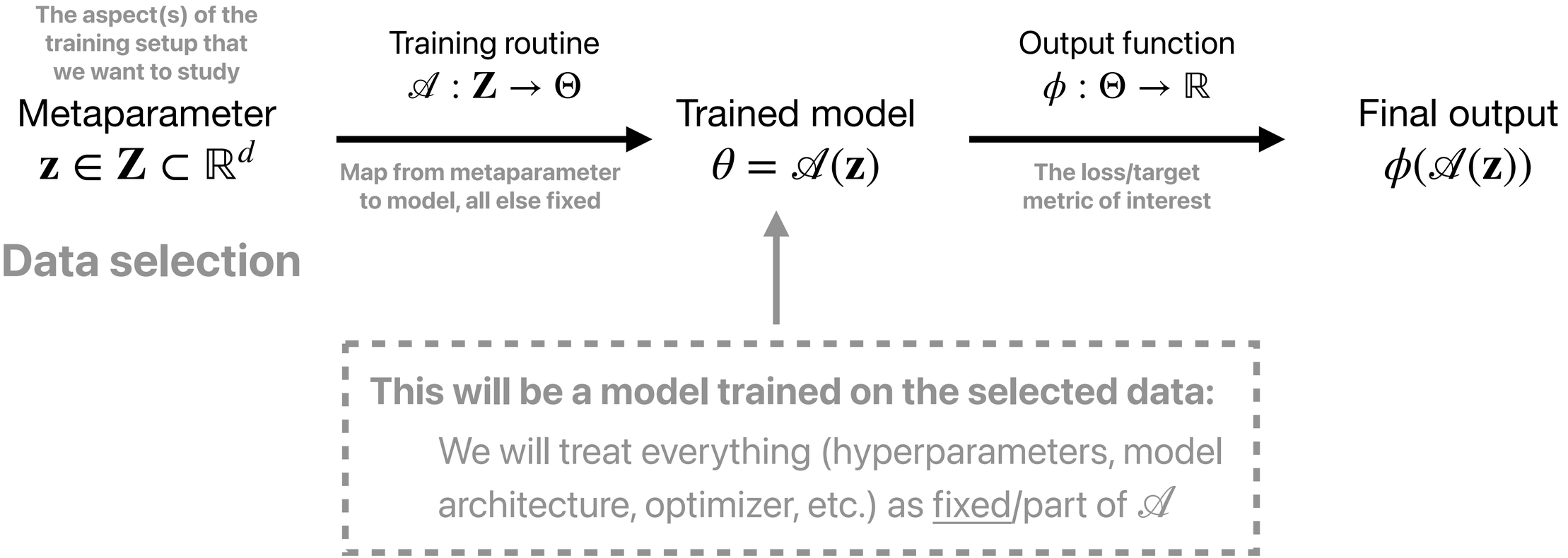
d = number of training examples

$\mathbf{z}_i = 0$ means don’t train on i

$\mathbf{z}_i > 0$ means train on i with importance weight $\mathbf{z}_i$

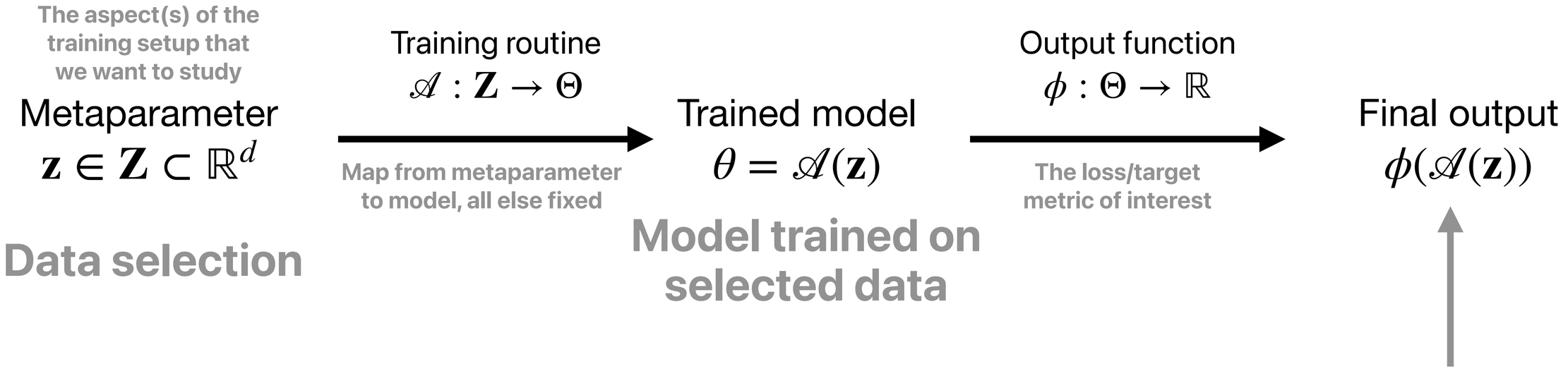
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



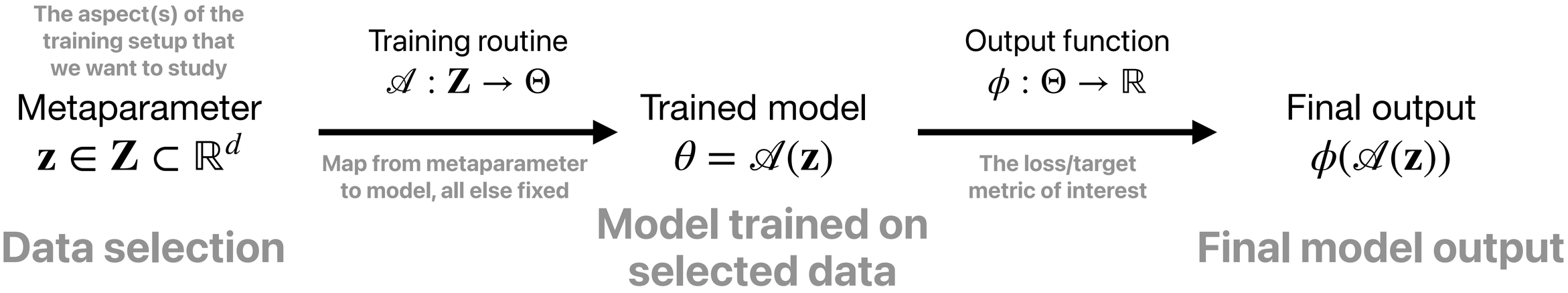
Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)

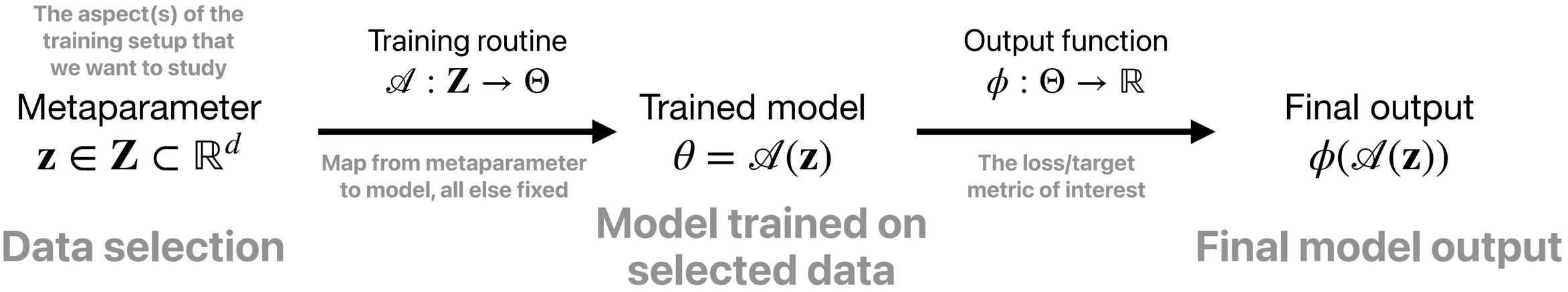


Goal (Data curation)

Find the selection $\mathbf{z}$ that minimizes the resulting model’s test loss

Problem: Metaparameter selection

Idea: write **model output** as a direct function of **training setup** (“metaparameter”)



Goal (Data curation)

Find the selection $\mathbf{z}$ that minimizes the resulting model’s test loss

Goal (Data attribution)

Predict how changing the selection $\mathbf{z}$ will change resulting model’s output

Roadmap

Main idea: The metagradient

Prequel: Two challenges

Looking forward

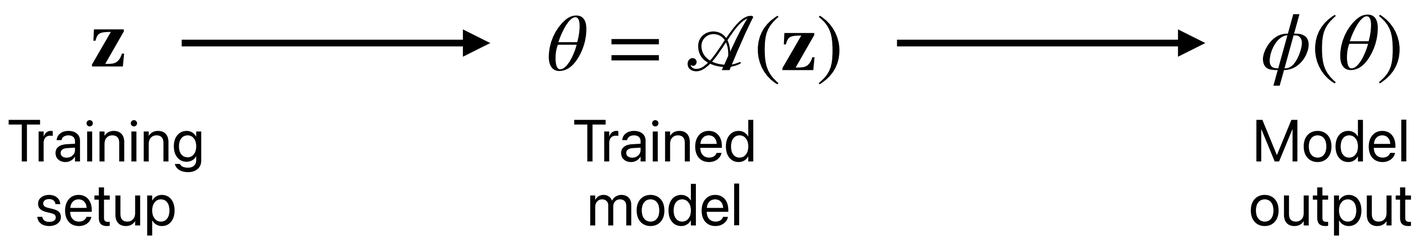
Roadmap

Main idea: The metagradient

Prequel: Two challenges

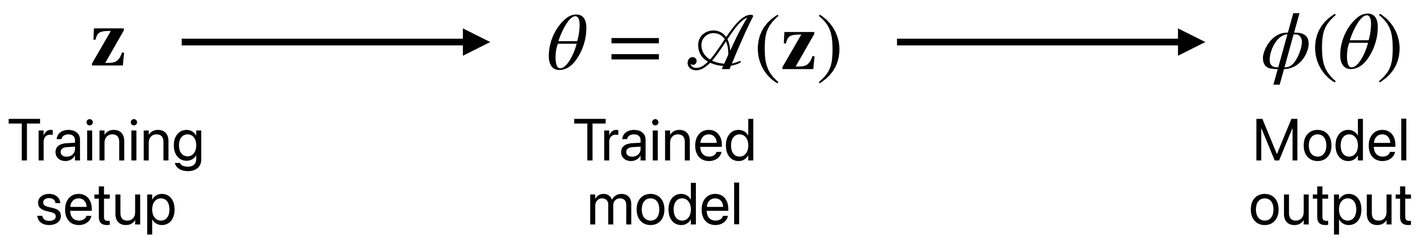
Looking forward

Main idea: The metagradient



Main idea: The metagradient

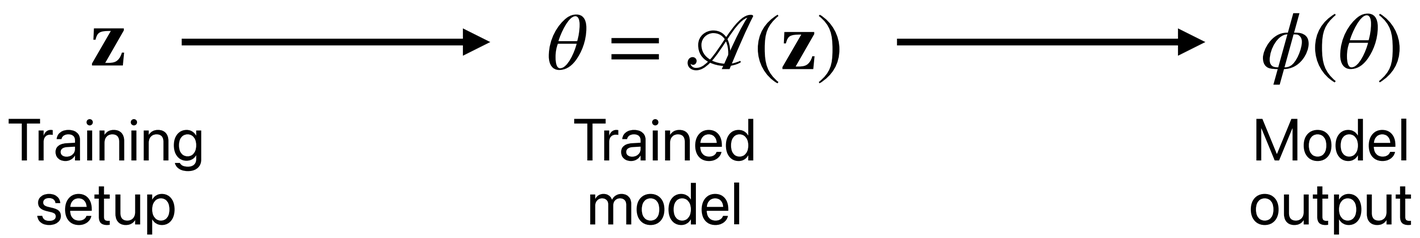
In many cases, metaparameter $\mathbf{z}$ is a continuous variable (or can be relaxed to one)



Main idea: The metagradient

In many cases, metaparameter $\mathbf{z}$ is a continuous variable (or can be relaxed to one)

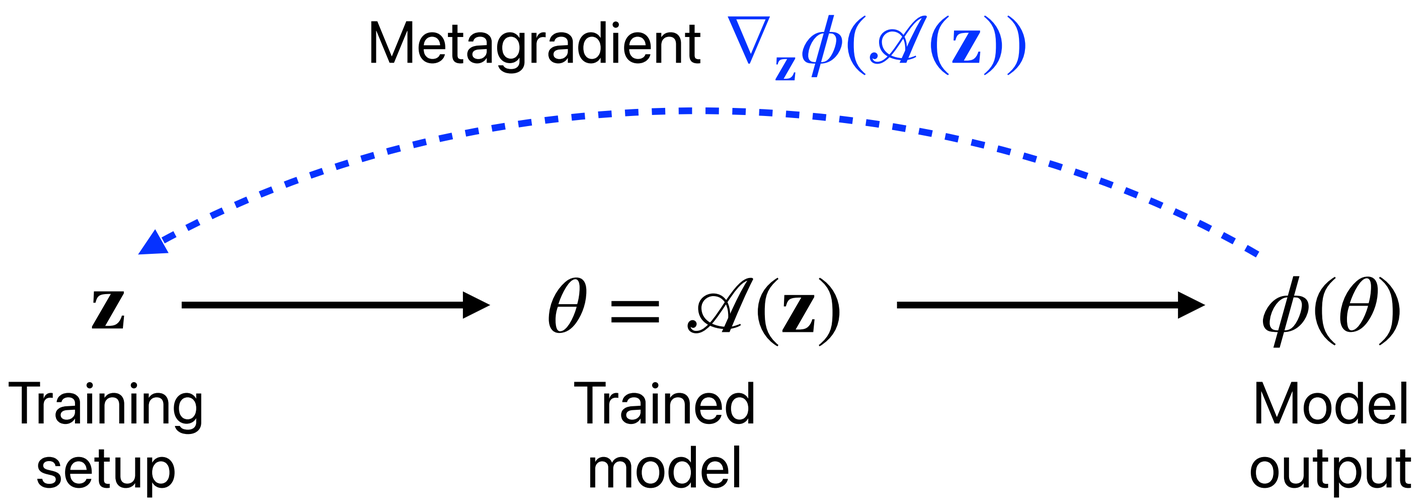
We will show how to **compute** and **use** the *metagradient*:



Main idea: The metagradient

In many cases, metaparameter $\mathbf{z}$ is a continuous variable (or can be relaxed to one)

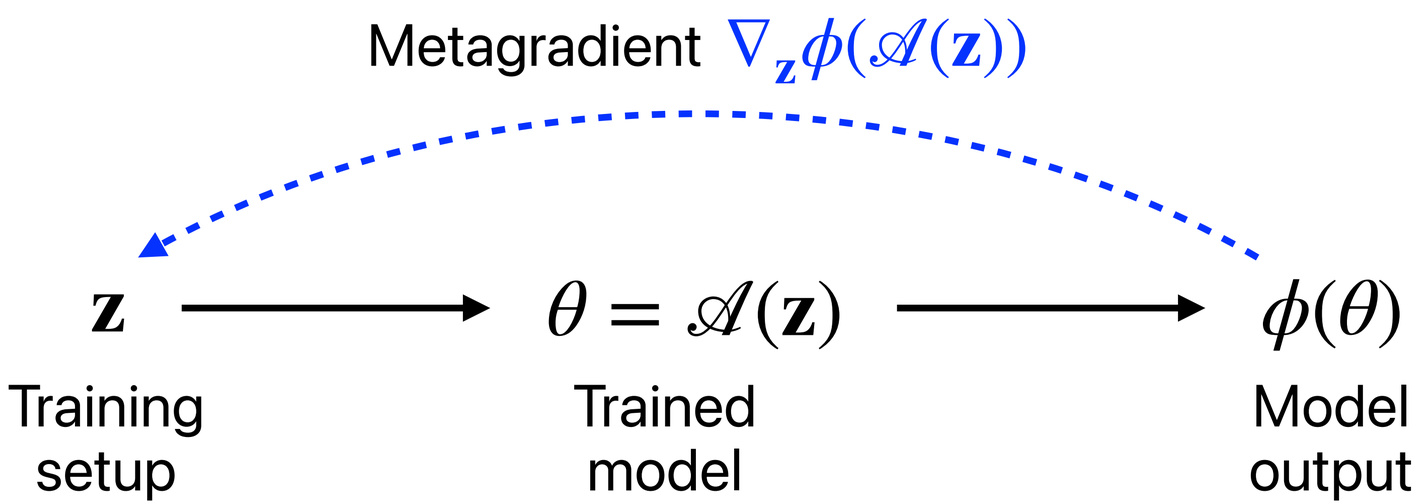
We will show how to **compute** and **use** the *metagradient*:



Main idea: The metagradient

In many cases, metaparameter $\mathbf{z}$ is a continuous variable (or can be relaxed to one)

We will show how to **compute** and **use** the *metagradient*:



Cost: In practice, "only" 3-5x the cost of model training!

Main idea: The metagradient

Metagradients offer a powerful way to optimize and predict (aspects of) model behavior

Optimizing with metagradients

Optimizing with metagradients

Given a task:

Optimizing with metagradients

Given a task:

1. **Rewrite** problem in terms of model outputs (as a function of a metaparameter)

Optimizing with metagradients

Given a task:

1. **Rewrite** problem in terms of model outputs (as a function of a metaparameter)

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes $\min_{z \in \mathbb{R}^d} \phi(\mathscr{A}(\mathbf{z}))$

Optimizing with metagradients

Given a task:

1. **Rewrite** problem in terms of model outputs (as a function of a metaparameter)

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes $\min_{z \in \mathbb{R}^d} \phi(\mathscr{A}(\mathbf{z}))$

2. **Solve** via (meta)gradient descent:

Optimizing with metagradients

Given a task:

1. **Rewrite** problem in terms of model outputs (as a function of a metaparameter)

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes $\min_{z \in \mathbb{R}^d} \phi(\mathscr{A}(\mathbf{z}))$

2. **Solve** via (meta)gradient descent:

   A. Pick a starting $\mathbf{z}^{(0)}$ and number of steps T

Optimizing with metagradients

Given a task:

1. **Rewrite** problem in terms of model outputs (as a function of a metaparameter)

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes $\min_{z \in \mathbb{R}^d} \phi(\mathscr{A}(\mathbf{z}))$

2. **Solve** via (meta)gradient descent:
   A. Pick a starting $\mathbf{z}^{(0)}$ and number of steps T
   B. For $i = 1 \ldots T$: Compute $\mathbf{g} := \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}^{(i-1)}))$ and descend

Optimizing with metagradients

Given a task:

1. **Rewrite** problem in terms of model outputs (as a function of a metaparameter)

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes $\min_{z \in \mathbb{R}^d} \phi(\mathscr{A}(\mathbf{z}))$

2. **Solve** via (meta)gradient descent:
   A. Pick a starting $\mathbf{z}^{(0)}$ and number of steps T
   B. For $i = 1 \ldots T$: Compute $\mathbf{g} := \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}^{(i-1)}))$ and descend

      Requires T total metagradient calculations

Task: Data selection

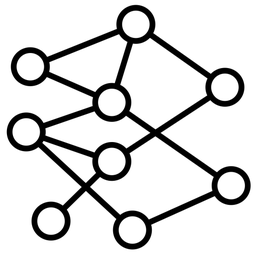
Task: Data selection

Choose train subset

Task: Data selection

Choose train subset

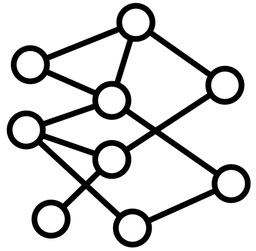
Train a model



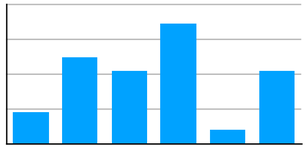
Task: Data selection

Choose train subset

Train a model



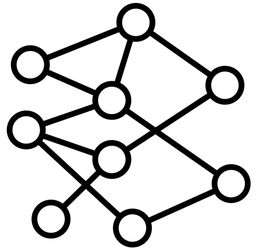
Measure loss on a target/test set:



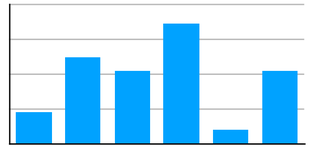
Task: Data selection

Choose train subset

Train a model



Measure loss on a target/test set:

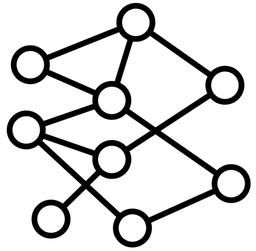


Goal: Select the training data subset minimizing loss on the target set

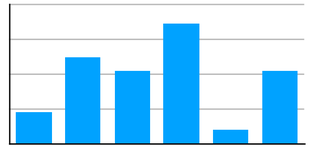
Task: Data selection

Choose train subset

Train a model



Measure loss on a target/test set:



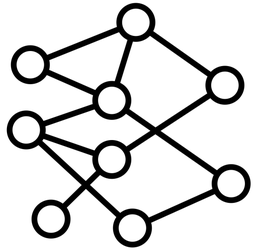
Goal: Select the training data subset minimizing loss on the target set

Equivalent goal: Find the training data *weighting* minimizing loss on the target set

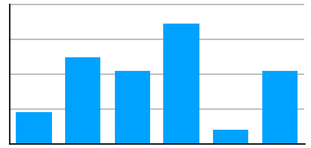
Task: Data selection

Choose train subset

Train a model



Measure loss on a target/test set:



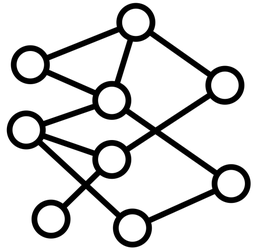
Goal: Select the training data subset minimizing loss on the target set

Equivalent goal: Find the training data *weighting* minimizing loss on the target set

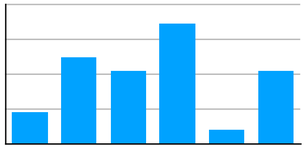
Task: Data selection

Choose train subset

Train a model



Measure loss on a target/test set:



Goal: Select the training data subset minimizing loss on the target set

Equivalent goal: Find the training data *weighting* minimizing loss on the target set

Increasingly recognized as a critical step in large-scale ML pipeline

Task: Selecting CLIP data

Approach

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Output function: Loss function L_{targ} evaluated on target tasks

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Output function: Loss function L_{targ} evaluated on target tasks

---

Repeat, starting from $\mathbf{z} = \mathbf{1}_N$:

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Output function: Loss function L_{targ} evaluated on target tasks

---

Repeat, starting from $\mathbf{z} = \mathbf{1}_N$:

1. Train model weighting the training points by $\mathbf{z}$

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Output function: Loss function L_{targ} evaluated on target tasks

---

Repeat, starting from $\mathbf{z} = \mathbf{1}_N$:

1. Train model weighting the training points by $\mathbf{z}$
2. Calculate metagradient $\mathbf{g} = \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Output function: Loss function L_{targ} evaluated on target tasks

---

Repeat, starting from $\mathbf{z} = \mathbf{1}_N$:

1. Train model weighting the training points by $\mathbf{z}$
2. Calculate metagradient $\mathbf{g} = \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$
3. Update $\mathbf{z} \leftarrow \mathbf{z} - \operatorname{sign}(\mathbf{g})$

Task: Selecting CLIP data

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^N$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standardized CLIP training routine

Output function: Loss function L_{targ} evaluated on target tasks

---

Repeat, starting from $\mathbf{z} = \mathbf{1}_N$:

1. Train model weighting the training points by $\mathbf{z}$
2. Calculate metagradient $\mathbf{g} = \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$
3. Update $\mathbf{z} \leftarrow \mathbf{z} - \text{sign}(\mathbf{g})$

*simplified

Task: Selecting CLIP data

Setup

Task: Selecting CLIP data

Setup

DATACOMP:
In search of the next generation of multimodal datasets

Samir Yitzhak Gadre*[2], Gabriel Ilharco*[1], Alex Fang*[1], Jonathan Hayase[1], Georgios Smyrnis[5], Thao Nguyen[1], Ryan Marten[7,9], Mitchell Wortsman[1], Dhruba Ghosh[1], Jieyu Zhang[1], Eyal Orgad[3], Rahim Entezari[10], Giannis Daras[5], Sarah Pratt[1], Vivek Ramanujan[1], Yonatan Bitton[11], Kalyani Marathe[1], Stephen Mussmann[1], Richard Vencu[6], Mehdi Cherti[6,8], Ranjay Krishna[1], Pang Wei Koh[1,12], Olga Saukh[10], Alexander Ratner[1,13], Shuran Song[2], Hannaneh Hajishirzi[1,7], Ali Farhadi[1], Romain Beaumont[6], Sewoong Oh[1], Alex Dimakis[5], Jenia Jitsev[6,8], Yair Carmon[3], Vaishaal Shankar[4], Ludwig Schmidt[1,6,7]

Task: Selecting CLIP data

Setup

DATACOMP:
In search of the next generation of multimodal datasets

Samir Yitzhak Gadre*[2], Gabriel Ilharco*[1], Alex Fang*[1], Jonathan Hayase[1], Georgios Smyrnis[5], Thao Nguyen[1], Ryan Marten[7,9], Mitchell Wortsman[1], Dhruba Ghosh[1], Jieyu Zhang[1], Eyal Orgad[3], Rahim Entezari[10], Giannis Daras[5], Sarah Pratt[1], Vivek Ramanujan[1], Yonatan Bitton[11], Kalyani Marathe[1], Stephen Mussmann[1], Richard Vencu[6], Mehdi Cherti[6,8], Ranjay Krishna[1], Pang Wei Koh[1,12], Olga Saukh[10], Alexander Ratner[1,13], Shuran Song[2], Hannaneh Hajishirzi[1,7], Ali Farhadi[1], Romain Beaumont[6], Sewoong Oh[1], Alex Dimakis[5], Jenia Jitsev[6,8], Yair Carmon[3], Vaishaal Shankar[4], Ludwig Schmidt[1,6,7]

Data selection task where the learning algorithm (CLIP training) and evaluation (zero-shot classification) are **fixed**

Task: Selecting CLIP data

Setup

DATACOMP:
In search of the next generation of multimodal datasets

Samir Yitzhak Gadre*[2], Gabriel Ilharco*[1], Alex Fang*[1], Jonathan Hayase[1], Georgios Smyrnis[5], Thao Nguyen[1], Ryan Marten[7,9], Mitchell Wortsman[1], Dhruba Ghosh[1], Jieyu Zhang[1], Eyal Orgad[3], Rahim Entezari[10], Giannis Daras[5], Sarah Pratt[1], Vivek Ramanujan[1], Yonatan Bitton[11], Kalyani Marathe[1], Stephen Mussmann[1], Richard Vencu[6], Mehdi Cherti[6,8], Ranjay Krishna[1], Pang Wei Koh[1,12], Olga Saukh[10], Alexander Ratner[1,13], Shuran Song[2], Hannaneh Hajishirzi[1,7], Ali Farhadi[1], Romain Beaumont[6], Sewoong Oh[1], Alex Dimakis[5], Jenia Jitsev[6,8], Yair Carmon[3], Vaishaal Shankar[4], Ludwig Schmidt[1,6,7]

Data selection task where the learning algorithm (CLIP training) and evaluation (zero-shot classification) are **fixed**

Goal: Curate the best possible dataset

Task: Selecting CLIP data

Setup

DATACOMP:
In search of the next generation of multimodal datasets

Samir Yitzhak Gadre*[2], Gabriel Ilharco*[1], Alex Fang*[1], Jonathan Hayase[1], Georgios Smyrnis[5], Thao Nguyen[1], Ryan Marten[7,9], Mitchell Wortsman[1], Dhruba Ghosh[1], Jieyu Zhang[1], Eyal Orgad[3], Rahim Entezari[10], Giannis Daras[5], Sarah Pratt[1], Vivek Ramanujan[1], Yonatan Bitton[11], Kalyani Marathe[1], Stephen Mussmann[1], Richard Vencu[6], Mehdi Cherti[6,8], Ranjay Krishna[1], Pang Wei Koh[1,12], Olga Saukh[10], Alexander Ratner[1,13], Shuran Song[2], Hannaneh Hajishirzi[1,7], Ali Farhadi[1], Romain Beaumont[6], Sewoong Oh[1], Alex Dimakis[5], Jenia Jitsev[6,8], Yair Carmon[3], Vaishaal Shankar[4], Ludwig Schmidt[1,6,7]

Data selection task where the learning algorithm (CLIP training) and evaluation (zero-shot classification) are **fixed**

Goal: Curate the best possible dataset

Example leaderboard (small):

Rank	Created	Submission	ImageNet acc.	Average perf.
1	03-14-2025	M-FLYT + SCS (alpha=0.5)	0.080	0.185
2	08-18-2024	EcoDatum	0.053	0.182
3	08-12-2023	> w <	0.056	0.180
4	11-04-2023	Hype Sampler	0.063	0.180
5	01-08-2024	Content Alignment Model ∩ Flipped CLIP Score	0.064	0.178
6	09-07-2023	Detection ∩ CLIP score	0.053	0.178
7	09-06-2023	BLIP2-COCO-finetuned_similarity_top-35%	0.053	0.177
8	08-03-2023	WS (baselines)	0.059	0.175
9	04-28-2023	Baseline: CLIP score (L/14 30%)	0.051	0.173
10	09-07-2023	OCR and Naive english filtering	0.057	0.173
11	08-05-2024	MoDO	0.051	0.173
12	08-25-2023	Fondant baseline, CLIP score (L/14 1%-30%), ascii > 4	0.053	0.168
13	08-25-2023	text masked CLIP 30%	0.048	0.164
14	08-02-2023	CLIP score (L/14 30%) baseline	0.049	0.164
15	04-28-2023	Baseline: Image-based	0.043	0.159

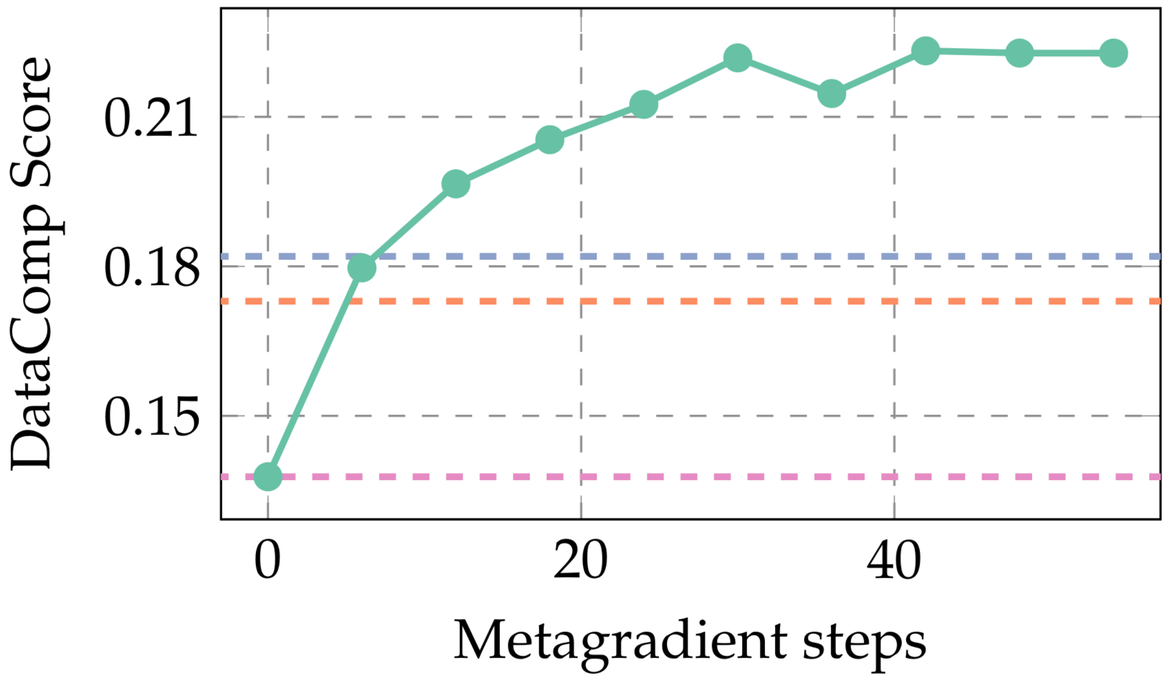
⋮

Task: Selecting CLIP data

Results (DataComp-small)

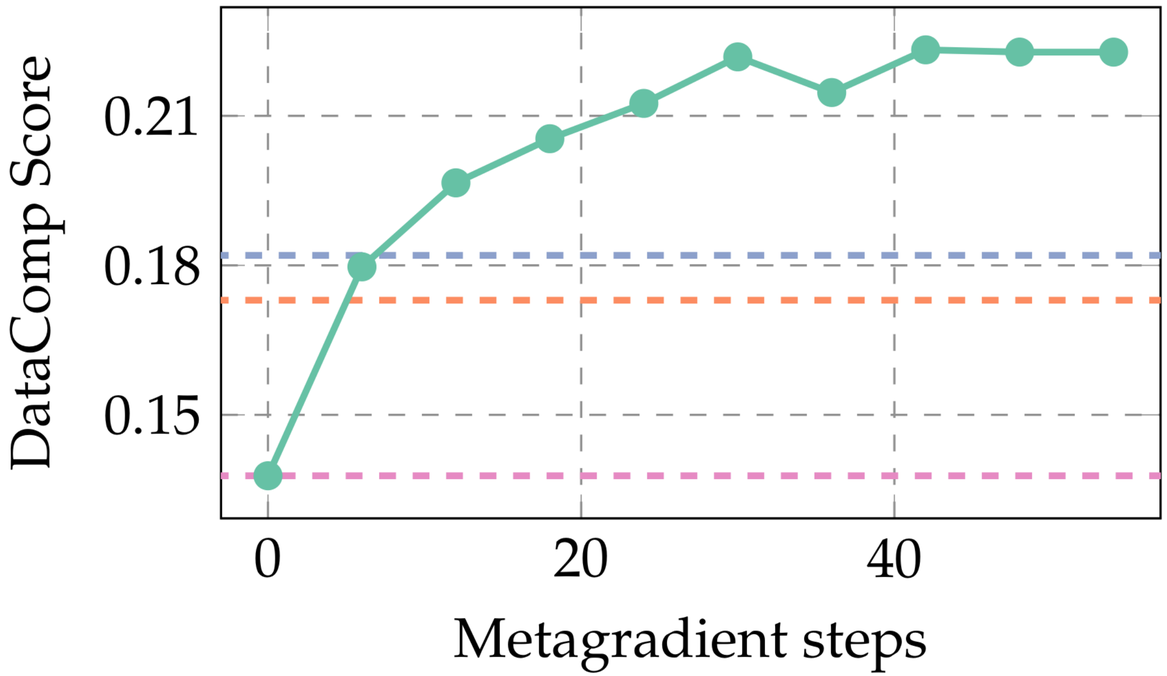
Task: Selecting CLIP data

Results (DataComp-small)



Task: Selecting CLIP data

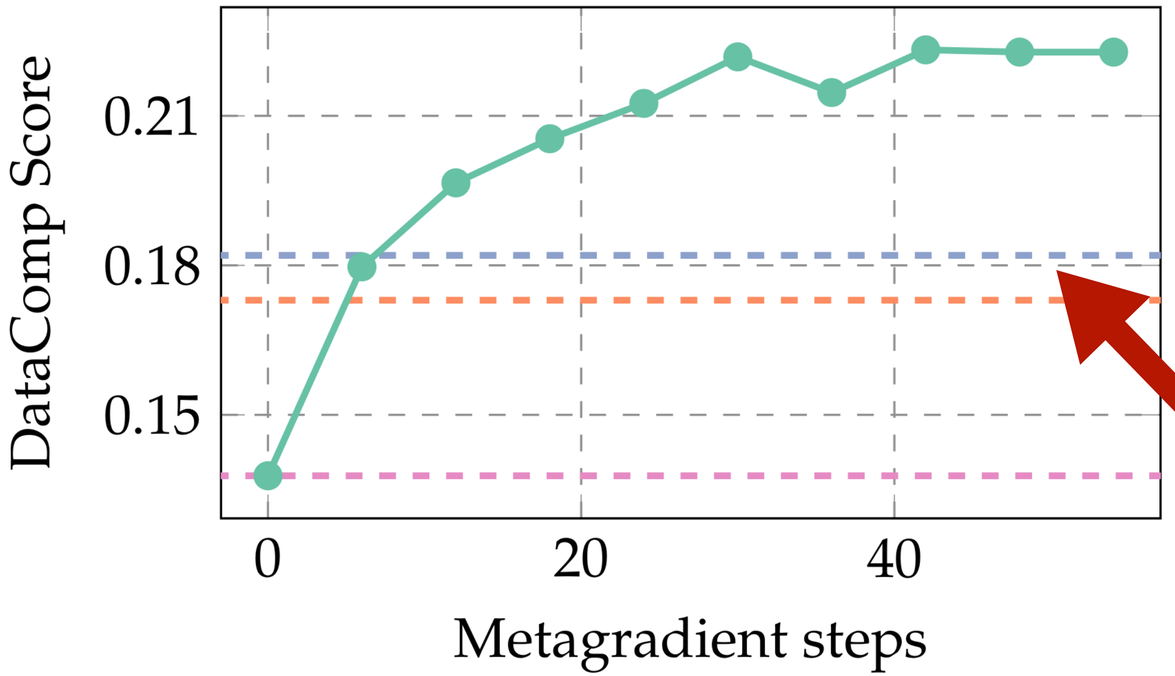
Results (DataComp-small)



Method	Score	Δ
Baseline: No filtering	0.13	–
Best baseline from [GIF+24]	0.17	+0.04
Previous SOTA [Eco24]	0.18	+0.05
MGD-DS (ours)	**0.22**	**+0.09**

Task: Selecting CLIP data

Results (DataComp-small)

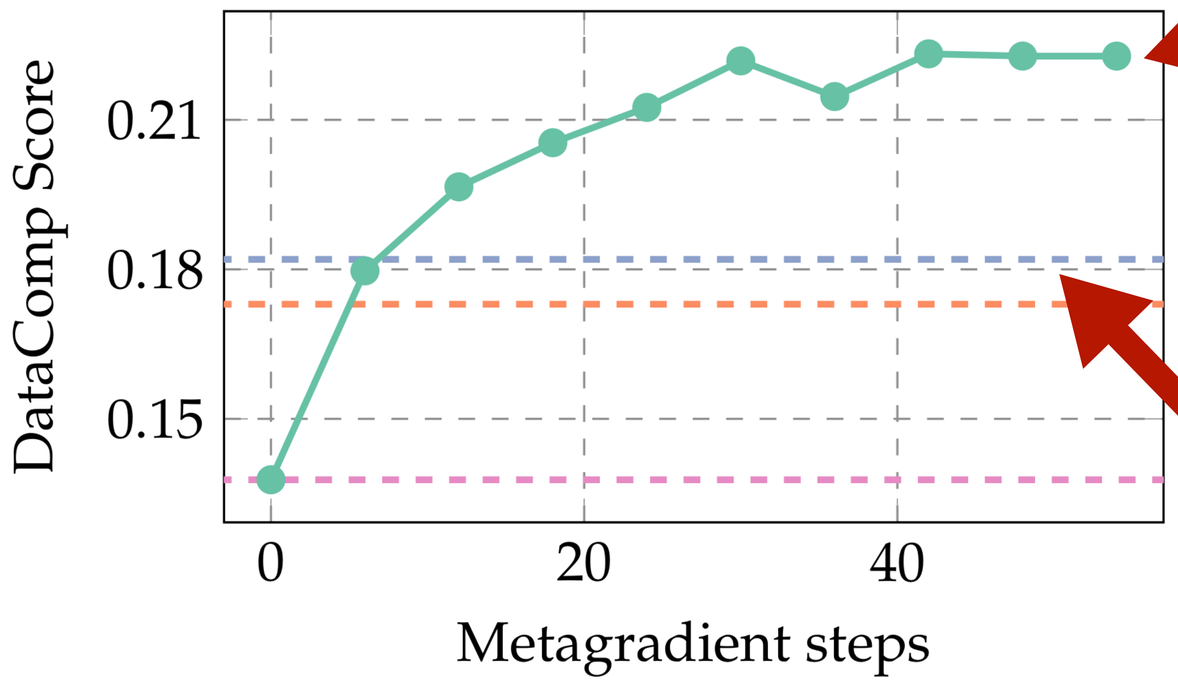


Method	Score	Δ
Baseline: No filtering	0.13	–
Best baseline from [GIF+24]	0.17	+0.04
Previous SOTA [Eco24]	0.18	+0.05
MGD-DS (ours)	**0.22**	**+0.09**

Best existing method

Task: Selecting CLIP data

Results (DataComp-small)



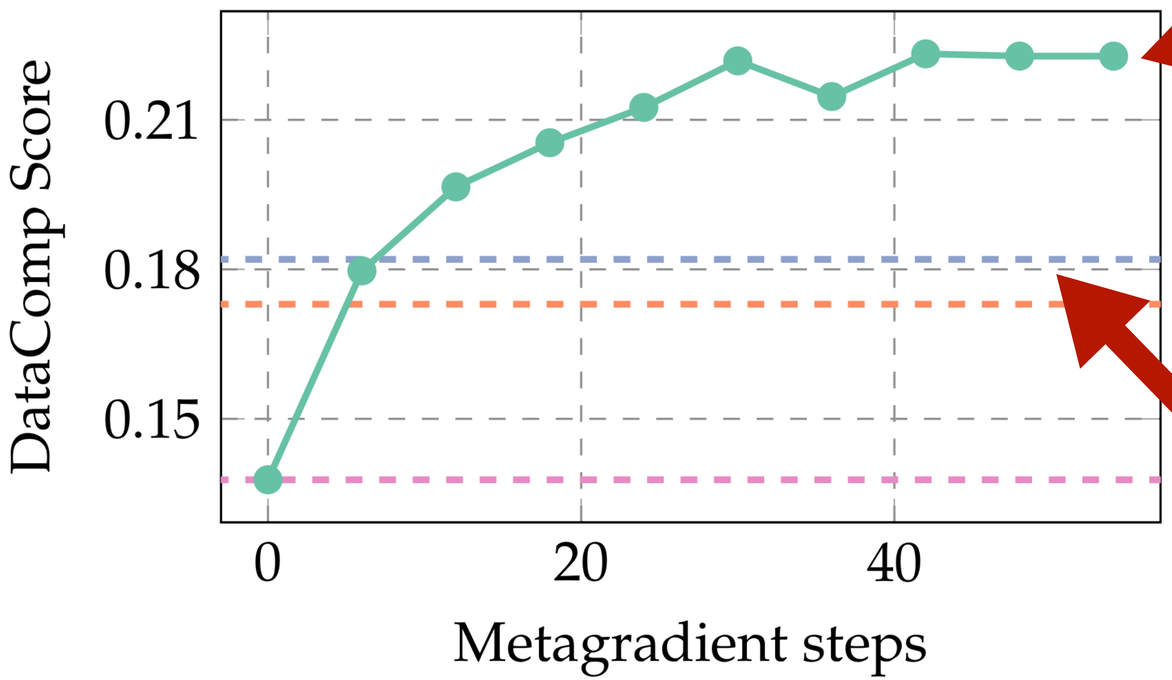
MGD improvement (relative to no selection) is **double** existing methods'

Method	Score	Δ
Baseline: No filtering	0.13	–
Best baseline from [GIF+24]	0.17	+0.04
Previous SOTA [Eco24]	0.18	+0.05
MGD-DS (ours)	**0.22**	**+0.09**

Best existing method

Task: Selecting CLIP data

Results (DataComp-small)



MGD improvement (relative to no selection) is **double** existing methods'

Method	Score	Δ
Baseline: No filtering	0.13	–
Best baseline from [GIF+24]	0.17	+0.04
Previous SOTA [Eco24]	0.18	+0.05
MGD-DS (ours)	**0.22**	**+0.09**

Best existing method

Now working on DataComp-medium (already #1!)

Task: Selecting instruction fine-tuning data

Results (LESS IFT setting)

Task: Selecting instruction fine-tuning data

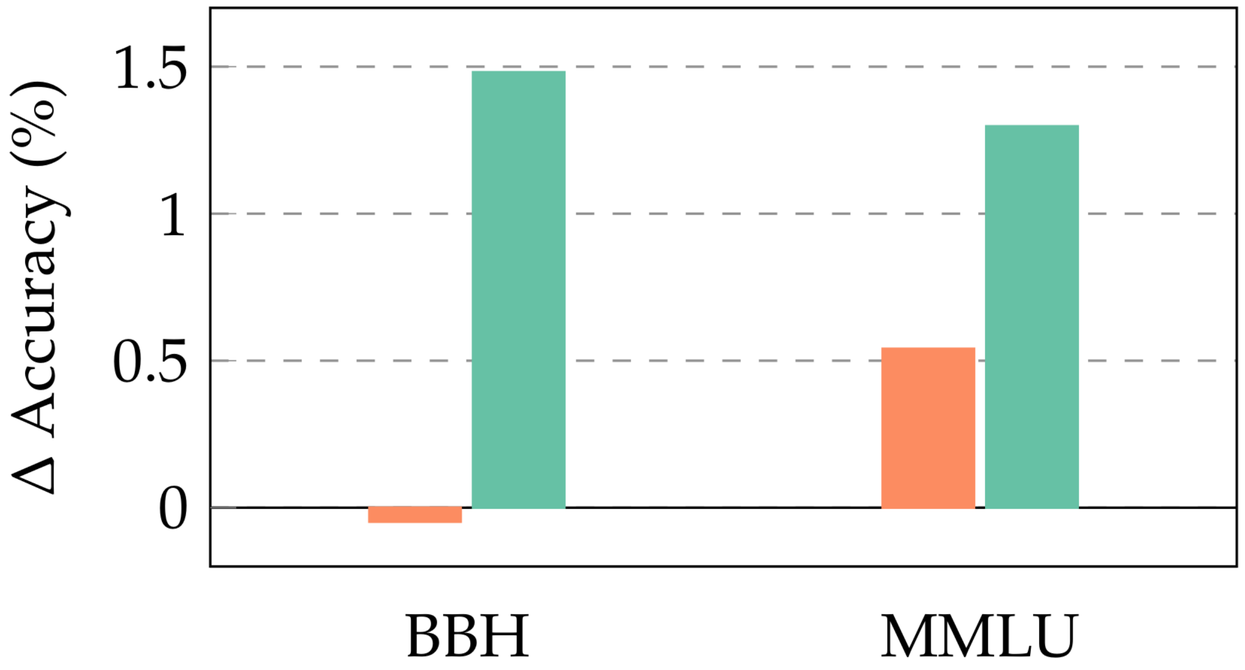
Results (LESS IFT setting)

New goal: Select instruction fine-tuning data for LM benchmarks (2B param model)

Task: Selecting instruction fine-tuning data

Results (LESS IFT setting)

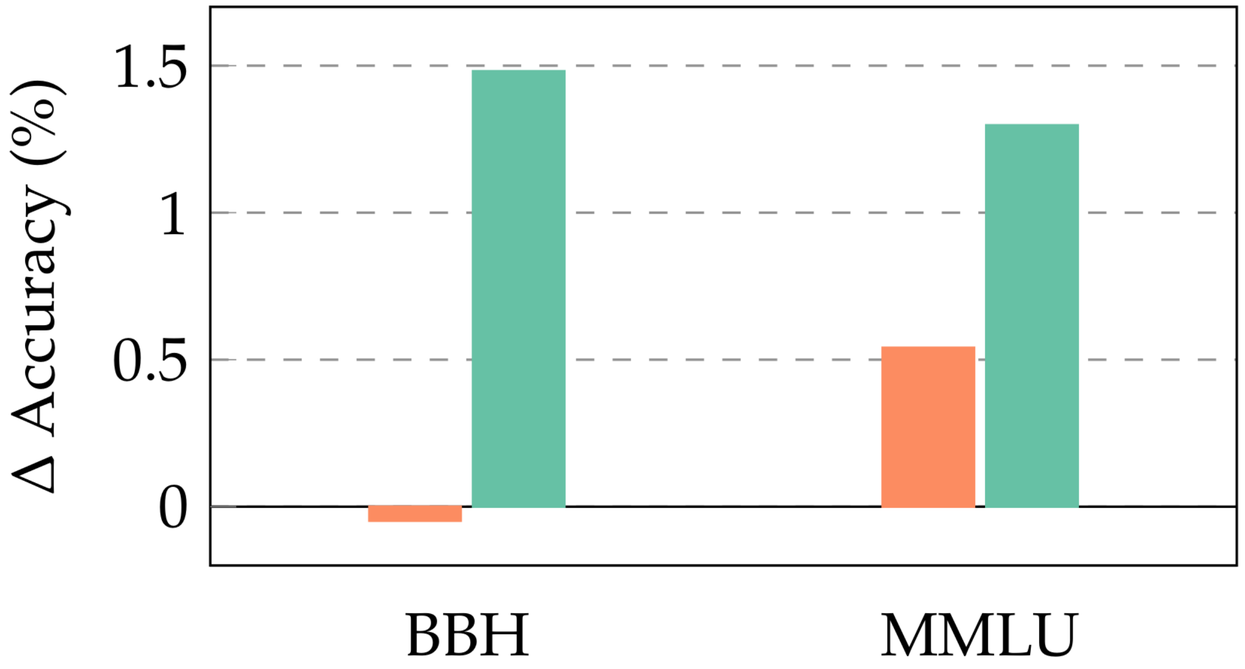
New goal: Select instruction fine-tuning data for LM benchmarks (2B param model)



Task: Selecting instruction fine-tuning data

Results (LESS IFT setting)

New goal: Select instruction fine-tuning data for LM benchmarks (2B param model)

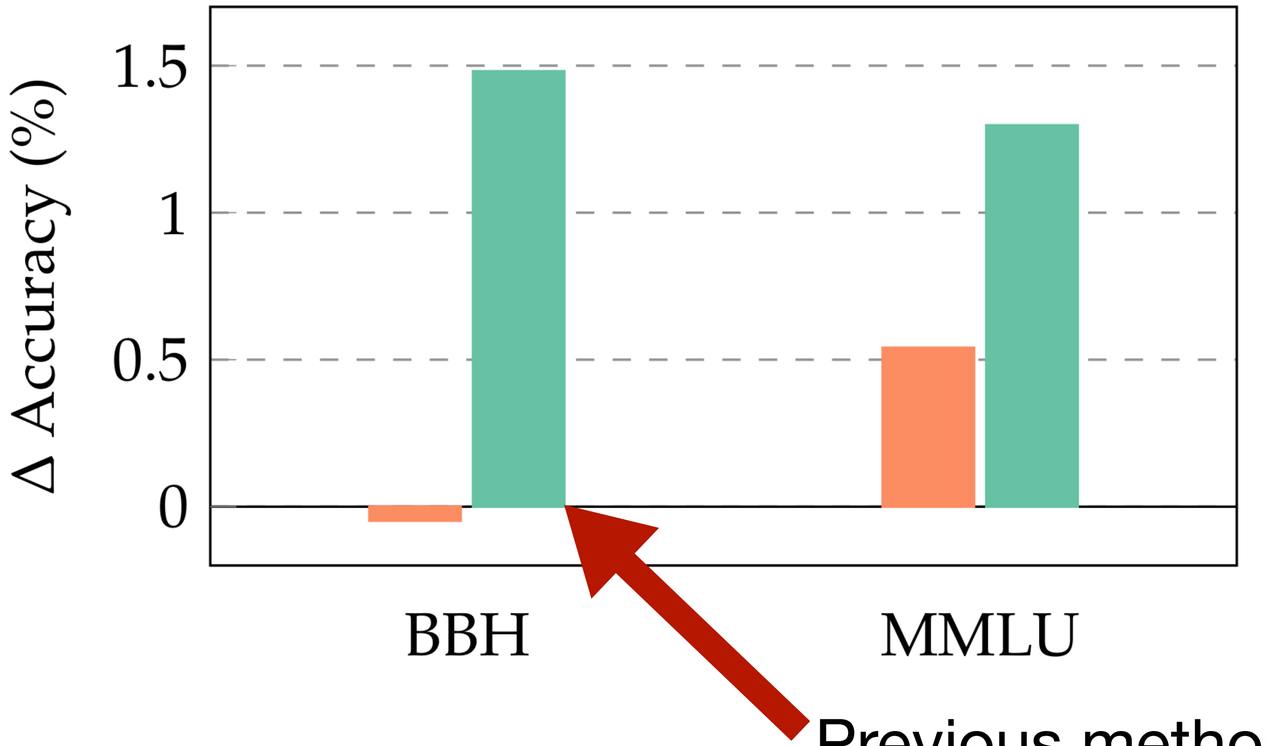


	BBH [SSS+22]		**MMLU** [HBB+20]	
	Acc.	Δ	Acc.	Δ
All Data	35.2%	–	41.2%	–
LESS	35.2%	−0.0%	41.8%	+0.5%
MGD-DS	**36.7%**	**+1.5%**	**42.5%**	**+1.3%**

Task: Selecting instruction fine-tuning data

Results (LESS IFT setting)

New goal: Select instruction fine-tuning data for LM benchmarks (2B param model)



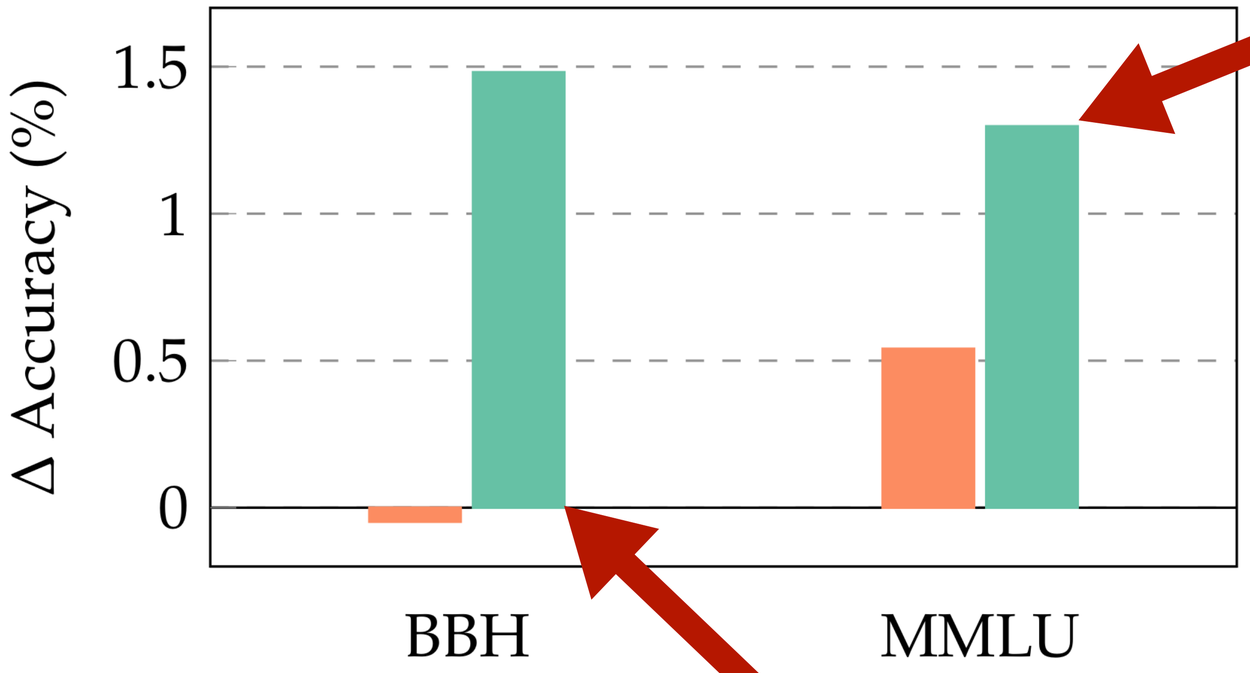
	BBH [SSS+22]		**MMLU** [HBB+20]	
	Acc.	Δ	Acc.	Δ
All Data	35.2%	–	41.2%	–
LESS	35.2%	−0.0%	41.8%	+0.5%
MGD-DS	**36.7%**	**+1.5%**	**42.5%**	**+1.3%**

Previous methods **harm** performance in BBH

Task: Selecting instruction fine-tuning data

Results (LESS IFT setting)

New goal: Select instruction fine-tuning data for LM benchmarks (2B param model)



2x the improvement in MMLU

	BBH [SSS+22]		**MMLU** [HBB+20]	
	Acc.	Δ	Acc.	Δ
All Data	35.2%	–	41.2%	–
LESS	35.2%	−0.0%	41.8%	+0.5%
MGD-DS	**36.7%**	**+1.5%**	**42.5%**	**+1.3%**

Previous methods **harm** performance in BBH

Task: Poisoning CIFAR-10 training data

Setup

Task: Poisoning CIFAR-10 training data

Setup

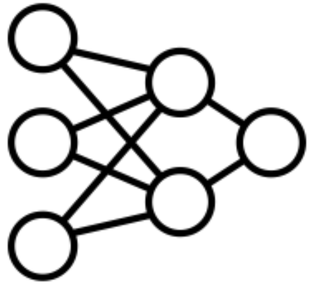
Insert poison data $\mathbf{z}$

Task: Poisoning CIFAR-10 training data

Setup

Insert poison data $\mathbf{z}$

Train model using fixed learning algorithm

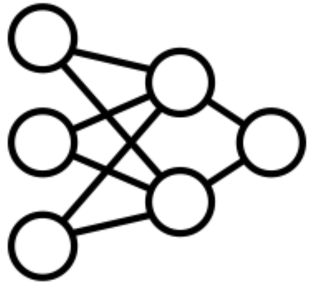


Task: Poisoning CIFAR-10 training data

Setup

Insert poison data $\mathbf{z}$

Train model using fixed learning algorithm



Goal: reduce overall test accuracy

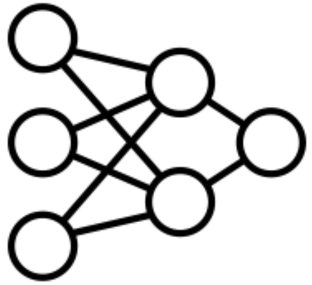
CIFAR10: 85%

Task: Poisoning CIFAR-10 training data

Setup

Insert poison data $\mathbf{z}$

Train model using fixed learning algorithm



Goal: reduce overall test accuracy

CIFAR10: 85%

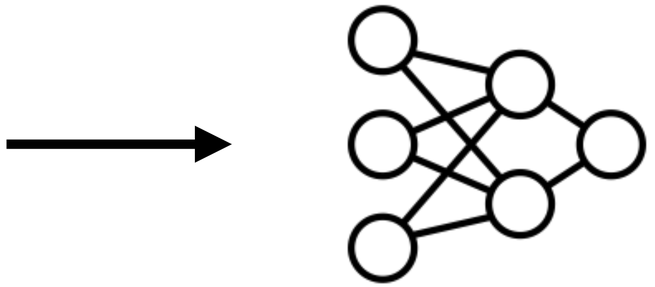
Goal: Insert poison datapoints that *maximize* overall test loss

Task: Poisoning CIFAR-10 training data

Setup

Insert poison data $\mathbf{z}$

Train model using fixed learning algorithm



Goal: reduce overall test accuracy

CIFAR10: 85%

> "However, in the setting of *indiscriminate* data poisoning, where an attacker aims to decrease the overall test accuracy by adding a small fraction of corrupted data, the effectiveness of existing attacks remains underwhelming."

[Lu Kamath Yu '23]

Goal: Insert poison datapoints that *maximize* overall test loss

Task: Poisoning CIFAR-10 training data

Approach

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Output function: Loss L_{val} evaluated on a heldout validation set

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Output function: Loss L_{val} evaluated on a heldout validation set

---

Repeat, starting from $\mathbf{z}$ being the first ε-fraction of the training data:

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Output function: Loss L_{val} evaluated on a heldout validation set

---

Repeat, starting from $\mathbf{z}$ being the first ε-fraction of the training data:

1. Train model, replacing the first ε-fraction of the training set with $\mathbf{z}$

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Output function: Loss L_{val} evaluated on a heldout validation set

---

Repeat, starting from $\mathbf{z}$ being the first ε-fraction of the training data:

1. Train model, replacing the first ε-fraction of the training set with $\mathbf{z}$
2. Calculate metagradient $\mathbf{g} = \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Output function: Loss L_{val} evaluated on a heldout validation set

---

Repeat, starting from $\mathbf{z}$ being the first ε-fraction of the training data:

1. Train model, replacing the first ε-fraction of the training set with $\mathbf{z}$
2. Calculate metagradient $\mathbf{g} = \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$
3. Update $\mathbf{z} \leftarrow \mathbf{z} + \eta \cdot \mathbf{g}$

Task: Poisoning CIFAR-10 training data

Approach

Metaparameter $\mathbf{z}$: tensor $\mathbf{z} \in \mathbb{R}^{\varepsilon N \times 32 \times 32 \times 3} \times \mathbb{R}^{\varepsilon N \times 10}$ of poison pixels and labels

Learning algorithm $\mathscr{A}$: standardized CIFAR-10 training (ResNet)

Output function: Loss L_{val} evaluated on a heldout validation set

---

Repeat, starting from $\mathbf{z}$ being the first ε-fraction of the training data:

1. Train model, replacing the first ε-fraction of the training set with $\mathbf{z}$
2. Calculate metagradient $\mathbf{g} = \nabla_{\mathbf{z}} \phi(\mathscr{A}(\mathbf{z}))$
3. Update $\mathbf{z} \leftarrow \mathbf{z} + \eta \cdot \mathbf{g}$

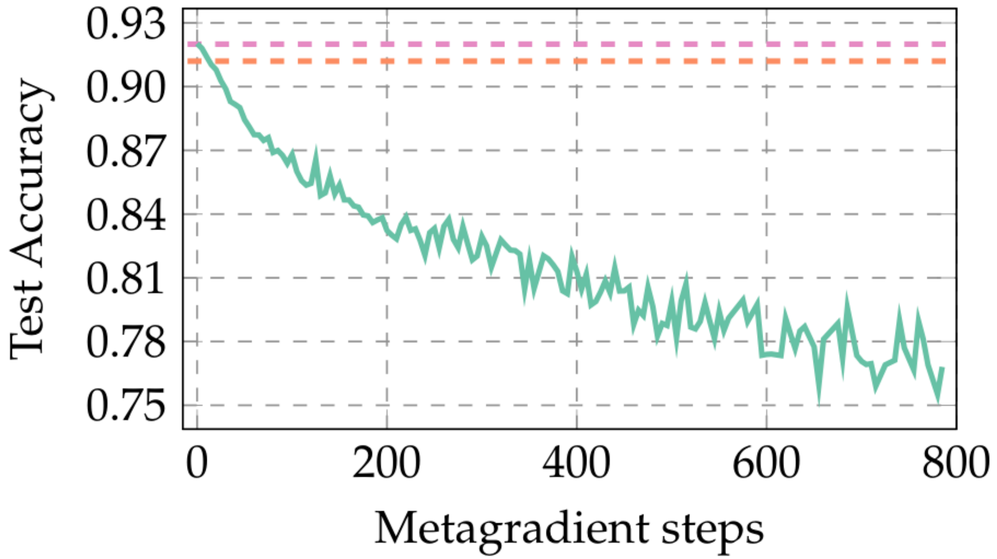
(this is the full algorithm)

Task: Poisoning CIFAR-10 training data

Results

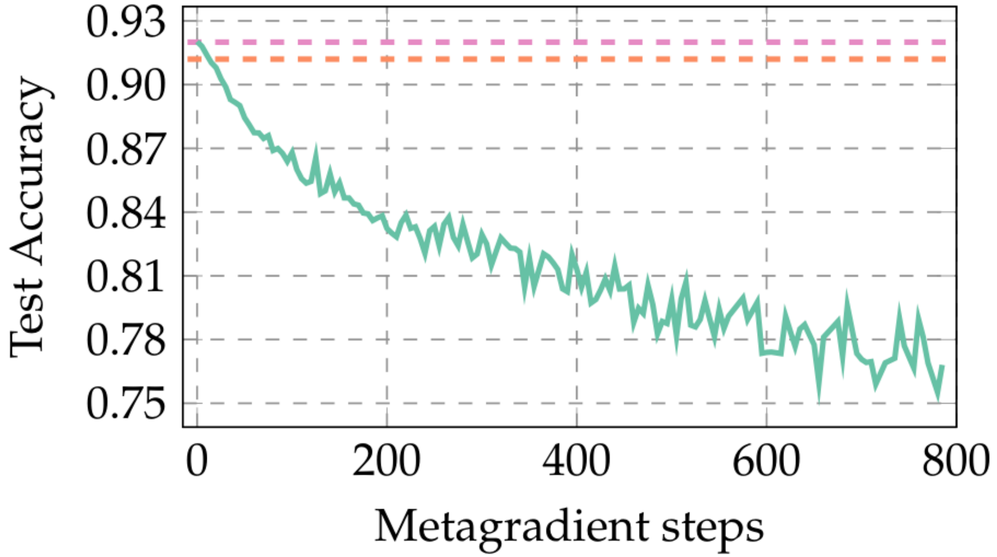
Task: Poisoning CIFAR-10 training data

Results



Task: Poisoning CIFAR-10 training data

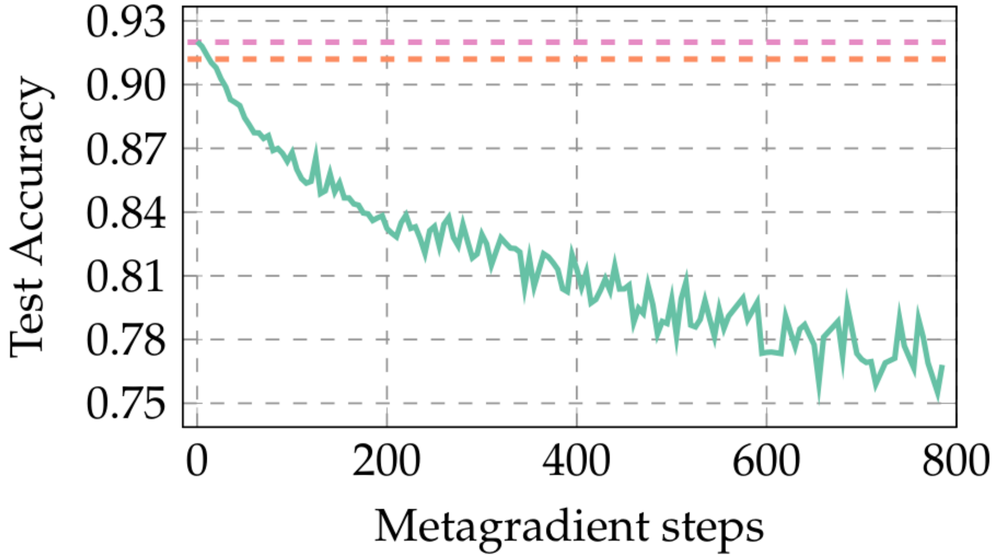
Results



Model	Acc.	Δ
Original model	92.0%	–
GradCancel [LKY23]	91.2%	−0.80%
MGD-DP (ours)	**78.1%**	**−13.9%**

Task: Poisoning CIFAR-10 training data

Results

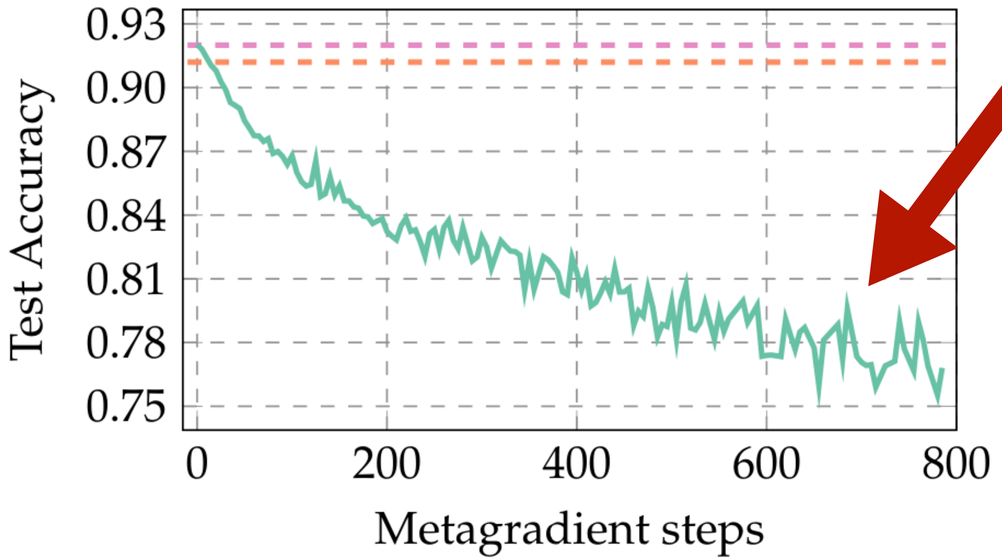


Model	Acc.	Δ
Original model	92.0%	–
GradCancel [LKY23]	91.2%	−0.80%
MGD-DP (ours)	**78.1%**	**−13.9%**

Poisoned samples:

Task: Poisoning CIFAR-10 training data

Results



We induce 17x the accuracy drop vs. existing methods

Model	Acc.	Δ
Original model	92.0%	–
GradCancel [LKY23]	91.2%	−0.80%
MGD-DP (ours)	**78.1%**	**−13.9%**

Poisoned samples:

Predicting with metagradients

Predicting with metagradients

Given a **prediction** task:

Predicting with metagradients

Given a **prediction** task:

1. **Rewrite** problem as a function of a metaparameter

Predicting with metagradients

Given a **prediction** task:

1. **Rewrite** problem as a function of a metaparameter

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes estimating $\phi(\mathscr{A}(\mathbf{z}))$

Predicting with metagradients

Given a **prediction** task:

1. **Rewrite** problem as a function of a metaparameter

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes estimating $\phi(\mathscr{A}(\mathbf{z}))$

2. For **reference metaparameter** $\mathbf{z}_0$, compute $\phi(\mathscr{A}(\mathbf{z_0}))$ and $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}_0))$

Predicting with metagradients

Given a **prediction** task:

1. **Rewrite** problem as a function of a metaparameter

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes estimating $\phi(\mathscr{A}(\mathbf{z}))$

2. For **reference metaparameter** $\mathbf{z}_0$, compute $\phi(\mathscr{A}(\mathbf{z_0}))$ and $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}_0))$

3. For **any** metaparameter $\mathbf{z}$, predict $\phi(\mathscr{A}(\mathbf{z}_0)) + (\mathbf{z} - \mathbf{z}_0)^\top \nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}_0))$

Predicting with metagradients

Given a **prediction** task:

1. **Rewrite** problem as a function of a metaparameter

   Decide on appropriate $\mathbf{z}$, $\mathscr{A}$, ϕ so that the problem becomes estimating $\phi(\mathscr{A}(\mathbf{z}))$

2. For **reference metaparameter** $\mathbf{z}_0$, compute $\phi(\mathscr{A}(\mathbf{z_0}))$ and $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}_0))$

3. For **any** metaparameter $\mathbf{z}$, predict $\phi(\mathscr{A}(\mathbf{z}_0)) + (\mathbf{z} - \mathbf{z}_0)^\top \nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}_0))$

   This is just a linear function of $\mathbf{z}$, no “extra computation” beyond $\mathbf{z}_0$

Task: Training data attribution

Setup

Task: Training data attribution

Setup

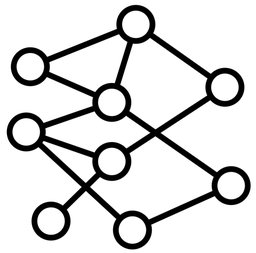
Choose train subset

Task: Training data attribution

Setup

Choose train subset

Train a model

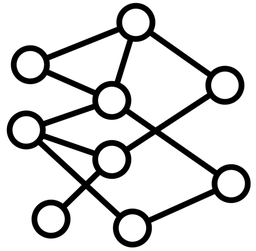


Task: Training data attribution

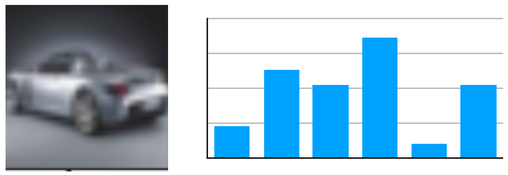
Setup

Choose train subset

Train a model



Measure loss on test example x:

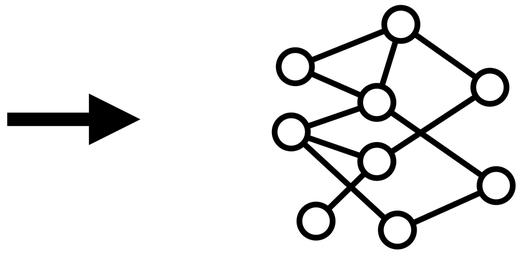


Task: Training data attribution

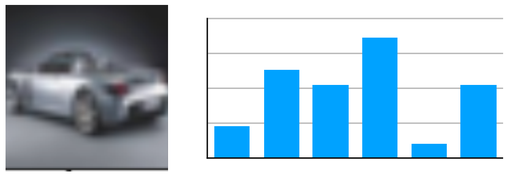
Setup

Choose train subset

Train a model



Measure loss on test example x:



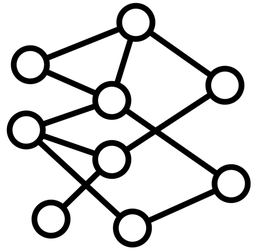
Goal: Predict the effect of removing some training data on the model's loss on x

Task: Training data attribution

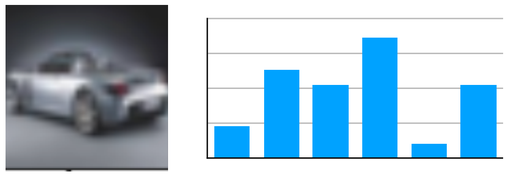
Setup

Choose train subset

Train a model



Measure loss on test example x:



Goal: Predict the effect of removing some training data on the model's loss on x

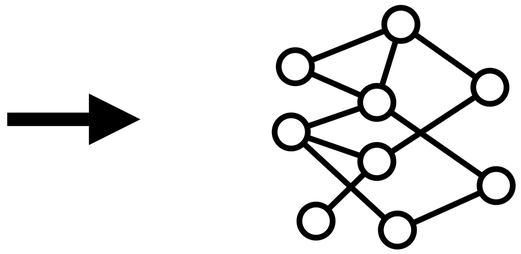
Equivalent goal: Predict the model's loss on x as a function of training data weighting

Task: Training data attribution

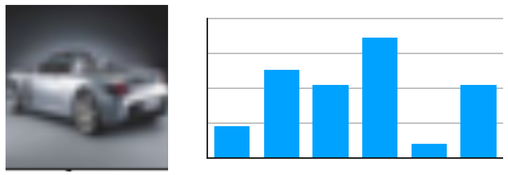
Setup

Choose train subset

Train a model



Measure loss on test example x:



Goal: Predict the effect of removing some training data on the model's loss on x

Equivalent goal: Predict the model's loss on x as a function of training data weighting

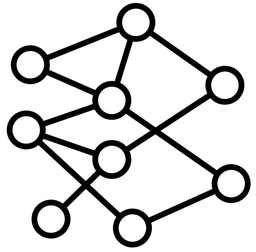
State of the art: predictions that are *moderately correlated* with model behavior

Task: Training data attribution

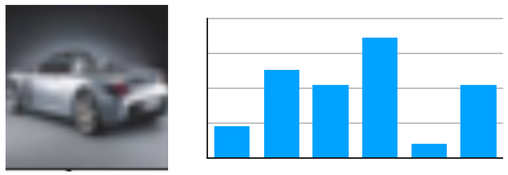
Setup

Choose train subset

Train a model



Measure loss on test example x:



Goal: Predict the effect of removing some training data on the model's loss on x

Equivalent goal: Predict the model's loss on x as a function of training data weighting

State of the art: predictions that are *moderately correlated* with model behavior

[Park Georgiev | Leclerc Madry '24; Grosse Bae Anil et al. '24; many others]

Task: Training data attribution

Approach

Task: Training data attribution

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^n$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Task: Training data attribution

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^n$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standard model training routine

Task: Training data attribution

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^n$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standard model training routine

Output function: Loss function $\ell(\,\cdot\,; x)$ evaluated on a specific test example

Task: Training data attribution

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^n$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standard model training routine

Output function: Loss function $\ell(\,\cdot\,; x)$ evaluated on a specific test example

---

Choose reference parameter to be all-ones vector and Taylor expand

Task: Training data attribution

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^n$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathscr{A}$: standard model training routine

Output function: Loss function $\ell(\,\cdot\,; x)$ evaluated on a specific test example

---

Choose reference parameter to be all-ones vector and Taylor expand

$$\phi(\mathscr{A}(\mathbf{z})) \approx \phi(\mathscr{A}((\mathbf{1}_n)) + \left.\frac{d\phi(\mathscr{A}(\mathbf{z}))}{d\mathbf{z}}\right|_{\mathbf{z}=\mathbf{1}_n} \left(\mathbf{z} - \mathbf{1}_n\right)$$

Task: Training data attribution

Approach

Metaparameter $\mathbf{z}$: vector $\mathbf{z} \in \mathbb{R}^n$ s.t., $\mathbf{z}_i$ is the weight on i-th training example

Learning algorithm $\mathcal{A}$: standard model training routine

Output function: Loss function $\ell(\cdot\,; x)$ evaluated on a specific test example

---

Choose reference parameter to be all-ones vector and Taylor expand

$$\phi(\mathcal{A}(\mathbf{z})) \approx \phi(\mathcal{A}((\mathbf{1}_n)) + \frac{d\phi(\mathcal{A}(\mathbf{z}))}{d\mathbf{z}}\bigg|_{\mathbf{z}=\mathbf{1}_n} \left(\mathbf{z} - \mathbf{1}_n\right)$$

AKA the exact influence function (Hampel ’47)

Task: Training data attribution

Results

Task: Training data attribution

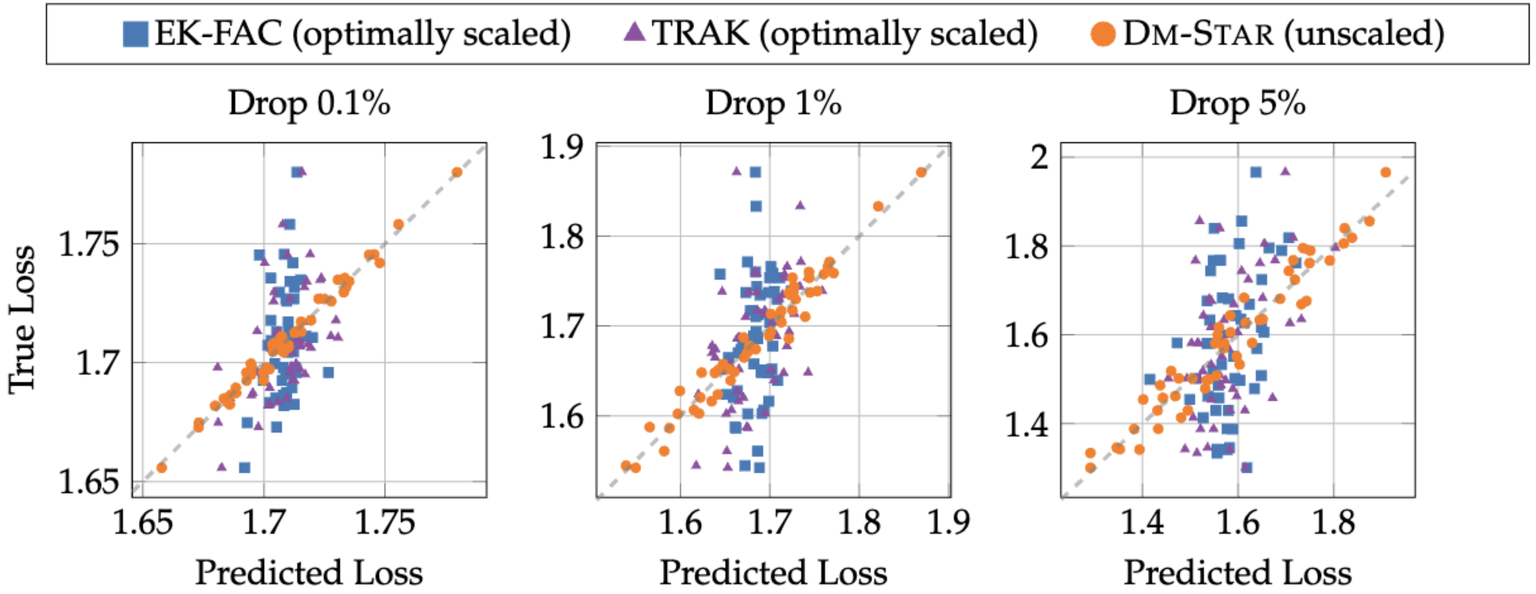
Results

Deleting 1% random subsets from CIFAR-10 training:

Task: Training data attribution

Results

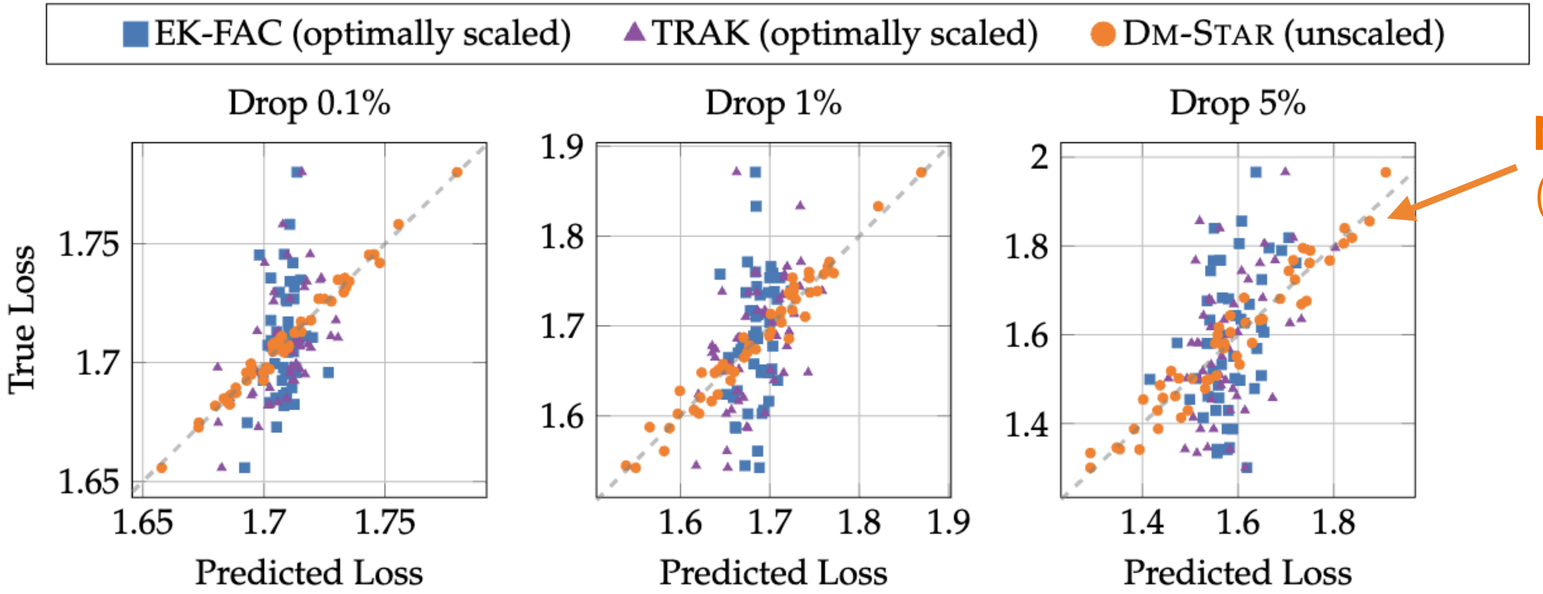
Deleting 1% random subsets from CIFAR-10 training:



Task: Training data attribution

Results

Deleting 1% random subsets from CIFAR-10 training:

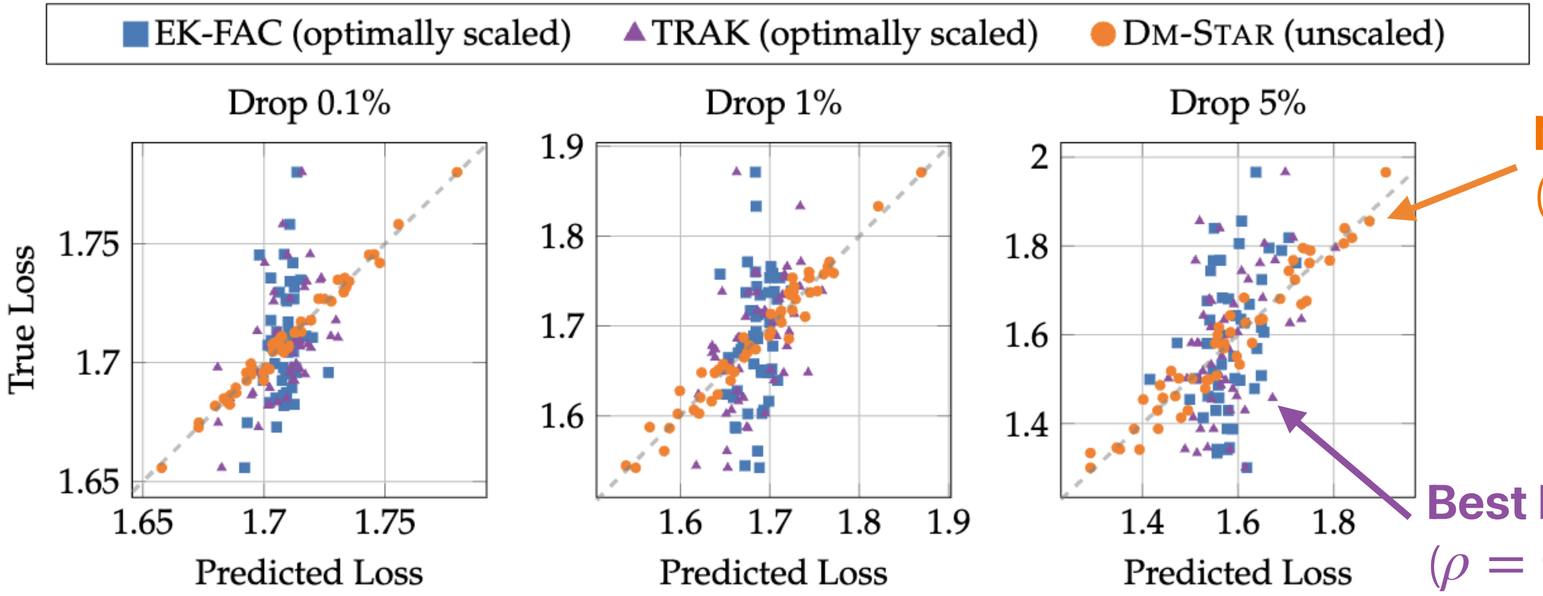


Near-perfect predictions
($\rho = 0.9$, well-scaled)

Task: Training data attribution

Results

Deleting 1% random subsets from CIFAR-10 training:



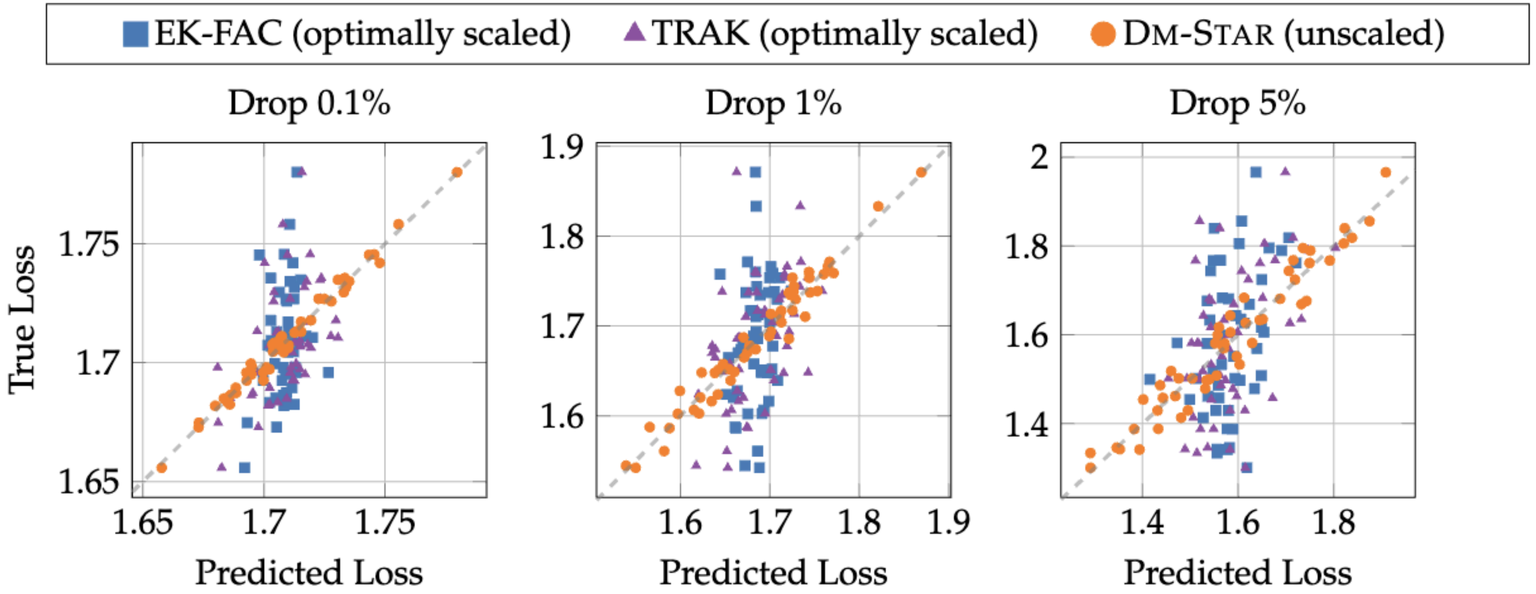
Near-perfect predictions
($\rho = 0.9$, well-scaled)

Best baseline
($\rho = 0.4$, uncalibrated)

Task: Training data attribution

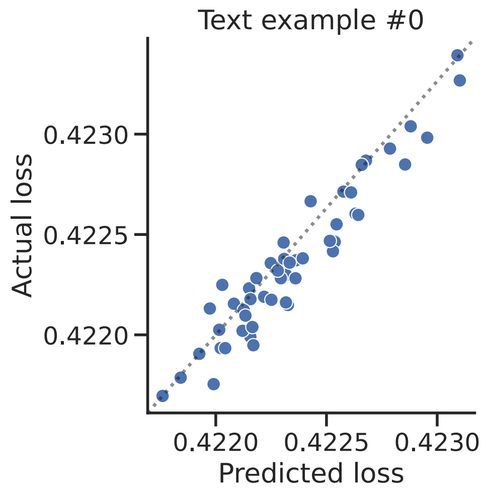
Results

Deleting 1% random subsets from CIFAR-10 training:



Also works for LMs!

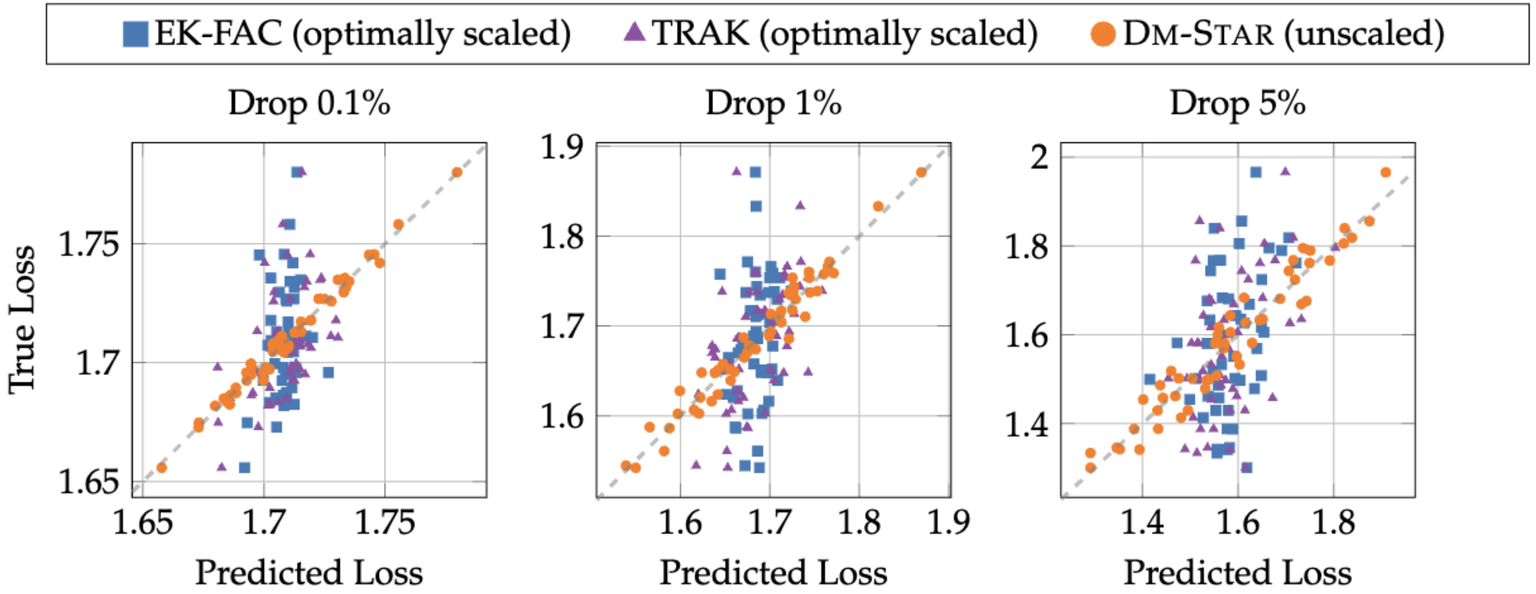
Deleting 1% of data from Gemma-2B fine-tuning



Task: Training data attribution

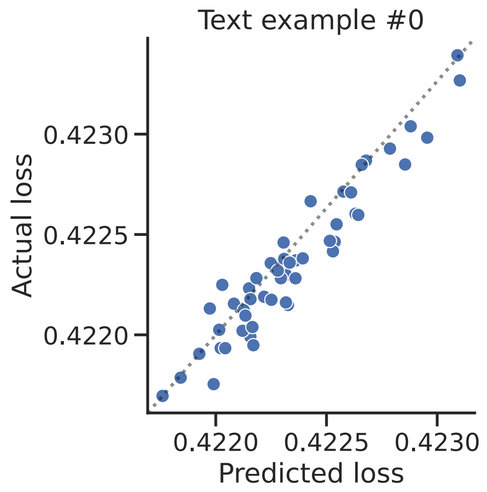
Results

Deleting 1% random subsets from CIFAR-10 training:



Also works for LMs!

Deleting 1% of data from Gemma-2B fine-tuning



(From a different paper; online soon!)

Roadmap

~~Main idea: The metagradient~~

Prequel: Two challenges

Looking forward

A brief history of the metagradient

Why hasn't this been done before?

A brief history of the metagradient

Why hasn't this been done before?

Exact metagradients at small scale [Maclaurin Duvenaud Adams '15; Franceschi et al. '17; ...]

- Optimized LR, weight decay, init, etc. for 100-iteration MNIST training
- Largest-scale setting: four-layer network on phone classification dataset

A brief history of the metagradient

Why hasn't this been done before?

Exact metagradients at small scale [Maclaurin Duvenaud Adams '15; Franceschi et al. '17; ...]

- Optimized LR, weight decay, init, etc. for 100-iteration MNIST training
- Largest-scale setting: four-layer network on phone classification dataset

Approximate (but large-scale) metagradients via implicit differentiation [Lorraine et al. '19]

- Scalable but coarse approximation based on convergence assumption; requires Hessian estimation & cannot optimize optimizer parameters such as learning rate

A brief history of the metagradient

Why hasn't this been done before?

Exact metagradients at small scale [Maclaurin Duvenaud Adams '15; Franceschi et al. '17; ...]

- Optimized LR, weight decay, init, etc. for 100-iteration MNIST training
- Largest-scale setting: four-layer network on phone classification dataset

Approximate (but large-scale) metagradients via implicit differentiation [Lorraine et al. '19]

- Scalable but coarse approximation based on convergence assumption; requires Hessian estimation & cannot optimize optimizer parameters such as learning rate

Note: These are just three representative works; significant line of work on differentiating through optimization [e.g., Domke '12; Nichol et al. '18; Hara '19; Shaban et al. '19; ...]

A brief history of the metagradient

Why hasn't this been done before?

Exact metagradients at small scale [Maclaurin Duvenaud Adams '15; Franceschi et al. '17; ...]

- Optimized LR, weight decay, init, etc. for 100-iteration MNIST training
- Largest-scale setting: four-layer network on phone classification dataset

Approximate (but large-scale) metagradients via implicit differentiation [Lorraine et al. '19]

- Scalable but coarse approximation based on convergence assumption; requires Hessian estimation & cannot optimize optimizer parameters such as learning rate

Note: These are just three representative works; significant line of work on differentiating through optimization [e.g., Domke '12; Nichol et al. '18; Hara '19; Shaban et al. '19; ...]

Broadly, nothing has scaled to "medium"-size training (e.g., >10M params, >1k iters)

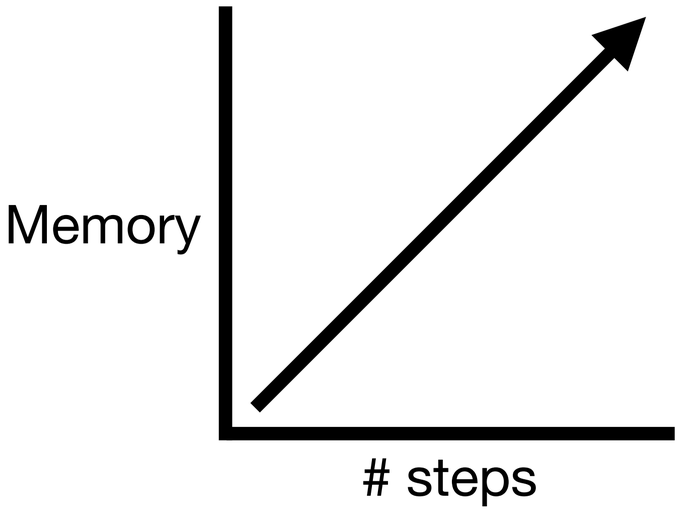
Metagradient roadmap: Two challenges

Metagradient roadmap: Two challenges

Two barriers to optimizing with the metagradient $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}))$

Metagradient roadmap: Two challenges

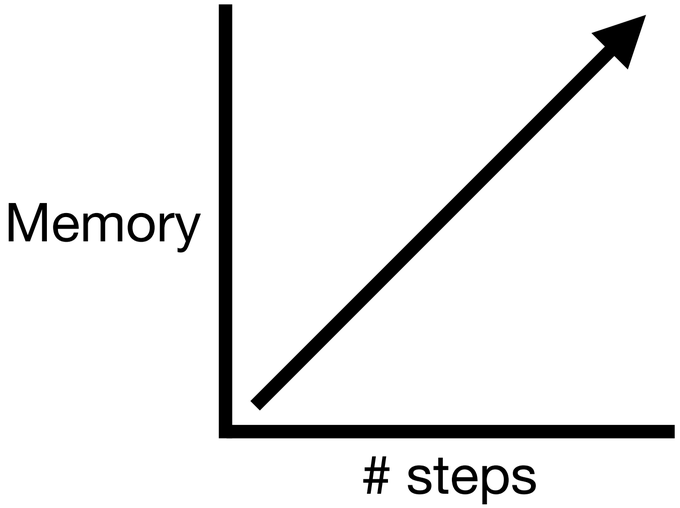
Two barriers to optimizing with the metagradient $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}))$



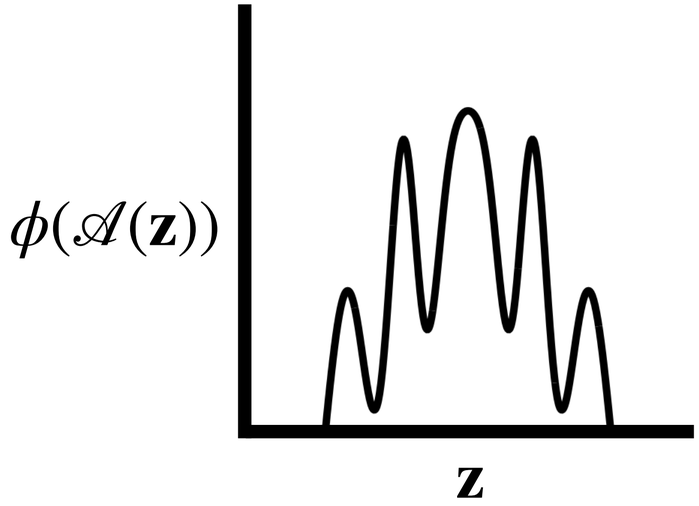
#1: Computational tractability

Metagradient roadmap: Two challenges

Two barriers to optimizing with the metagradient $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}))$



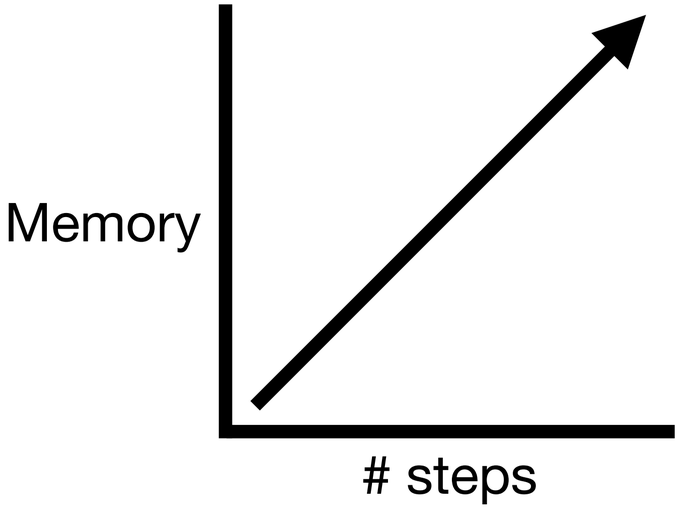
#1: Computational tractability



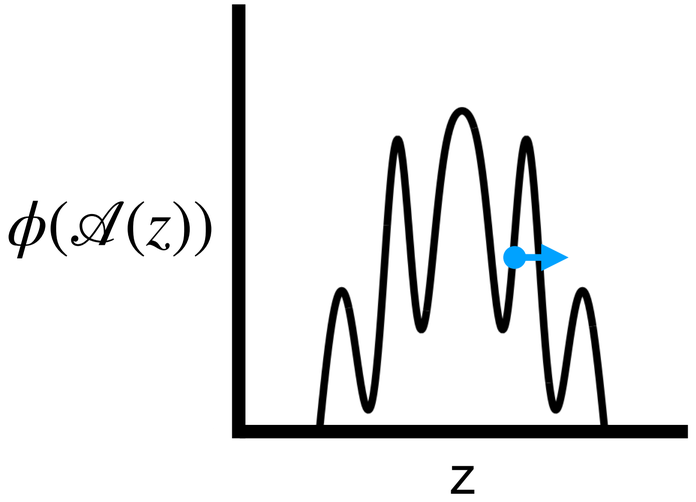
#2: Optimization (non)smoothness

Metagradient roadmap: Two challenges

Two barriers to optimizing with the metagradient $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}))$



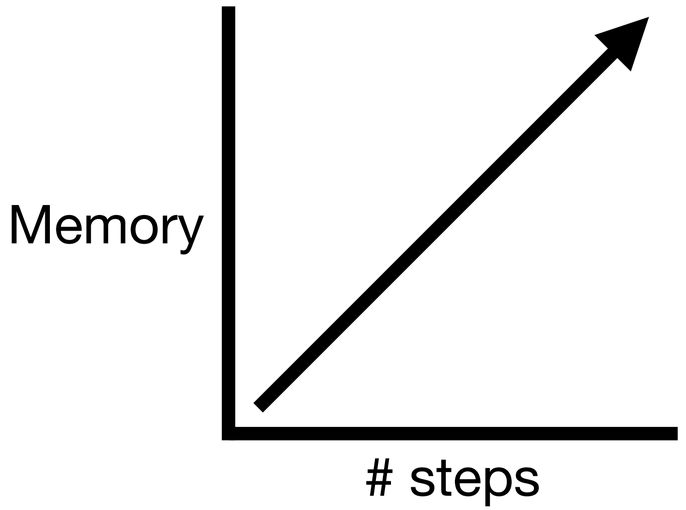
#1: Computational tractability



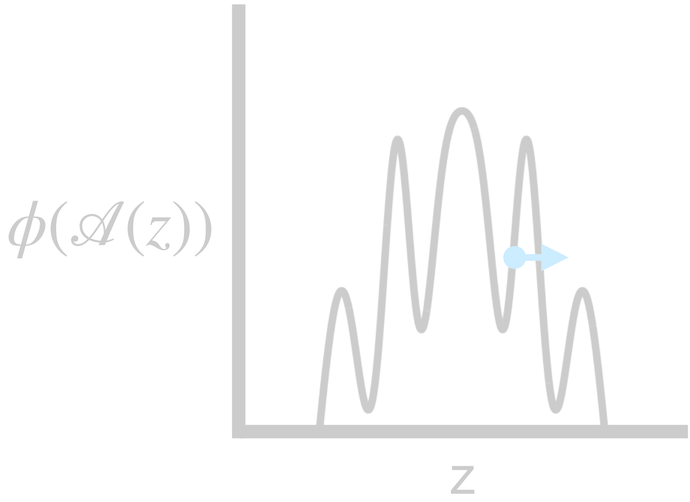
#2: Optimization (non)smoothness

Metagradient roadmap: Two challenges

Two barriers to optimizing with the metagradient $\nabla_{\mathbf{z}}\phi(\mathscr{A}(\mathbf{z}))$



#1: Computational tractability



#2: Optimization (non)smoothness

Problem #1: Calculating the gradient

Idea: Direct autodiff (e.g., Domke et al., 2012)

Problem #1: Calculating the gradient

Idea: Direct autodiff (e.g., Domke et al., 2012)

Direct approach: Auto-differentiation (AD)

Problem #1: Calculating the gradient

Idea: Direct autodiff (e.g., Domke et al., 2012)

Direct approach: Auto-differentiation (AD)

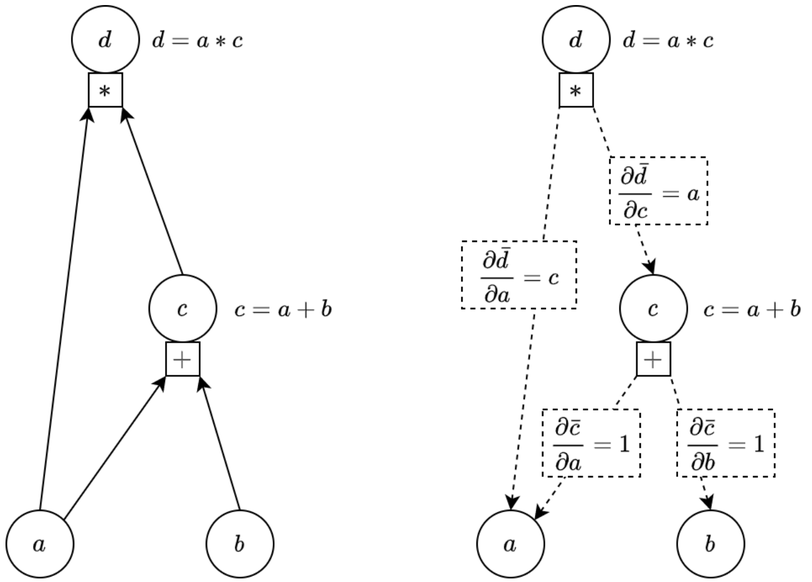
Take derivatives of any function defined by simple operations by saving intermediate products

Problem #1: Calculating the gradient

Idea: Direct autodiff (e.g., Domke et al., 2012)

Direct approach: Auto-differentiation (AD)

Take derivatives of any function defined by simple operations by saving intermediate products



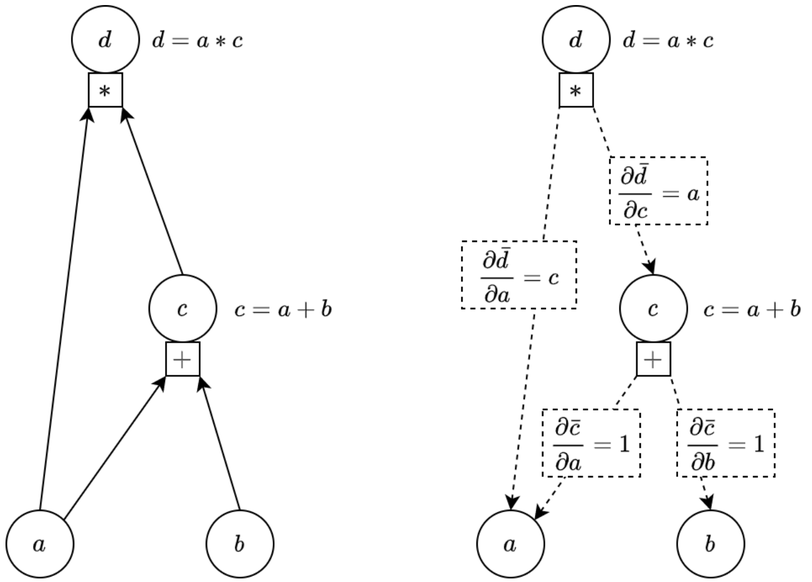
Problem #1: Calculating the gradient

Idea: Direct autodiff (e.g., Domke et al., 2012)

Direct approach: Auto-differentiation (AD)

Take derivatives of any function defined by simple operations by saving intermediate products

Key insight: Training an ML model is just a (very, very, very long) sequence of operations! Maybe we can just backprop through it?



Problem #1: Calculating the gradient

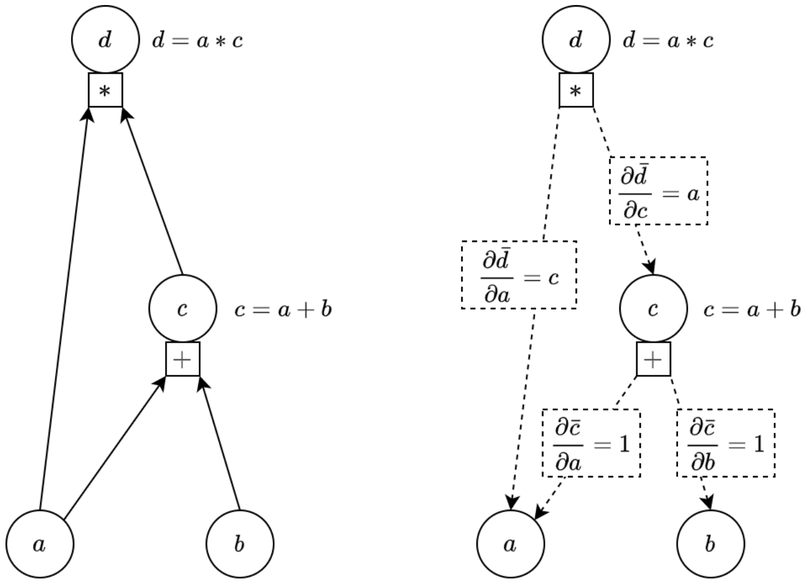
Idea: Direct autodiff (e.g., Domke et al., 2012)

Direct approach: Auto-differentiation (AD)

Take derivatives of any function defined by simple operations by saving intermediate products

Key insight: Training an ML model is just a (very, very, very long) sequence of operations! Maybe we can just backprop through it?

Problem: Too many intermediate products!



Problem #1: Calculating the gradient

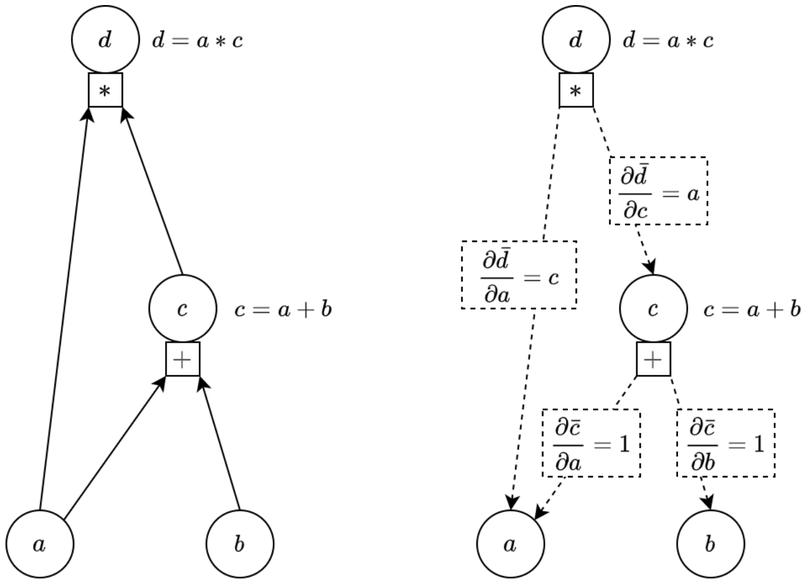
Idea: Direct autodiff (e.g., Domke et al., 2012)

Direct approach: Auto-differentiation (AD)

Take derivatives of any function defined by simple operations by saving intermediate products

Key insight: Training an ML model is just a (very, very, very long) sequence of operations! Maybe we can just backprop through it?

Problem: Too many intermediate products!



We will think of AD as a black box that can compute gradients of a many $\rightarrow$ one compute graph $O(\#\text{ edges})$ memory

Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

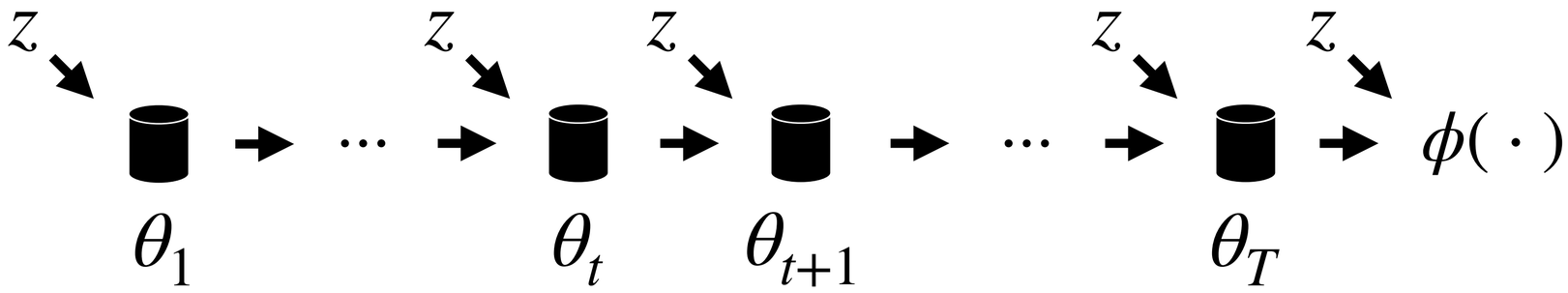
Naive approach: AD on full model training - too many "edges" on compute graph to store!

Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

Naive approach: AD on full model training - too many "edges" on compute graph to store!

Goal: $\frac{d\phi(\mathcal{A}(z))}{dz}$ from

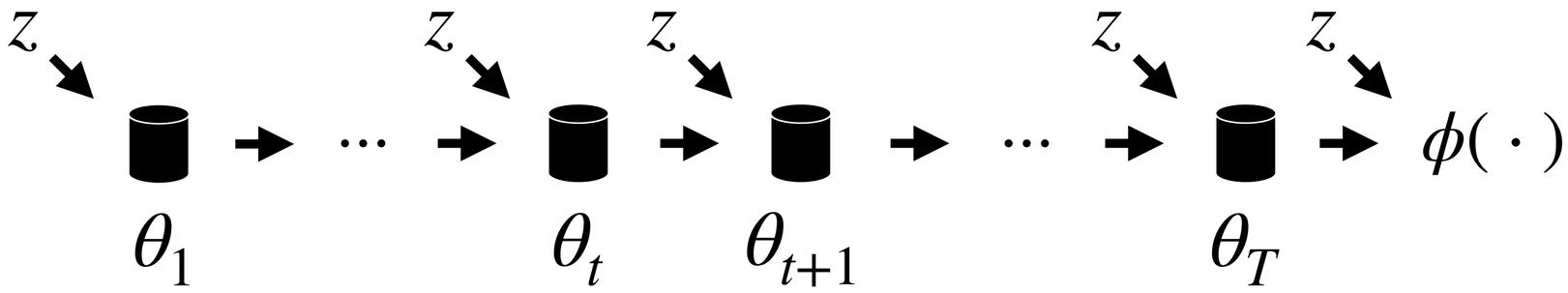


Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

Naive approach: AD on full model training - too many "edges" on compute graph to store!

Goal: $\frac{d\phi(\mathcal{A}(z))}{dz}$ from



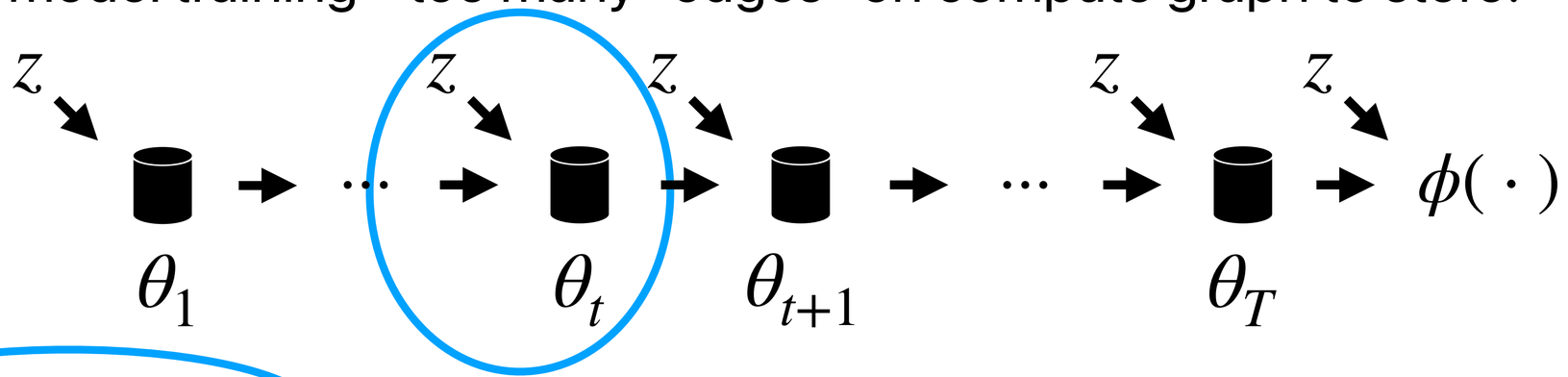
Stepwise AD insight: step t derivatives depend only on step $t+1$ derivatives, step t state

Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

Naive approach: AD on full model training - too many "edges" on compute graph to store!

Goal: $\frac{d\phi(\mathcal{A}(z))}{dz}$ from



Stepwise AD insight: step t derivatives depend only on step $t+1$ derivatives, step t state

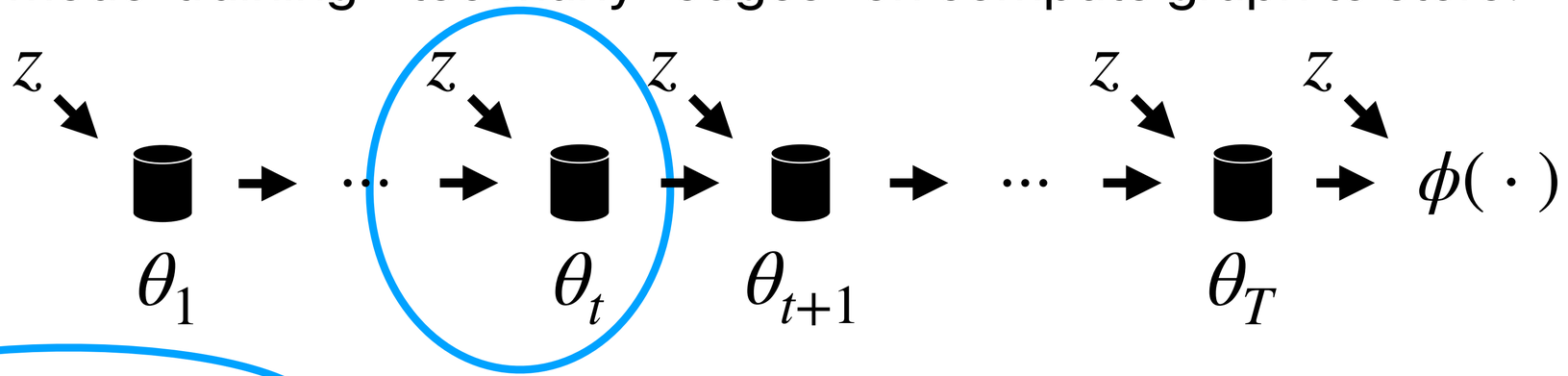
$$\frac{df(z)}{dz} = \frac{df(z)}{d\theta_{t+1}} \cdot \frac{d\theta_{t+1}}{dz} \qquad \frac{df(z)}{d\theta_t} = \frac{df(z)}{d\theta_{t+1}} \cdot \frac{d\theta_{t+1}}{d\theta_t}$$

Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

Naive approach: AD on full model training - too many "edges" on compute graph to store!

Goal: $\frac{d\phi(\mathcal{A}(z))}{dz}$ from



Stepwise AD insight: step t derivatives depend only on step $t+1$ derivatives, step t state

$$\frac{df(z)}{dz} = \frac{df(z)}{d\theta_{t+1}} \cdot \underbrace{\frac{d\theta_{t+1}}{dz}}_{\text{One autograd call}}$$

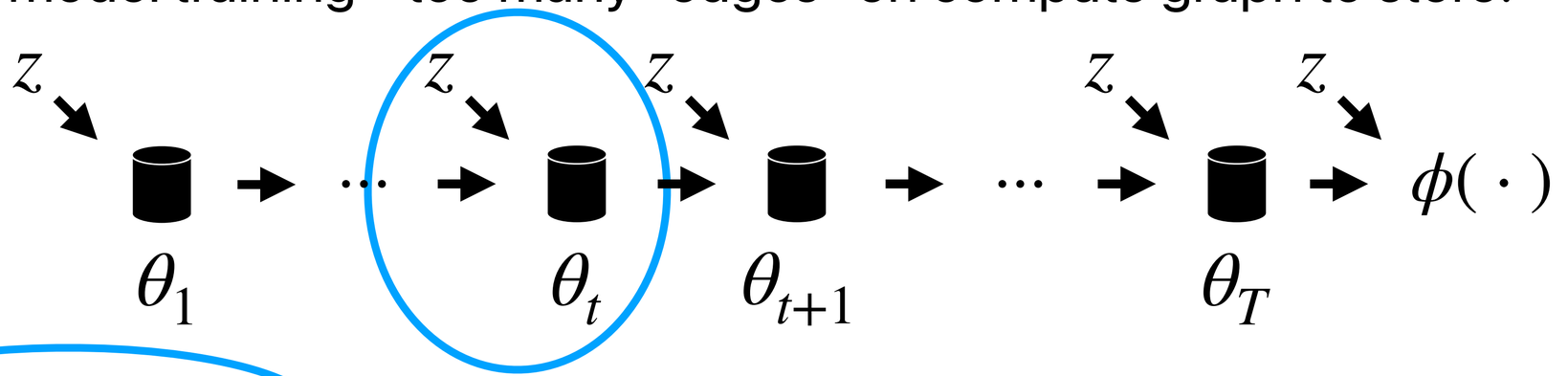
$$\frac{df(z)}{d\theta_t} = \frac{df(z)}{d\theta_{t+1}} \cdot \underbrace{\frac{d\theta_{t+1}}{d\theta_t}}_{\text{One autograd call}}$$

Problem #1: Calculating the gradient

Stepwise AD (Franceschi et al. 2017; Hara et al. 2019; many others)

Naive approach: AD on full model training - too many "edges" on compute graph to store!

Goal: $\frac{d\phi(\mathcal{A}(z))}{dz}$ from



Stepwise AD insight: step t derivatives depend only on step $t+1$ derivatives, step t state

$$\frac{df(z)}{dz} = \frac{df(z)}{d\theta_{t+1}} \cdot \frac{d\theta_{t+1}}{dz}$$

One autograd call

$$\frac{df(z)}{d\theta_t} = \frac{df(z)}{d\theta_{t+1}} \cdot \frac{d\theta_{t+1}}{d\theta_t}$$

One autograd call

Algorithm: save all states $\{\theta_i\}_{i=1}^{T}$, do autograd one step at a time from iterate T down to 1

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

- Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
  (Maclaurin Duvenaud & Adams, 2015)
- Deterministic training allows us to “replay” from a previous state to rematerialize
- Design a data structure that allows us to trade off time and space:

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Step-wise AD: Save every state, traverse states in reverse

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

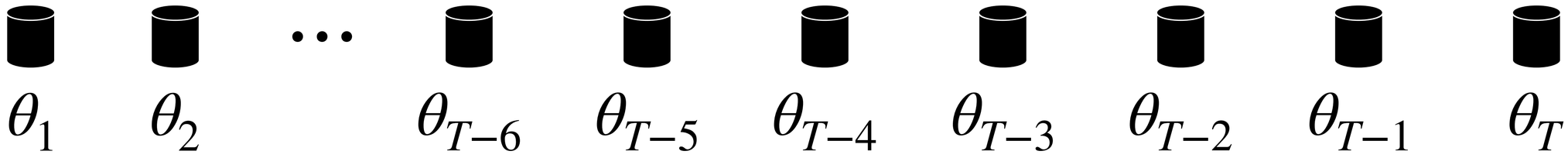
Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Step-wise AD: Save every state, traverse states in reverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

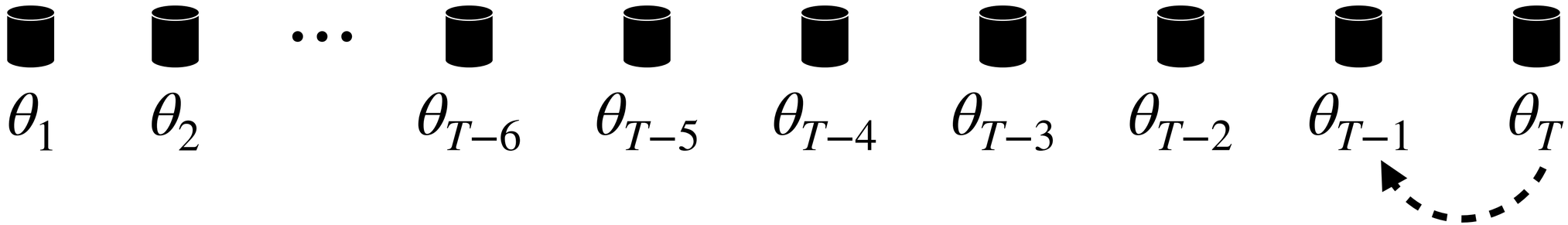
Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Step-wise AD: Save every state, traverse states in reverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

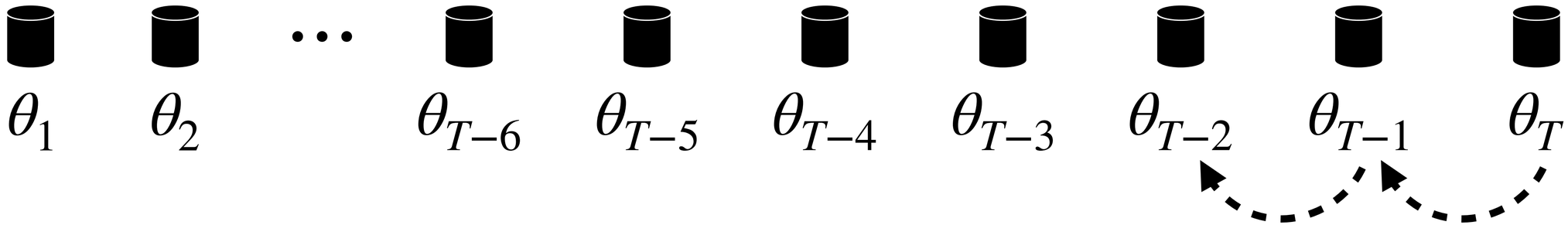
Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Step-wise AD: Save every state, traverse states in reverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

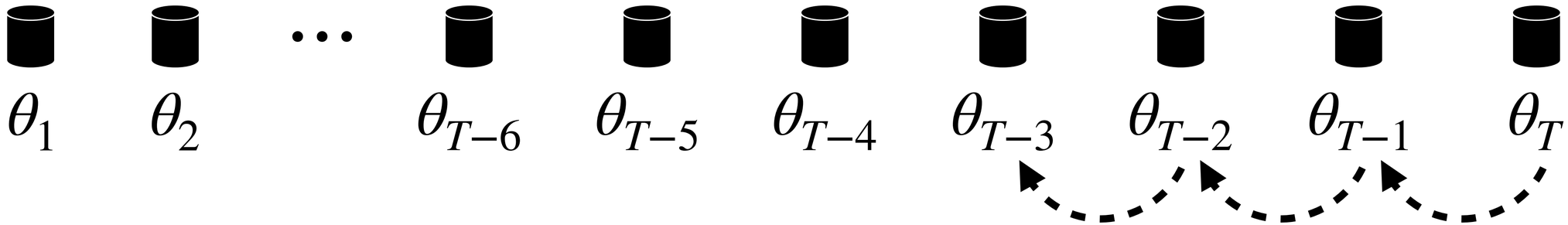
A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Step-wise AD: Save every state, traverse states in reverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

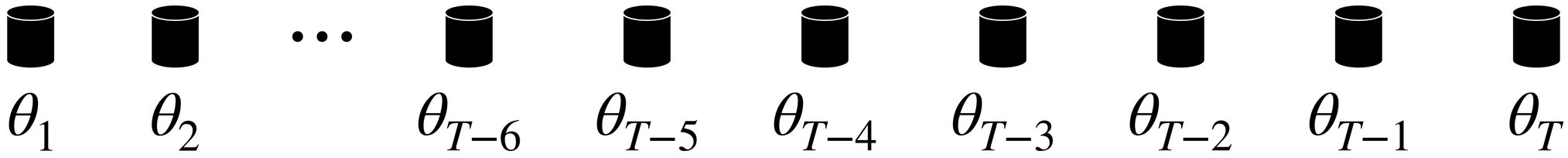
A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

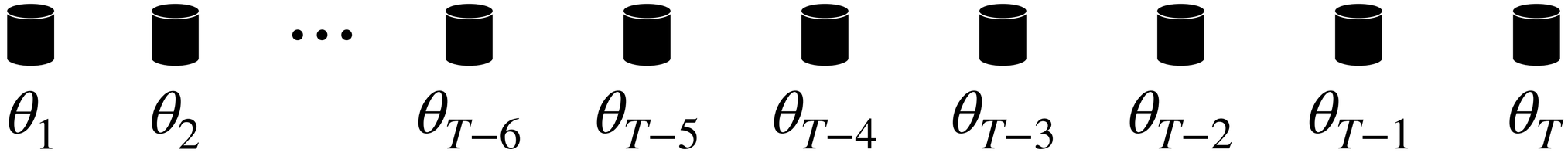
A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #1: Save every k-th state, “hop” back to saved states and replay to traverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

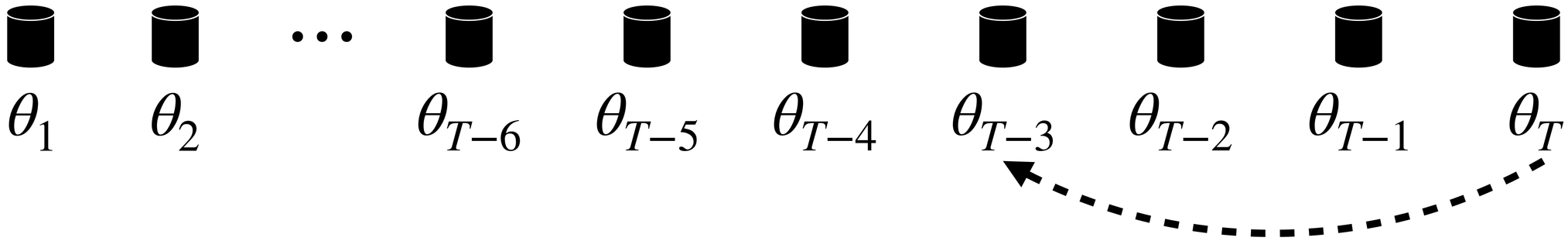
A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #1: Save every k-th state, “hop” back to saved states and replay to traverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

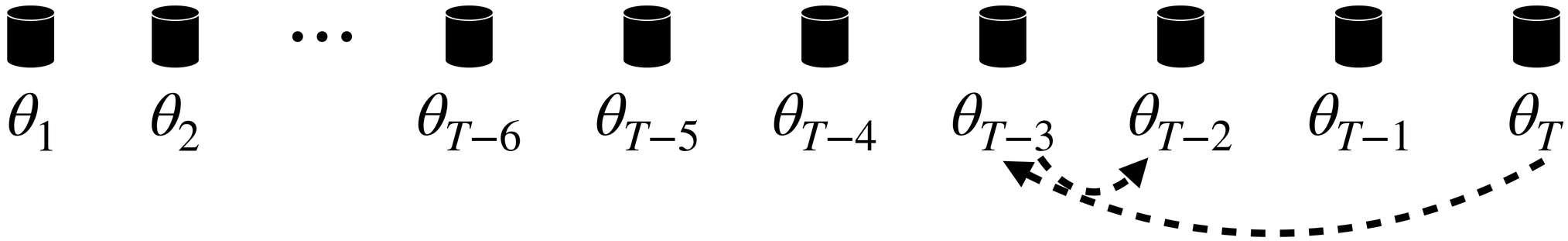
Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #1: Save every k-th state, “hop” back to saved states and replay to traverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

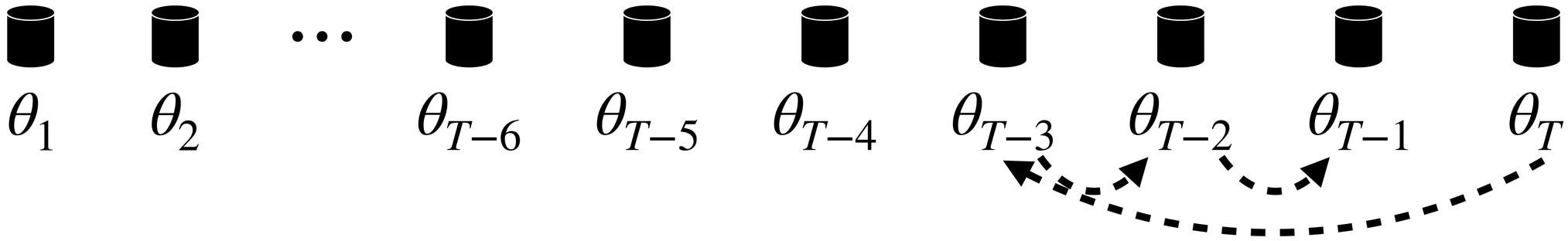
A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #1: Save every k-th state, “hop” back to saved states and replay to traverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

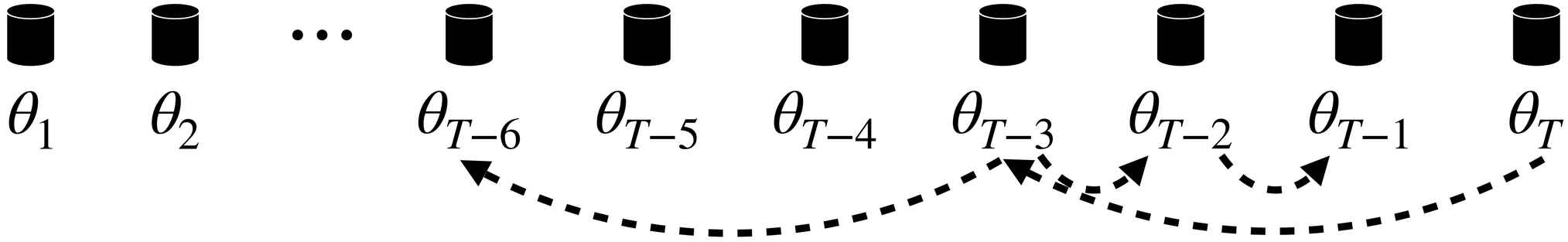
A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #1: Save every k-th state, “hop” back to saved states and replay to traverse



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order

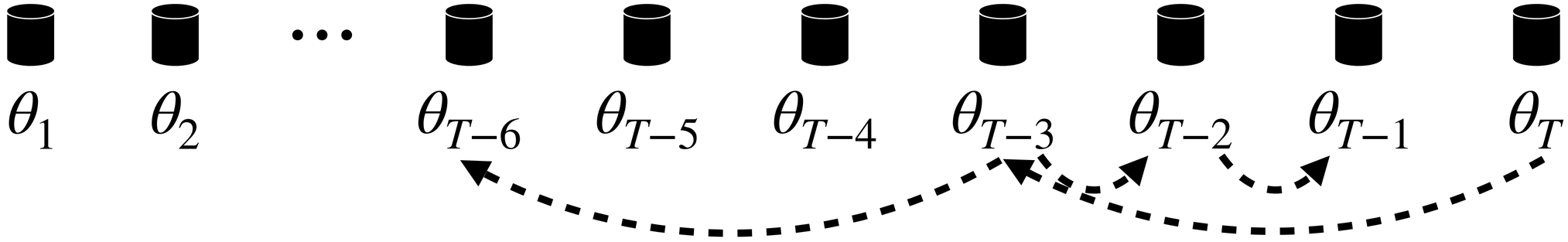
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to "replay" from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #1: Save every k-th state, "hop" back to saved states and replay to traverse

($T/k + k$ space $\rightarrow 2\sqrt{T}$ space, T extra training steps)



Problem #1: Calculating the gradient

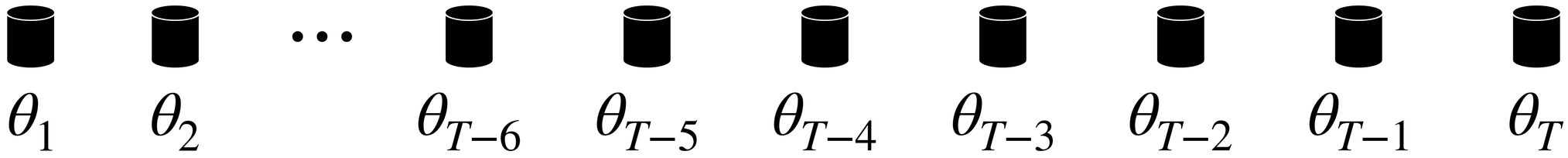
REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY (Recursive rematerialization for metagradients)

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to “replay” from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

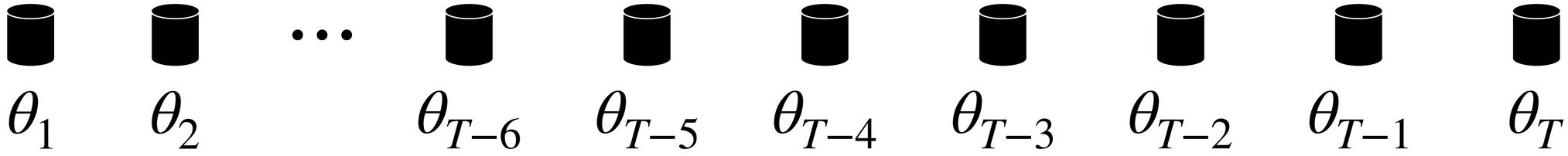
A simple modification: REPLAY (Recursive rematerialization for metagradients)

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to "replay" from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #2: Recursively apply same idea across k training segments



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

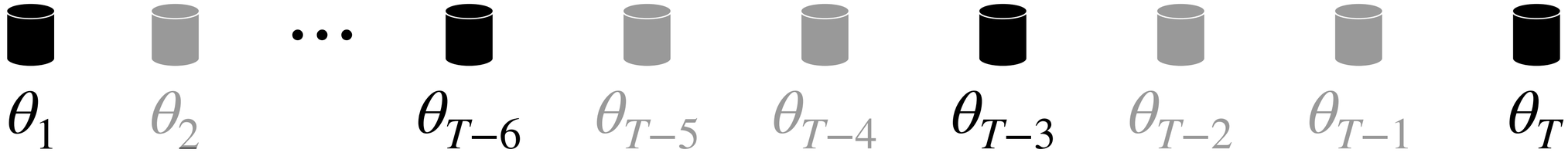
A simple modification: REPLAY (Recursive rematerialization for metagradients)

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to "replay" from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #2: Recursively apply same idea across k training segments



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

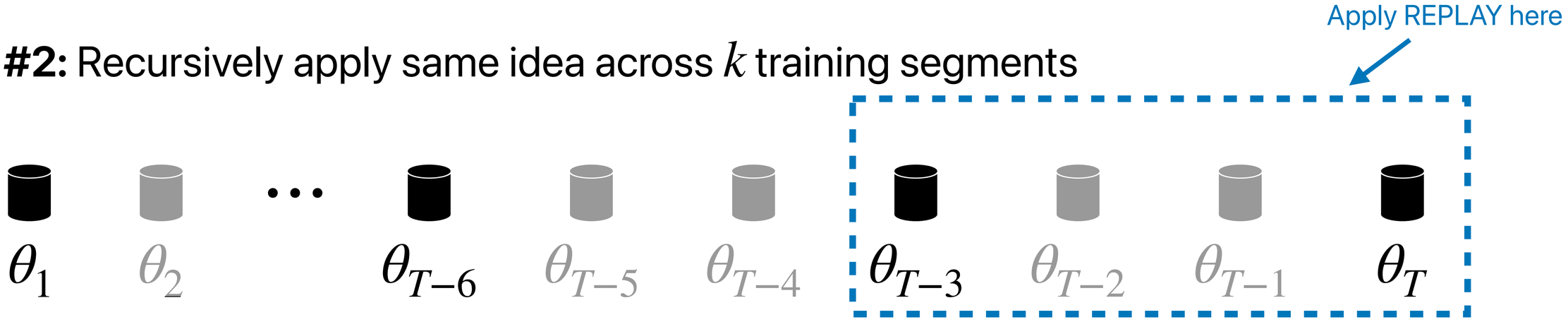
A simple modification: REPLAY (Recursive rematerialization for metagradients)

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to "replay" from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #2: Recursively apply same idea across k training segments



Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

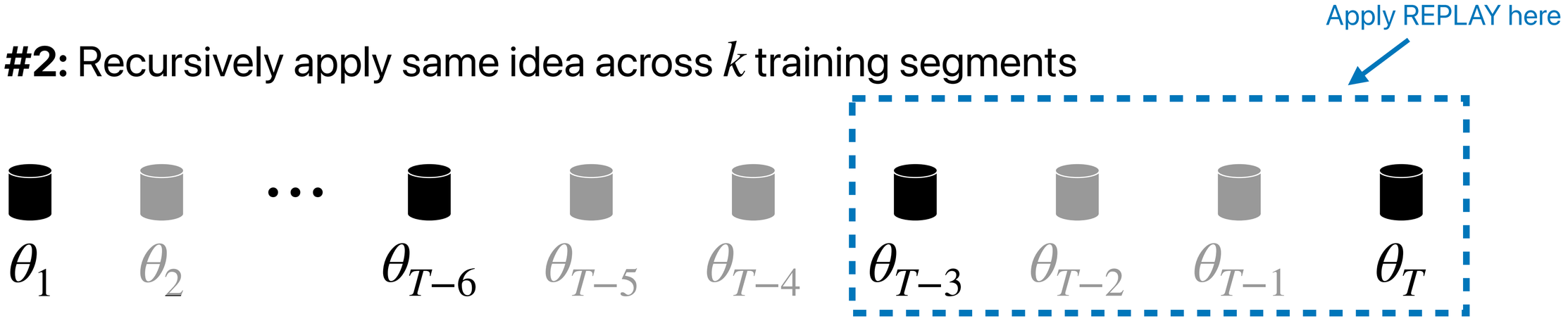
A simple modification: REPLAY (Recursive rematerialization for metagradients)

Key primitive of step-wise AD: **traversing** the states $\theta_T \ldots \theta_1$ in reverse order
(Maclaurin Duvenaud & Adams, 2015)

Deterministic training allows us to "replay" from a previous state to rematerialize

Design a data structure that allows us to trade off time and space:

Idea #2: Recursively apply same idea across k training segments



(For any k, uses $k \log_k(T)$ space, $T \log_k(T)$ extra training steps)

Problem #1: Calculating the gradient

REPLAY: Efficiently calculating metagradients at scale

A simple modification: REPLAY (Recursive rematerialization for metagradients)

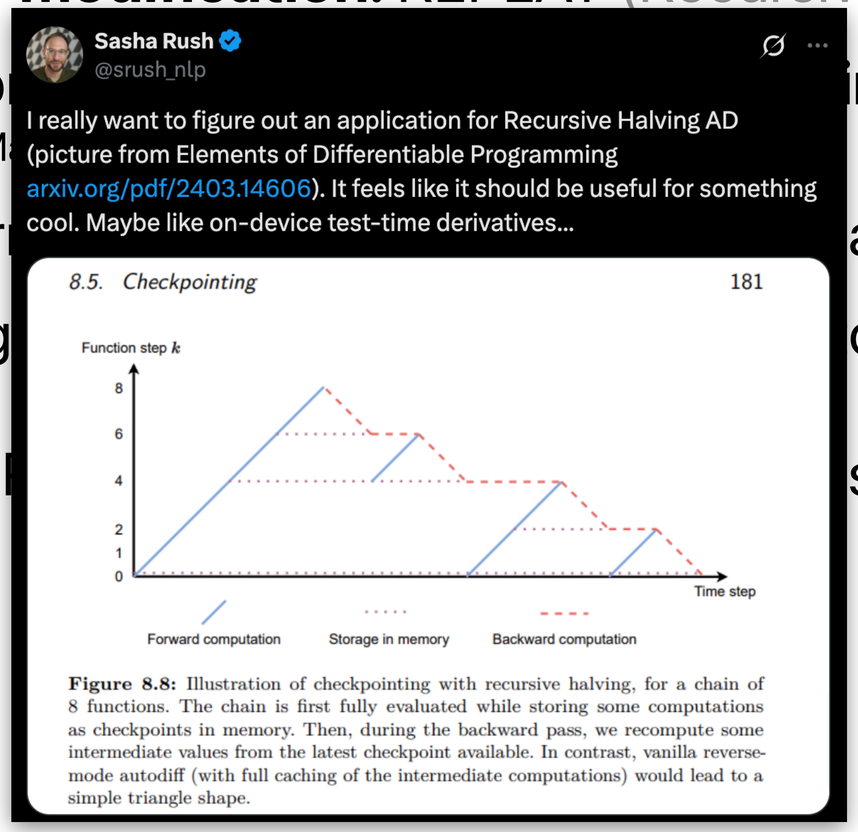
Key p**ing** the states $\theta_T \ldots \theta_1$ in reverse order

(M

Deter**ay"** from a previous state to rematerialize

Desig**o** trade off time and space:

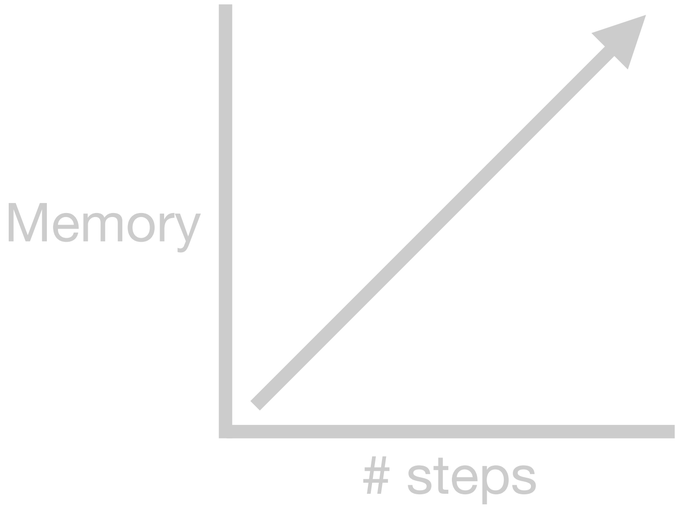
Idea #2: **ss** k training segments



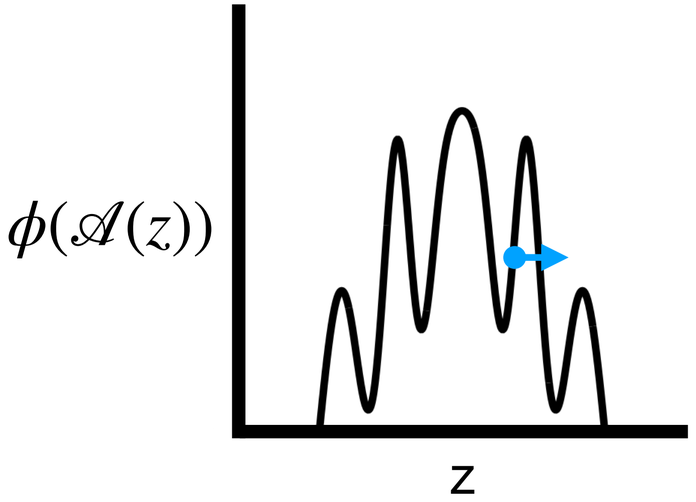
(For any k, uses $k \log_k(T)$ space, $T \log_k(T)$ extra training steps)

Metagradient roadmap: Two challenges

Two barriers to optimizing with the metagradient $\nabla_z \phi(\mathcal{A}(z))$



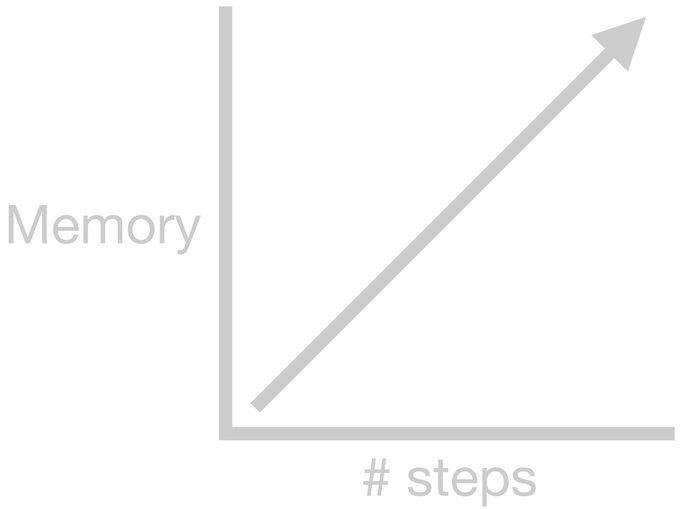
#1: Computational tractability



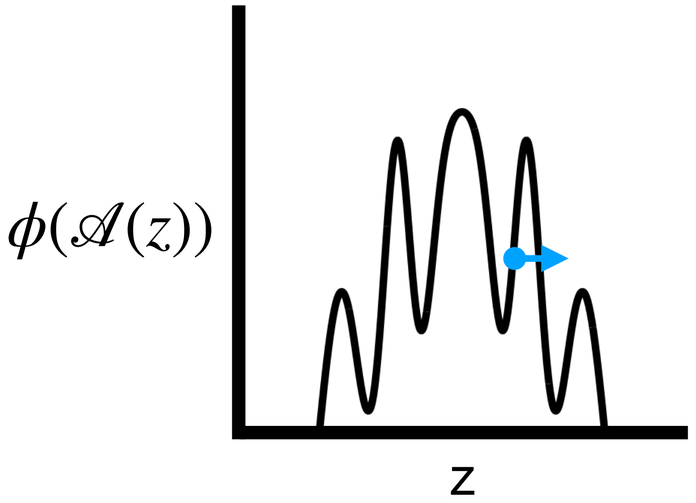
#2: Optimization (non)smoothness

Metagradient roadmap: Two challenges

Two barriers to optimizing with the metagradient $\nabla_z \phi(\mathcal{A}(z))$



#1: Computational tractability



#2: Optimization (non)smoothness

Problem #2: (Lack of) Smoothness

Are our gradients useful?

Problem #2: (Lack of) Smoothness

Are our gradients useful?

A gradient is only "useful" if it locally predicts function behavior

> In optimization terms, we want $\phi(\mathcal{A}(\mathbf{z}))$ to be "smooth"

Problem #2: (Lack of) Smoothness

Are our gradients useful?

A gradient is only "useful" if it locally predicts function behavior

> In optimization terms, we want $\phi(\mathcal{A}(\mathbf{z}))$ to be "smooth"

Given the complexity of $\mathcal{A}(\mathbf{z})$ for large-scale models, cannot take this for granted!

Problem #2: (Lack of) Smoothness

Are our gradients useful?

A gradient is only "useful" if it locally predicts function behavior

> In optimization terms, we want $\phi(\mathcal{A}(\mathbf{z}))$ to be "smooth"

Given the complexity of $\mathcal{A}(\mathbf{z})$ for large-scale models, cannot take this for granted!

Indeed:

Problem #2: (Lack of) Smoothness

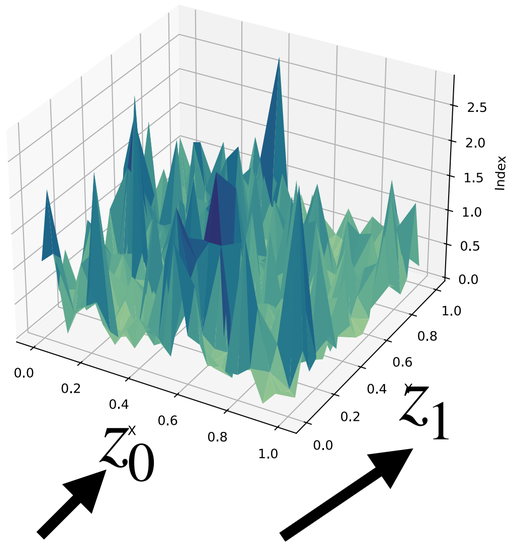
Are our gradients useful?

A gradient is only "useful" if it locally predicts function behavior

In optimization terms, we want $\phi(\mathcal{A}(\mathbf{z}))$ to be "smooth"

Given the complexity of $\mathcal{A}(\mathbf{z})$ for large-scale models, cannot take this for granted!

Indeed: $\phi(\mathcal{A}(\mathbf{z}))$



Small changes in training data pixels!

Problem #2: (Lack of) Smoothness

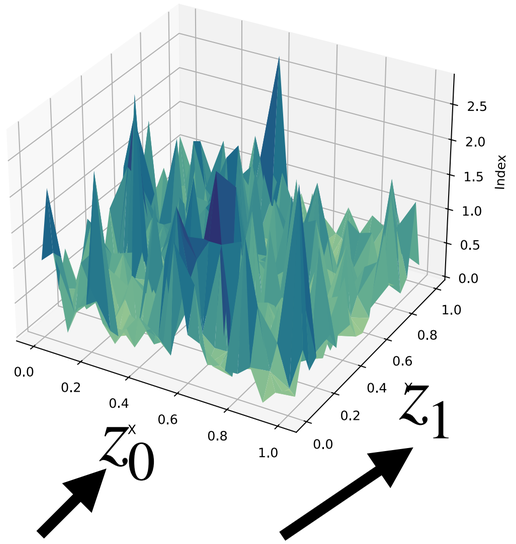
Are our gradients useful?

A gradient is only "useful" if it locally predicts function behavior

In optimization terms, we want $\phi(\mathcal{A}(\mathbf{z}))$ to be "smooth"

Given the complexity of $\mathcal{A}(\mathbf{z})$ for large-scale models, cannot take this for granted!

Indeed: $\phi(\mathcal{A}(\mathbf{z}))$



...standard model training is very "non-smooth!"

Small changes in training data pixels!

Problem #2: (Lack of) Smoothness

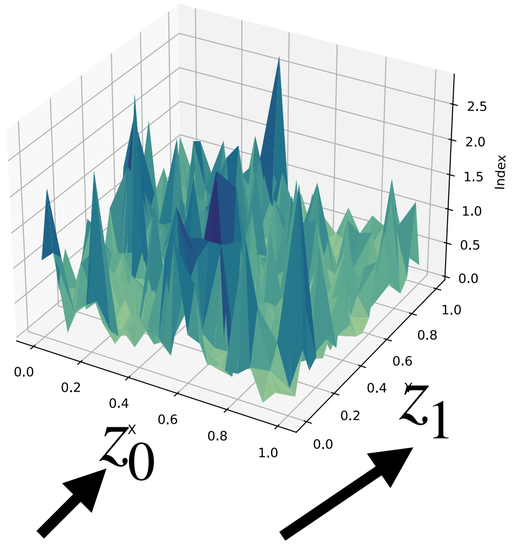
Are our gradients useful?

A gradient is only "useful" if it locally predicts function behavior

In optimization terms, we want $\phi(\mathcal{A}(\mathbf{z}))$ to be "smooth"

Given the complexity of $\mathcal{A}(\mathbf{z})$ for large-scale models, cannot take this for granted!

Indeed: $\phi(\mathcal{A}(\mathbf{z}))$



Small changes in training data pixels!

...standard model training is very "non-smooth!"

Need to find "smooth" model training routines!

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Desired diagnostic ("Smoothness"): Approximate Taylor predictiveness

$$\approx \max_{\Delta \mathbf{w}} \left\| \mathscr{A}(\mathbf{w} + \mathbf{\Delta w}) - \mathscr{A}(\mathbf{w}) - \frac{d\mathscr{A}(\mathbf{w})}{d\mathbf{w}} \cdot \mathbf{\Delta w} \right\|$$

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Desired diagnostic ("Smoothness"): Approximate Taylor predictiveness

$$\approx \max_{\Delta \mathbf{w}} \left\| \mathscr{A}(\mathbf{w} + \mathbf{\Delta w}) - \underbrace{\mathscr{A}(\mathbf{w}) - \frac{d\mathscr{A}(\mathbf{w})}{d\mathbf{w}} \cdot \mathbf{\Delta w}}_{\text{First-order effect of metaparameter changes}} \right\|$$

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Desired diagnostic ("Smoothness"): Approximate Taylor predictiveness

Can't take a max!

$$\approx \max_{\Delta \mathbf{w}} \left\| \mathscr{A}(\mathbf{w} + \Delta\mathbf{w}) - \mathscr{A}(\mathbf{w}) - \frac{d\mathscr{A}(\mathbf{w})}{d\mathbf{w}} \cdot \Delta\mathbf{w} \right\|$$

First-order effect of metaparameter changes

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Desired diagnostic ("Smoothness"): Approximate Taylor predictiveness

Can't take a max!

$$\approx \max_{\Delta \mathbf{w}} \left\| \mathscr{A}(\mathbf{w} + \Delta\mathbf{w}) - \mathscr{A}(\mathbf{w}) - \frac{d\mathscr{A}(\mathbf{w})}{d\mathbf{w}} \cdot \Delta\mathbf{w} \right\|$$

First-order effect of metaparameter changes

Hard to interpret this quantity!

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Desired diagnostic ("Smoothness"): Approximate Taylor predictiveness

Can't take a max!

Don't have full Jacobian!

$$\approx \max_{\Delta \mathbf{w}} \left\| \mathscr{A}(\mathbf{w} + \Delta\mathbf{w}) - \mathscr{A}(\mathbf{w}) - \frac{d\mathscr{A}(\mathbf{w})}{d\mathbf{w}} \cdot \Delta\mathbf{w} \right\|$$

First-order effect of metaparameter changes

Hard to interpret this quantity!

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Goal: design model training routines that *do* have "smooth" optimization landscapes

Problem: need a cheap + accurate measure of "smoothness"

Desired diagnostic ("Smoothness"): Approximate Taylor predictiveness

Can't take a max!

Don't have full Jacobian!

$$\approx \max_{\Delta \mathbf{w}} \left\| \mathscr{A}(\mathbf{w} + \Delta \mathbf{w}) - \mathscr{A}(\mathbf{w}) - \frac{d\mathscr{A}(\mathbf{w})}{d\mathbf{w}} \cdot \Delta \mathbf{w} \right\|$$

First-order effect of metaparameter changes

Hard to interpret this quantity!

We adapt this estimator to a new metric Metasmoothness$(\mathscr{A}) \in [-1,1]$

Our final estimator: feasible, interpretable, and accurate!

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Problem #2: (Lack of) Smoothness

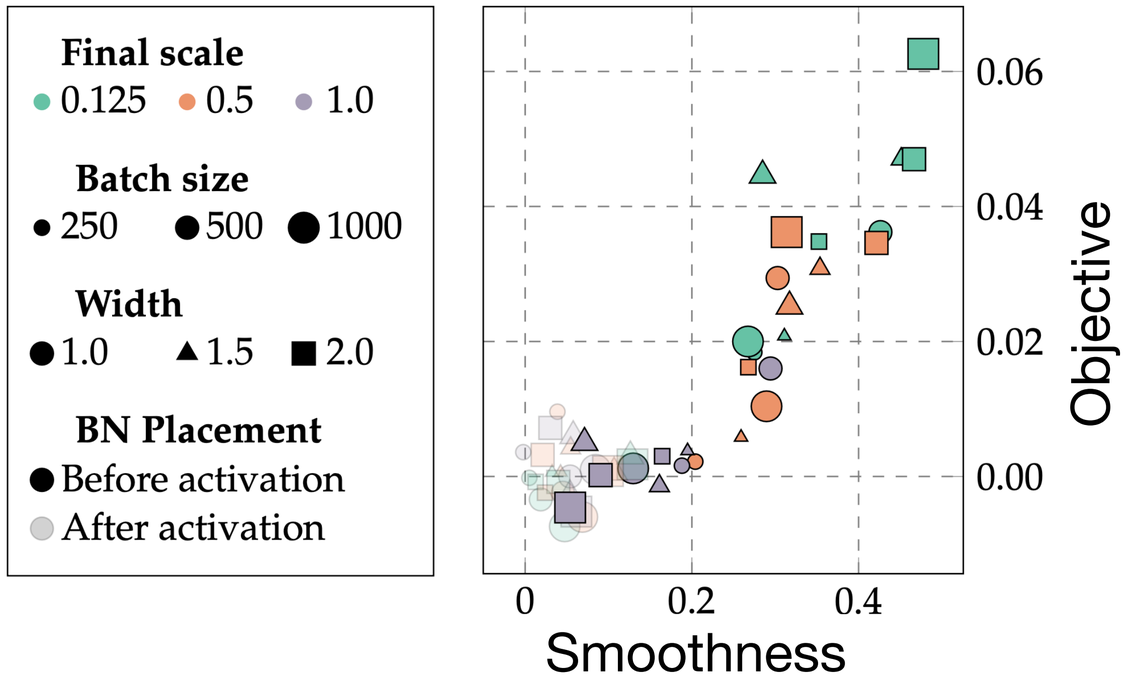
Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

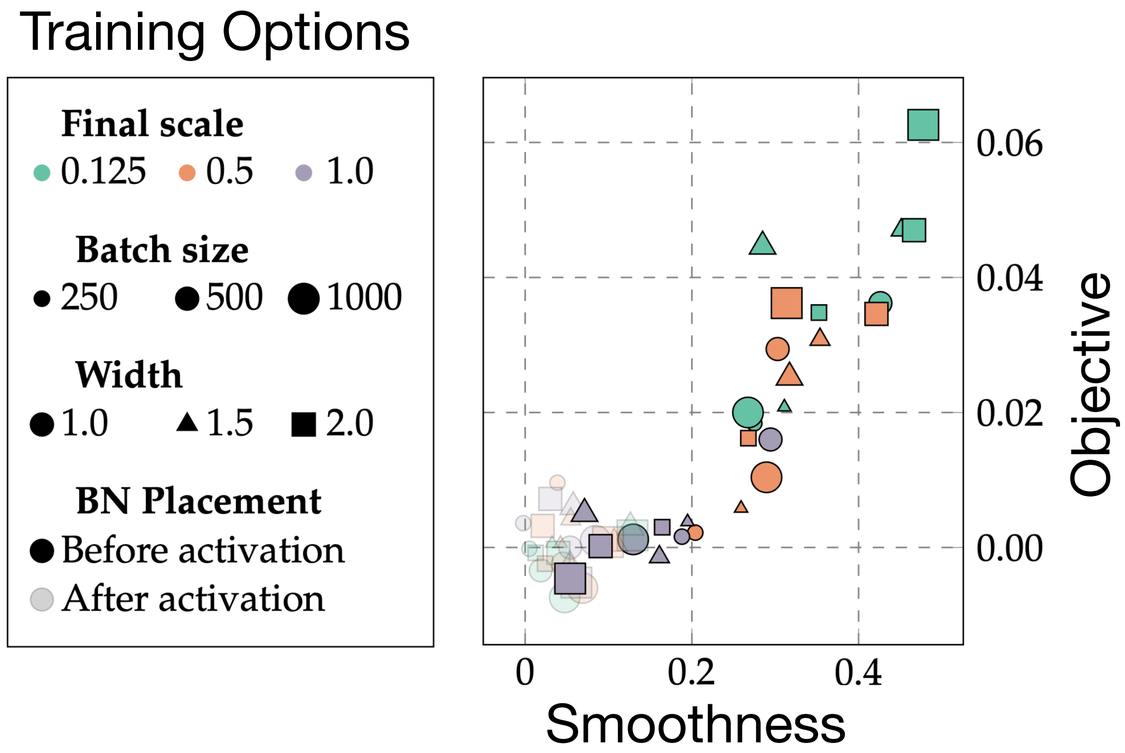
Our metric accurately predicts optimization success in practice!



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

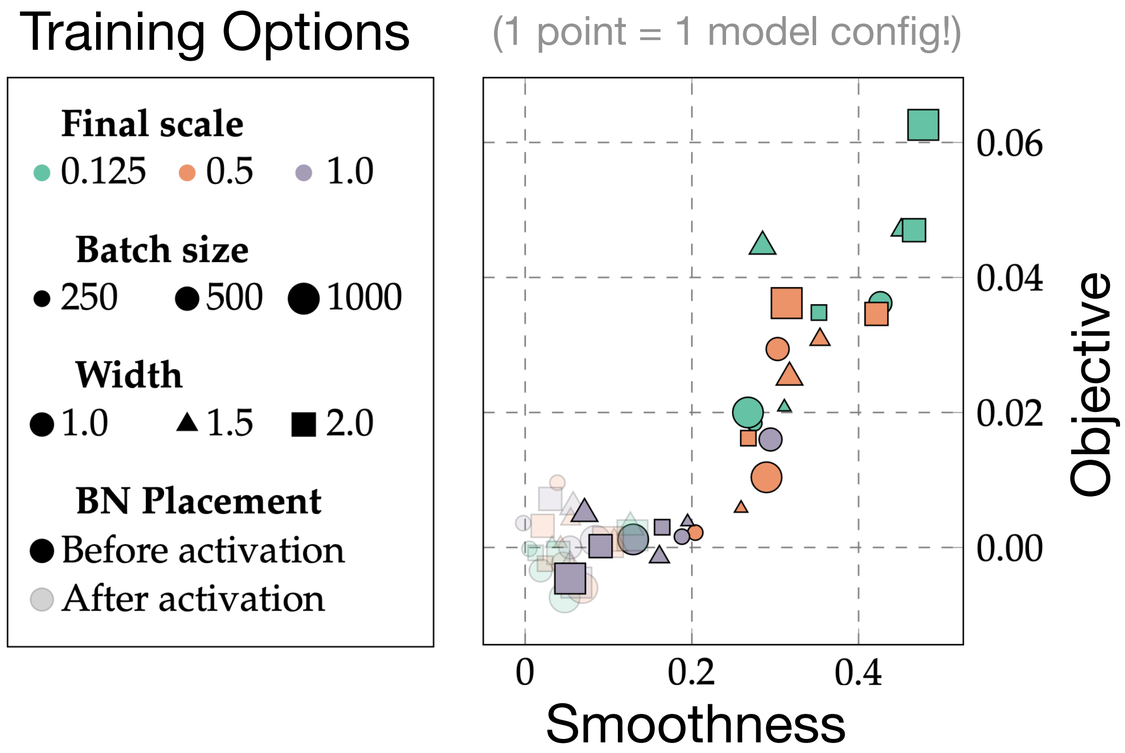
Our metric accurately predicts optimization success in practice!



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

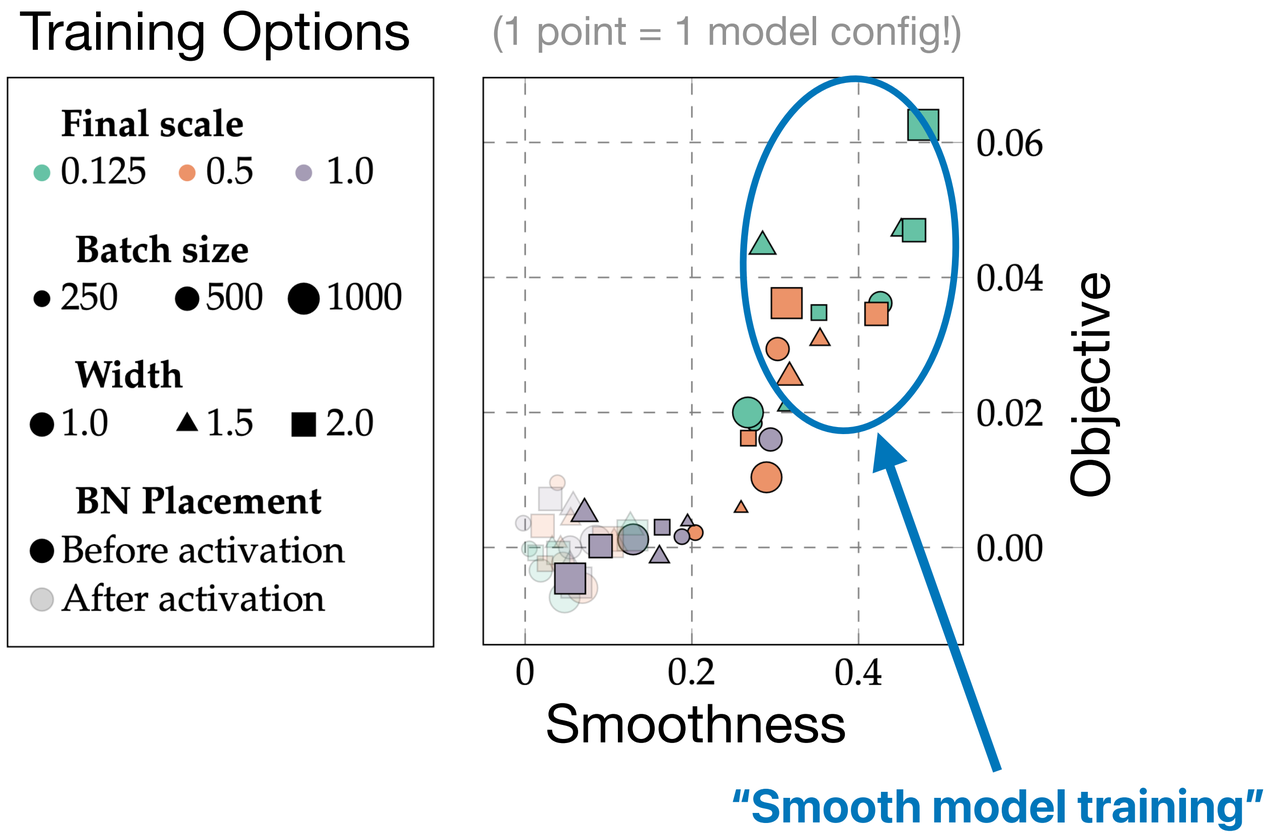
Our metric accurately predicts optimization success in practice!



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

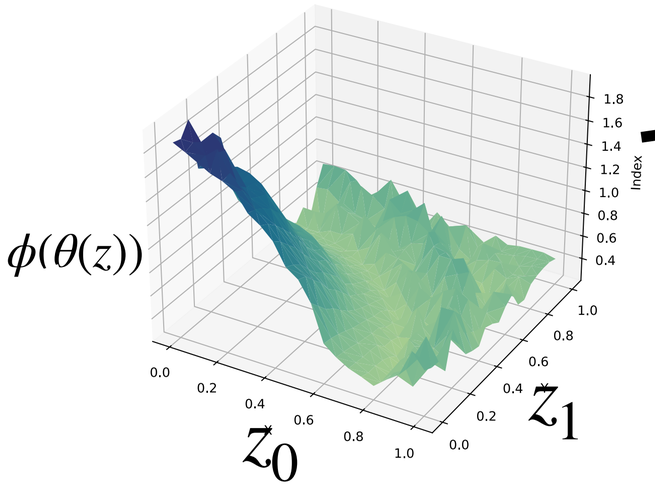
Our metric accurately predicts optimization success in practice!

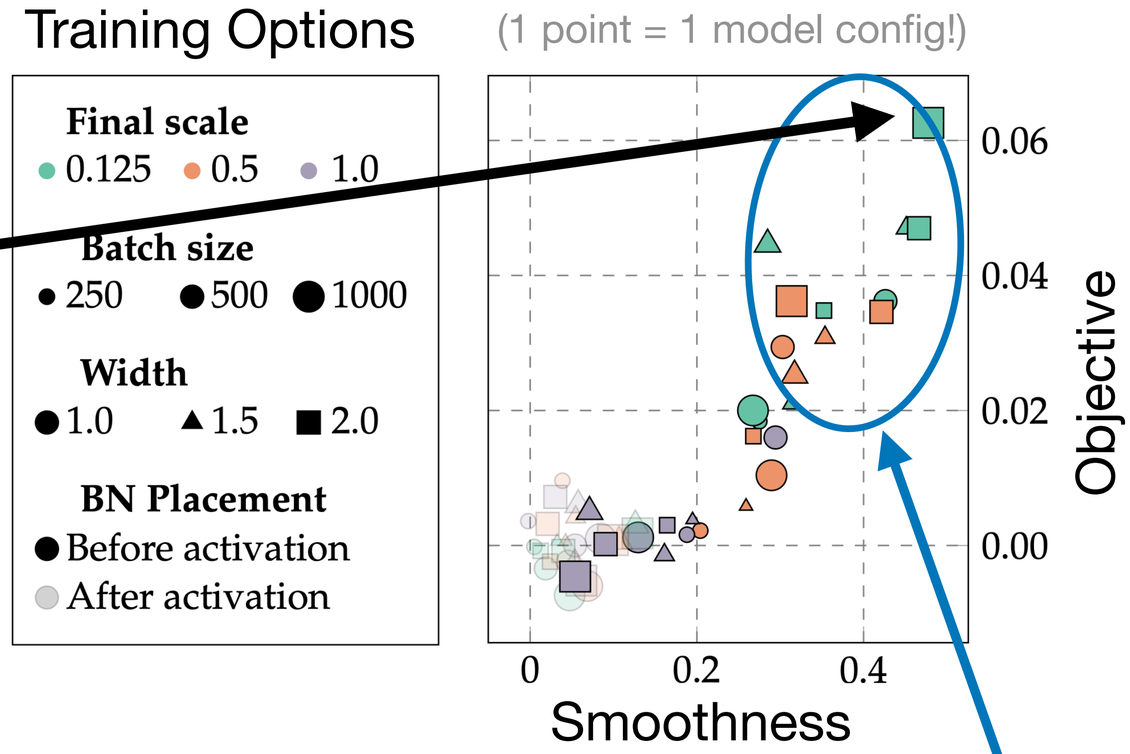


Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

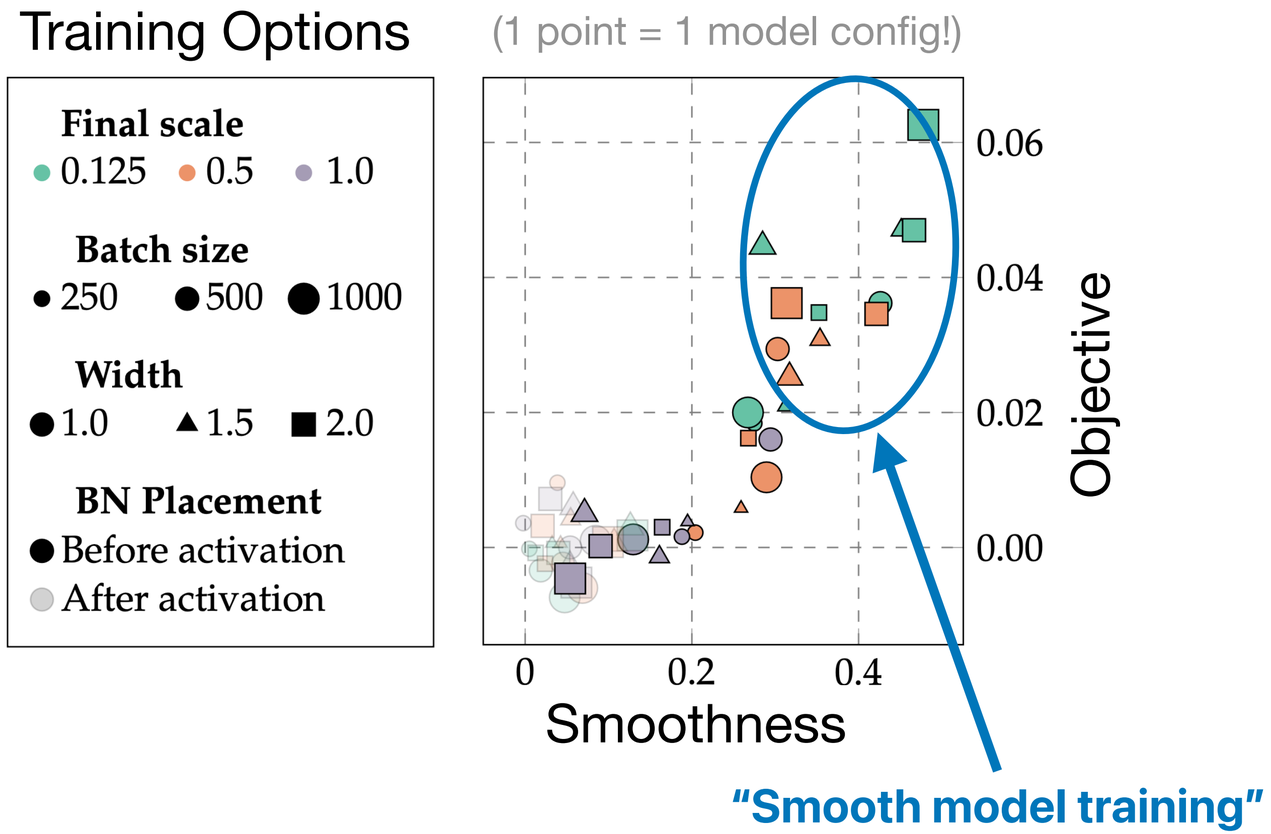




Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

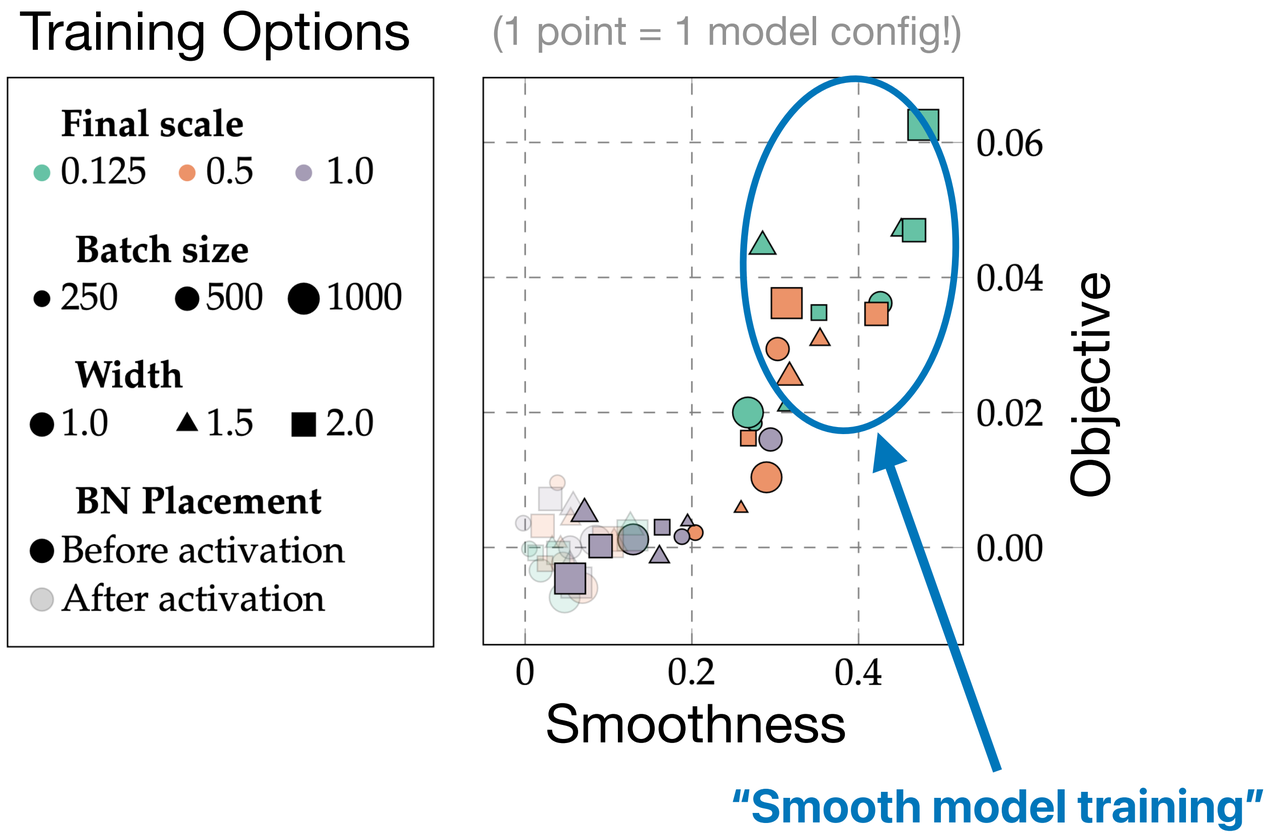


Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:



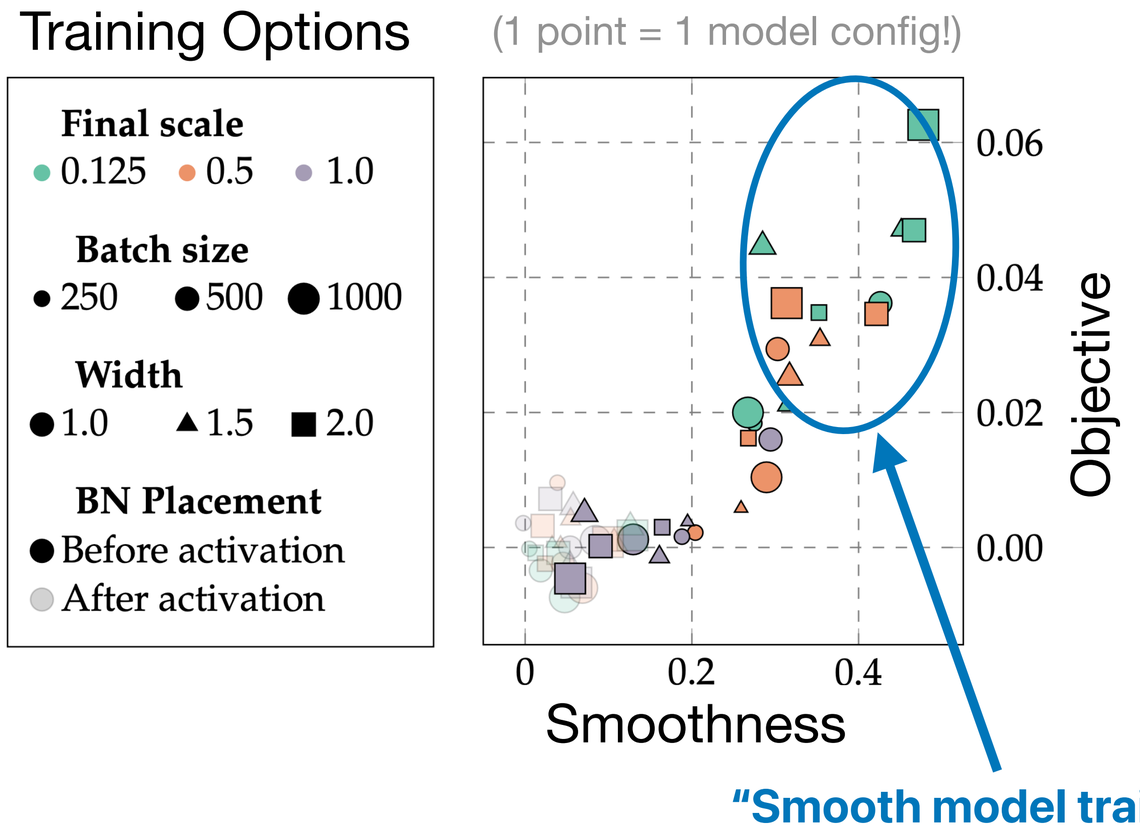
Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

Layer composition, width



Problem #2: (Lack of) Smoothness

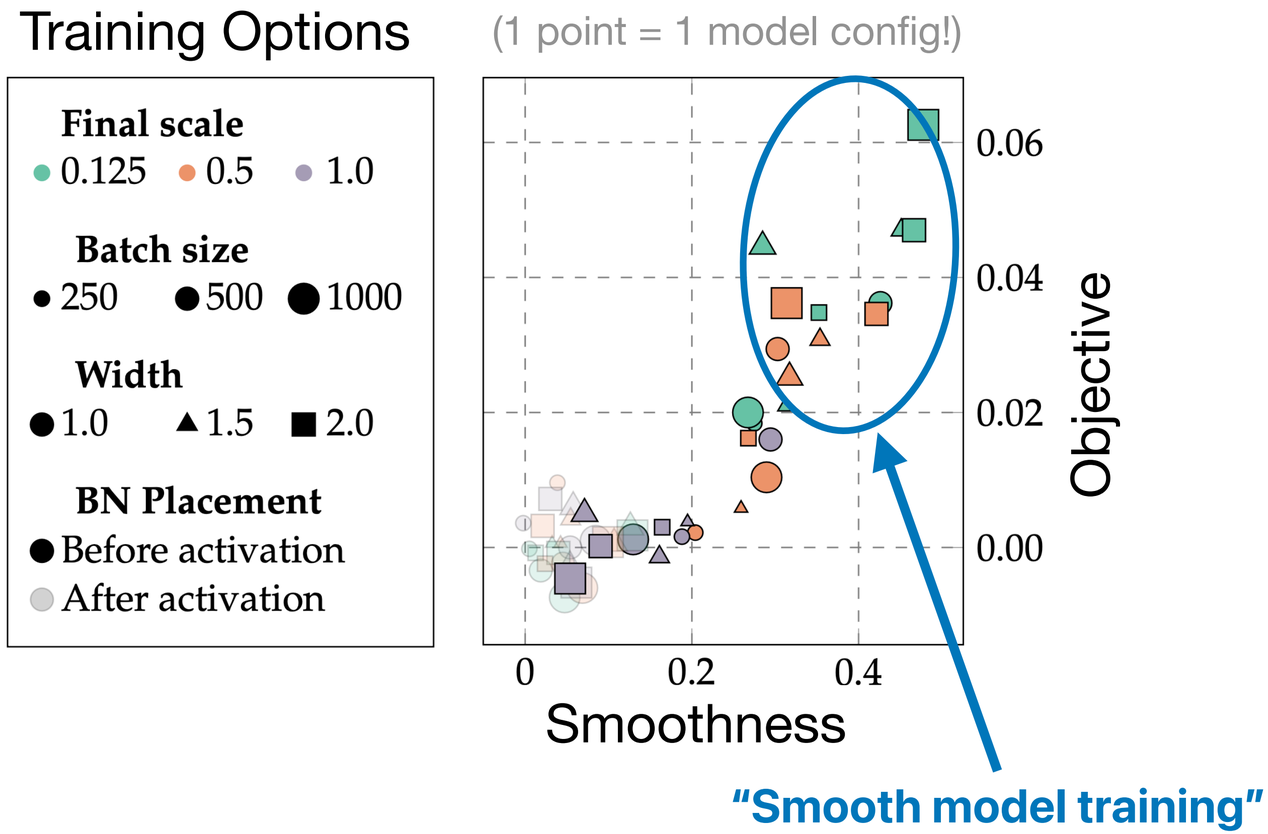
Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

Layer composition, width

Learning rate/weight decay/batch size



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

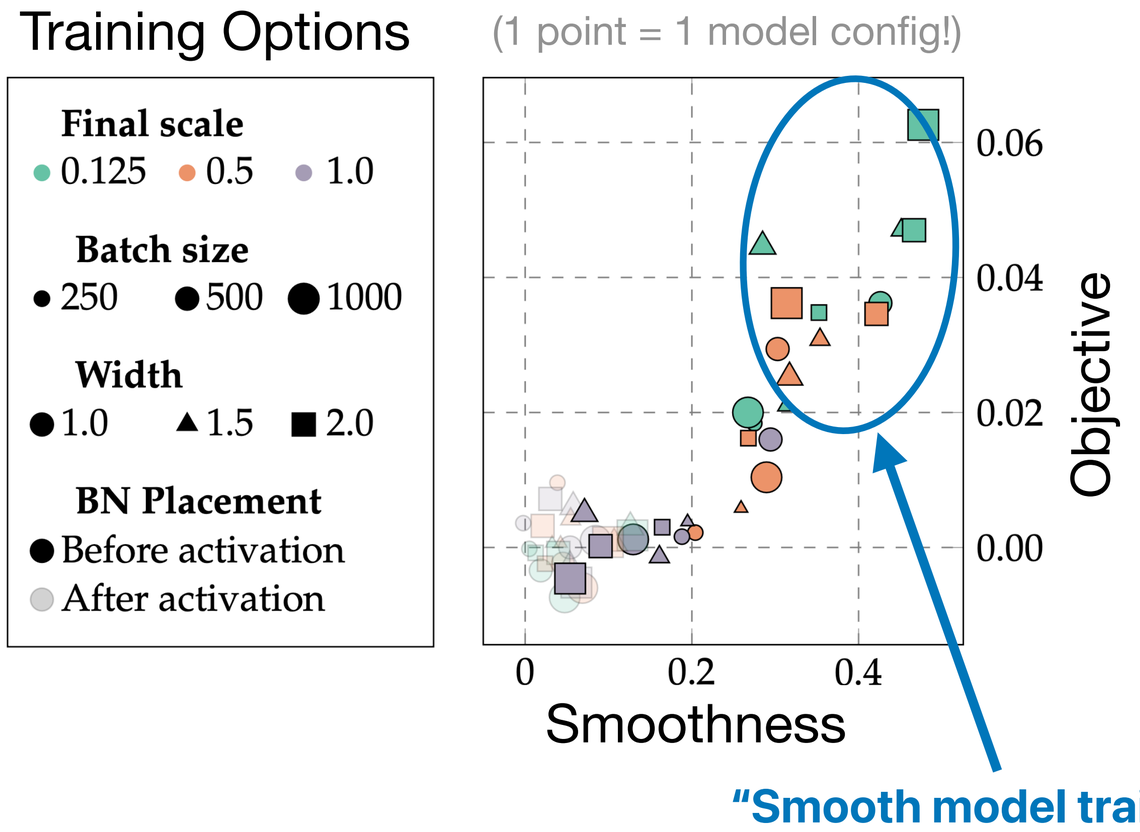
Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

Layer composition, width

Learning rate/weight decay/batch size

Adam ϵ_{root}, momentum/warmup



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

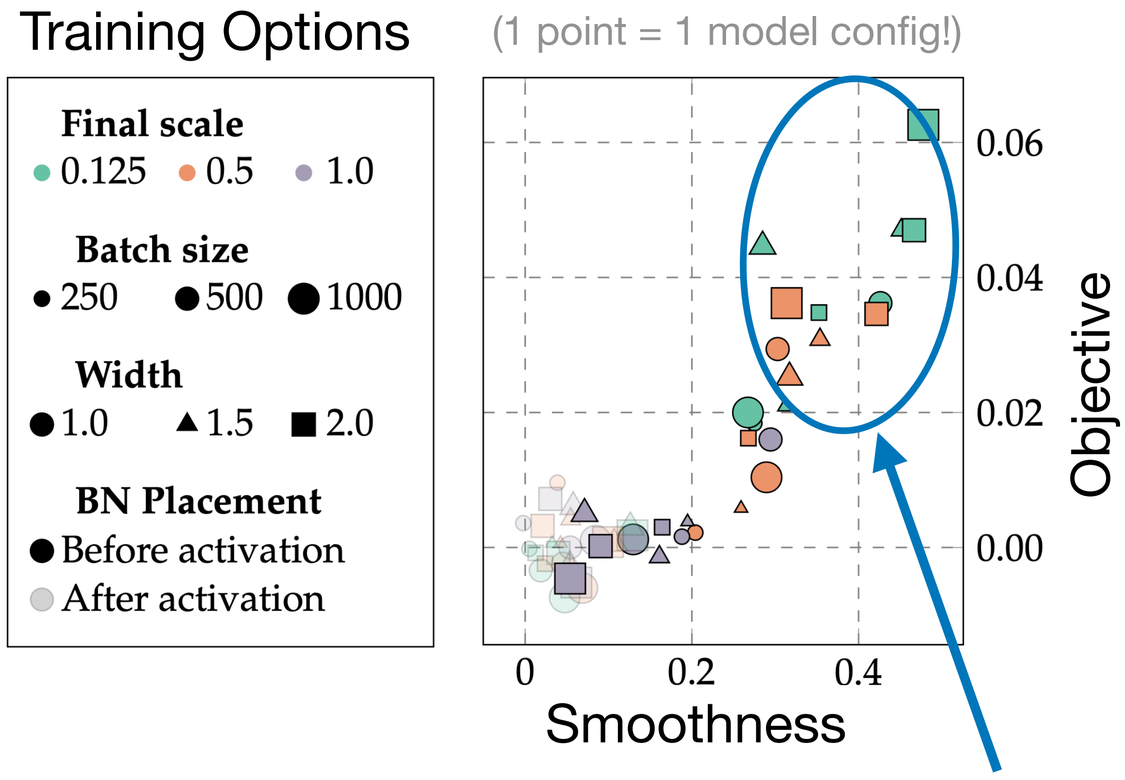
How to train your *smooth* model:

Layer composition, width

Learning rate/weight decay/batch size

Adam ϵ_{root}, momentum/warmup

Normalizer ϵ



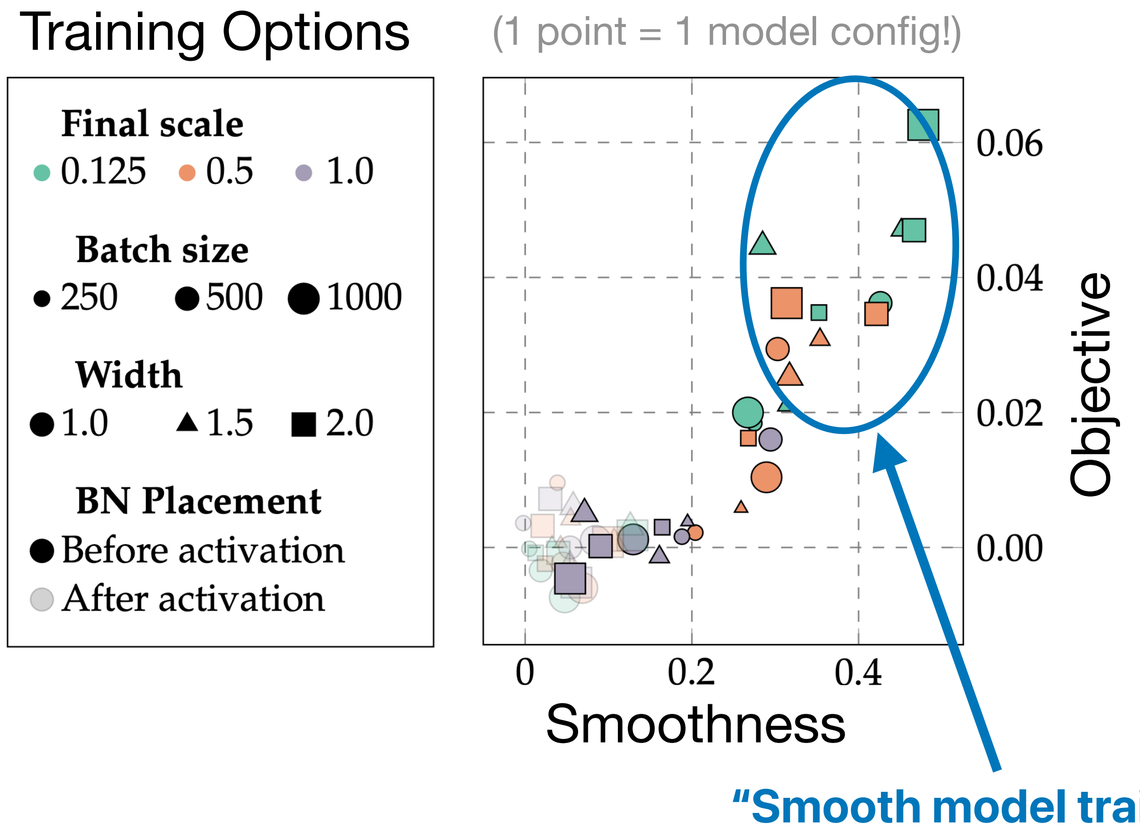
Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

- Layer composition, width
- Learning rate/weight decay/batch size
- Adam ϵ_{root}, momentum/warmup
- Normalizer ϵ
- Logit scales



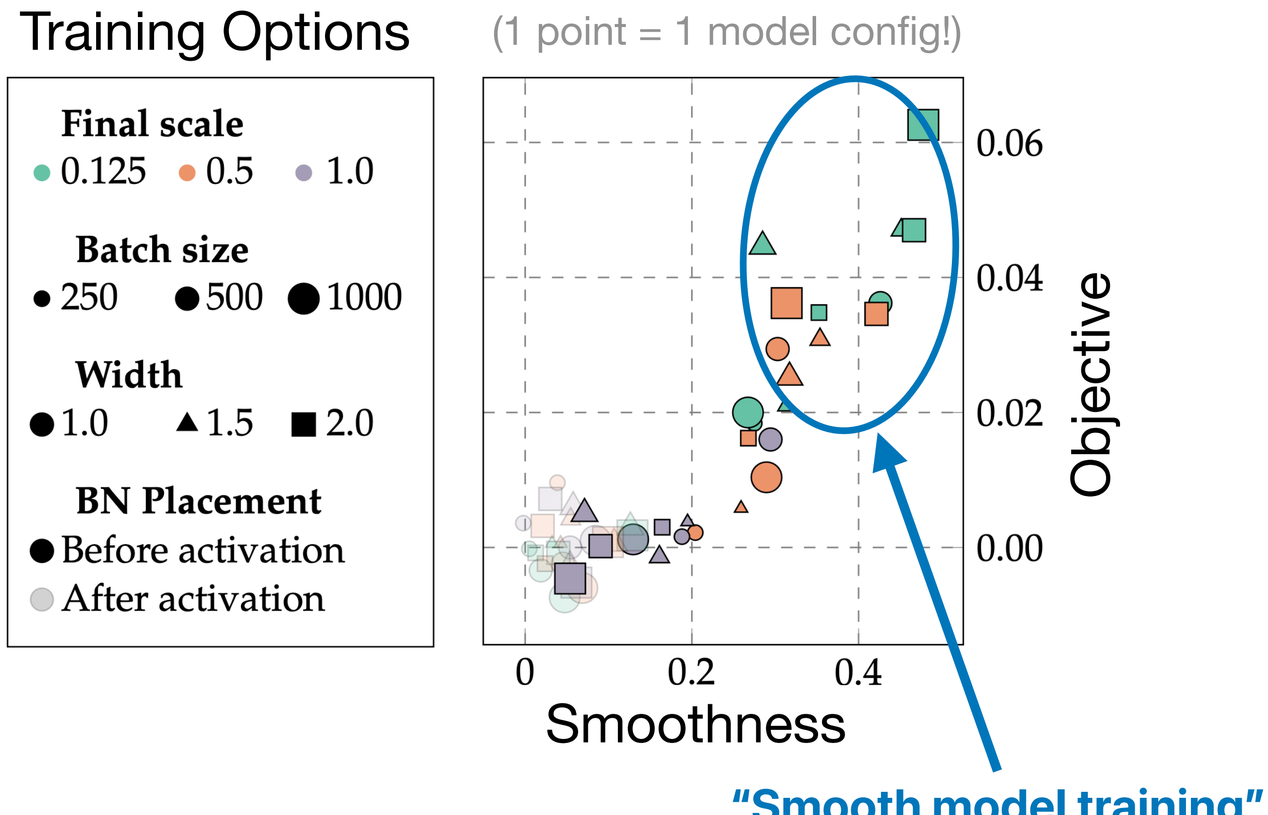
Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

- Layer composition, width
- Learning rate/weight decay/batch size
- Adam ϵ_{root}, momentum/warmup
- Normalizer ϵ
- Logit scales
- LM only:



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

Layer composition, width

Learning rate/weight decay/batch size

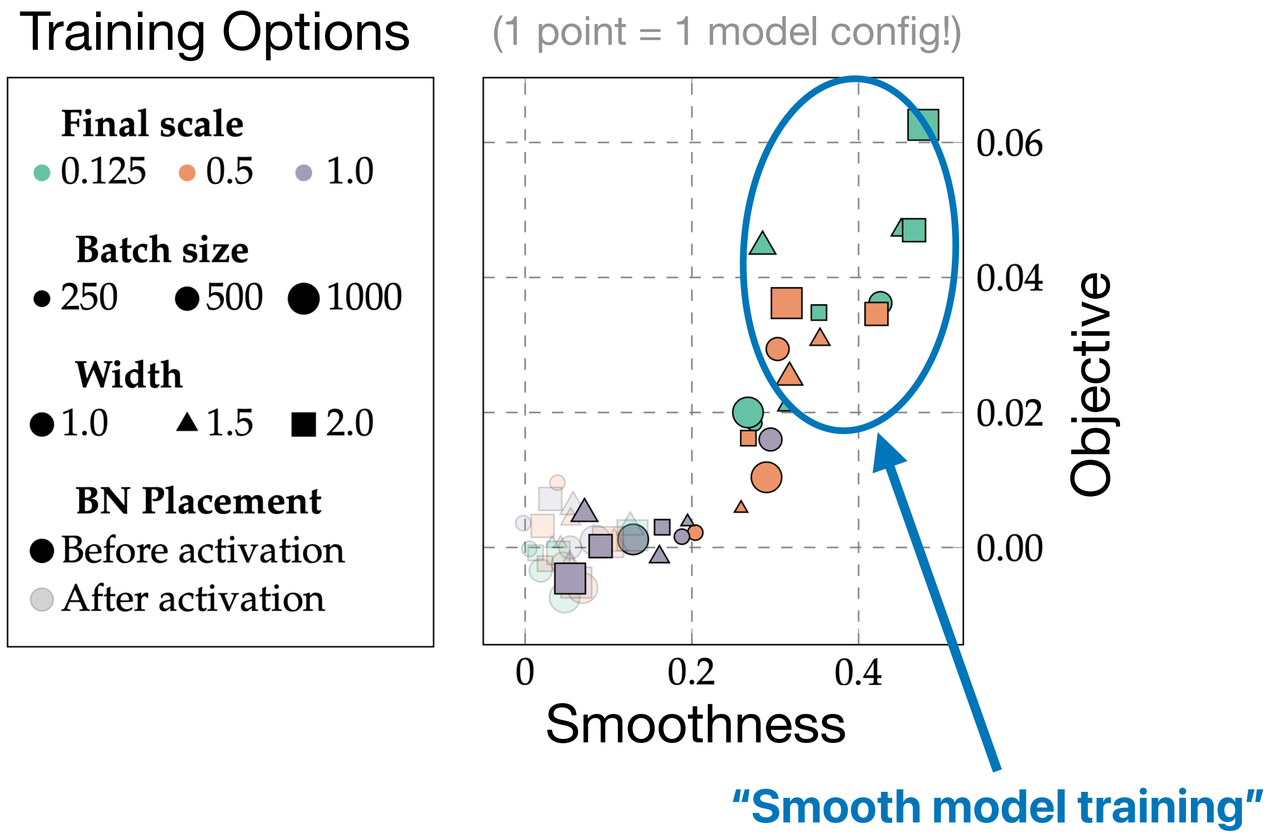
Adam ϵ_{root}, momentum/warmup

Normalizer ϵ

Logit scales

LM only:

QK-layernorm



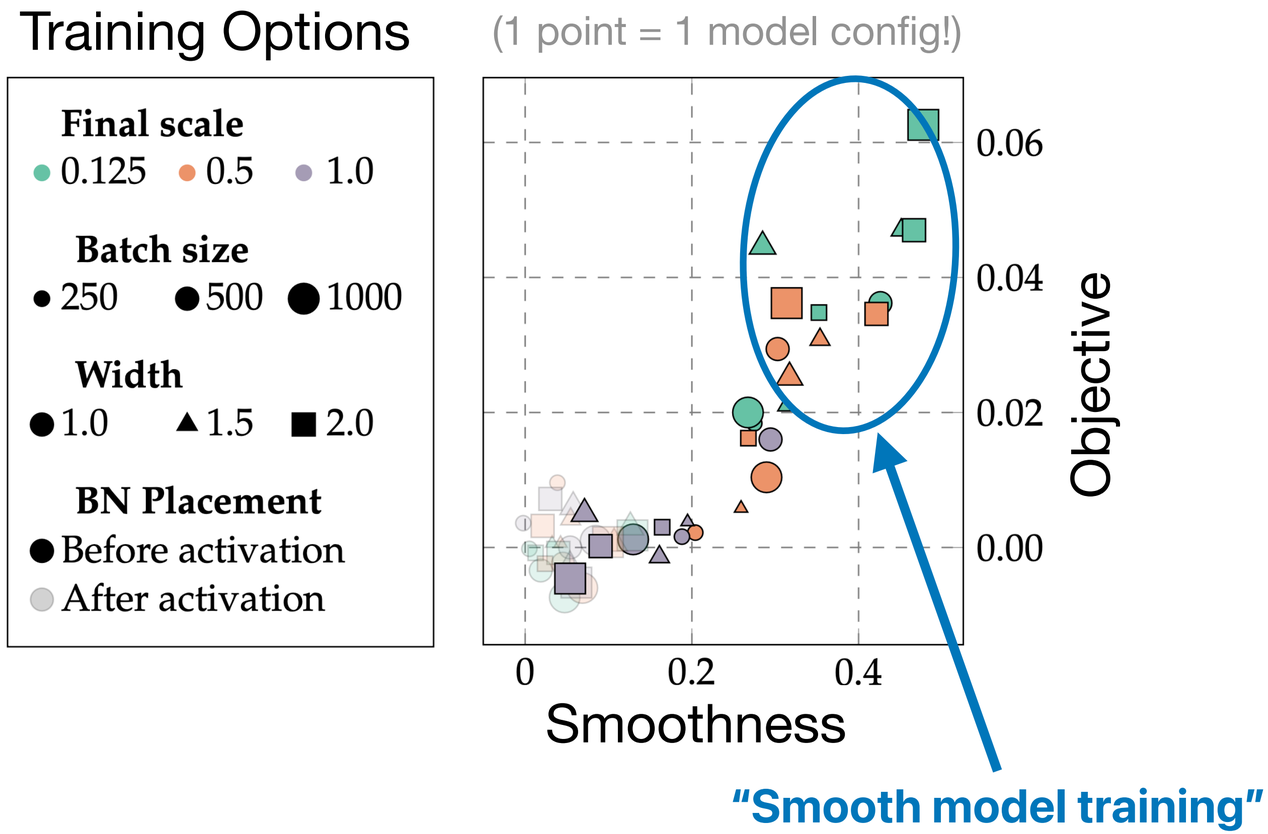
Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

- Layer composition, width
- Learning rate/weight decay/batch size
- Adam ϵ_{root}, momentum/warmup
- Normalizer ϵ
- Logit scales
- LM only:
  - QK-layernorm
  - Uncoupled (un)embedding



Problem #2: (Lack of) Smoothness

Constructing "smooth" model training routines

Our metric accurately predicts optimization success in practice!

How to train your *smooth* model:

Layer composition, width

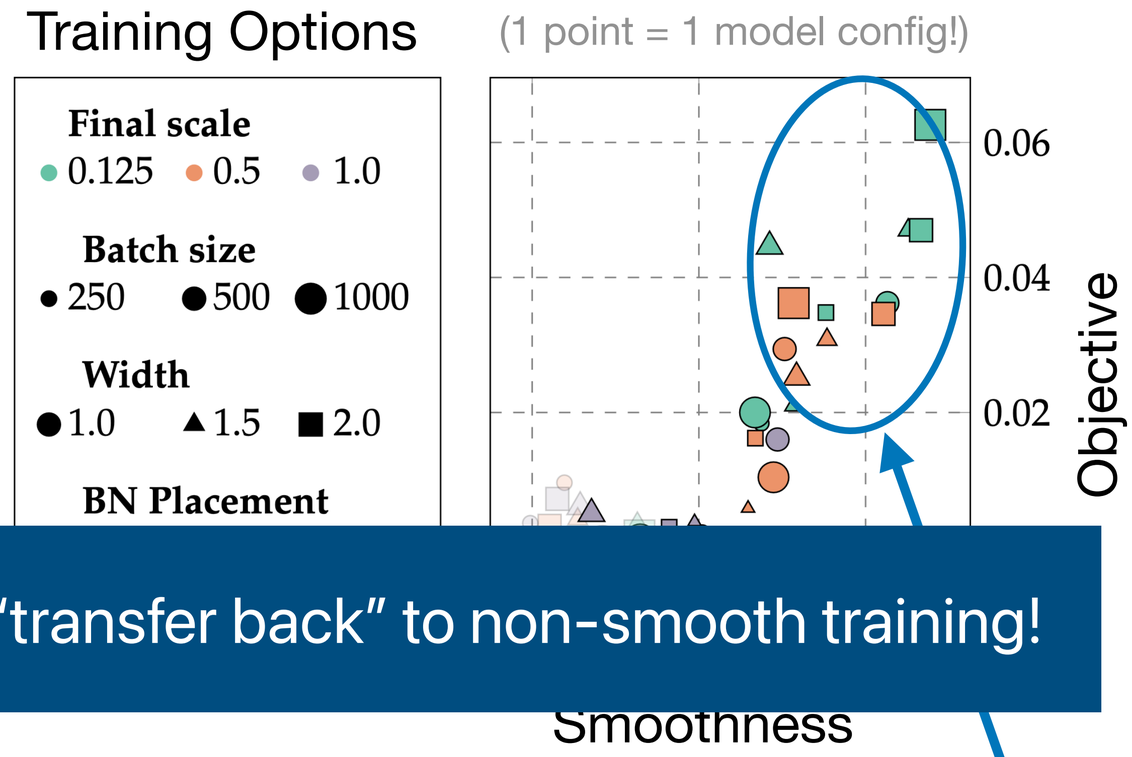
Learning rate/weight decay/batch size

Adam ϵ_{root}, momentum/warmup

Normalizer ϵ

Logit scales

Uncoupled (un)embedding



Key property: Optimized parameters "transfer back" to non-smooth training!

Roadmap

~~Main idea: The metagradient~~

~~Prequel: Two challenges~~

Looking forward

Future work

Our goal/hope is for the following recipe to be a general one:

1. Pick a design choice you're interested in optimizing/predicting
2. Write problem as a metaparameter prediction or optimization problem
3. Find a smooth training setup
4. Compute the metagradient
5. Apply gradient descent/Taylor approximation

What do we need to get there?

Future work

What do we need to get there?

1. A more systematic way of finding smooth models
2. More computational efficiency gains
3. More creative ways of applying metagradients
   - Hyperparameter optimization? (Simple example in the paper)
   - Data weighting *networks*?
   - Synthetic data generators?
   - Gradient pre-conditioners?

   "Anything we can continuously parameterize, we can (try to) optimize"

Conclusion

Design space of ML is too large to explore → how do we think about design choices?

Approach: Take the **gradient** of a model output with respect to metaparameters

Challenges:

- Computing the metagradient at scale
- Making the metagradient useful for optimization

Addressing these challenges unlocks:

- Data selection (> 2x more effective) & attribution
- Data poisoning (> 10x more effective)
- Gradient-based hyperparameter optimization
- **What next?**

See the paper for more!

🔗 https://arxiv.org/abs/2503.13751

📪 andrewi@stanford.edu

🌐 andrewilyas.com