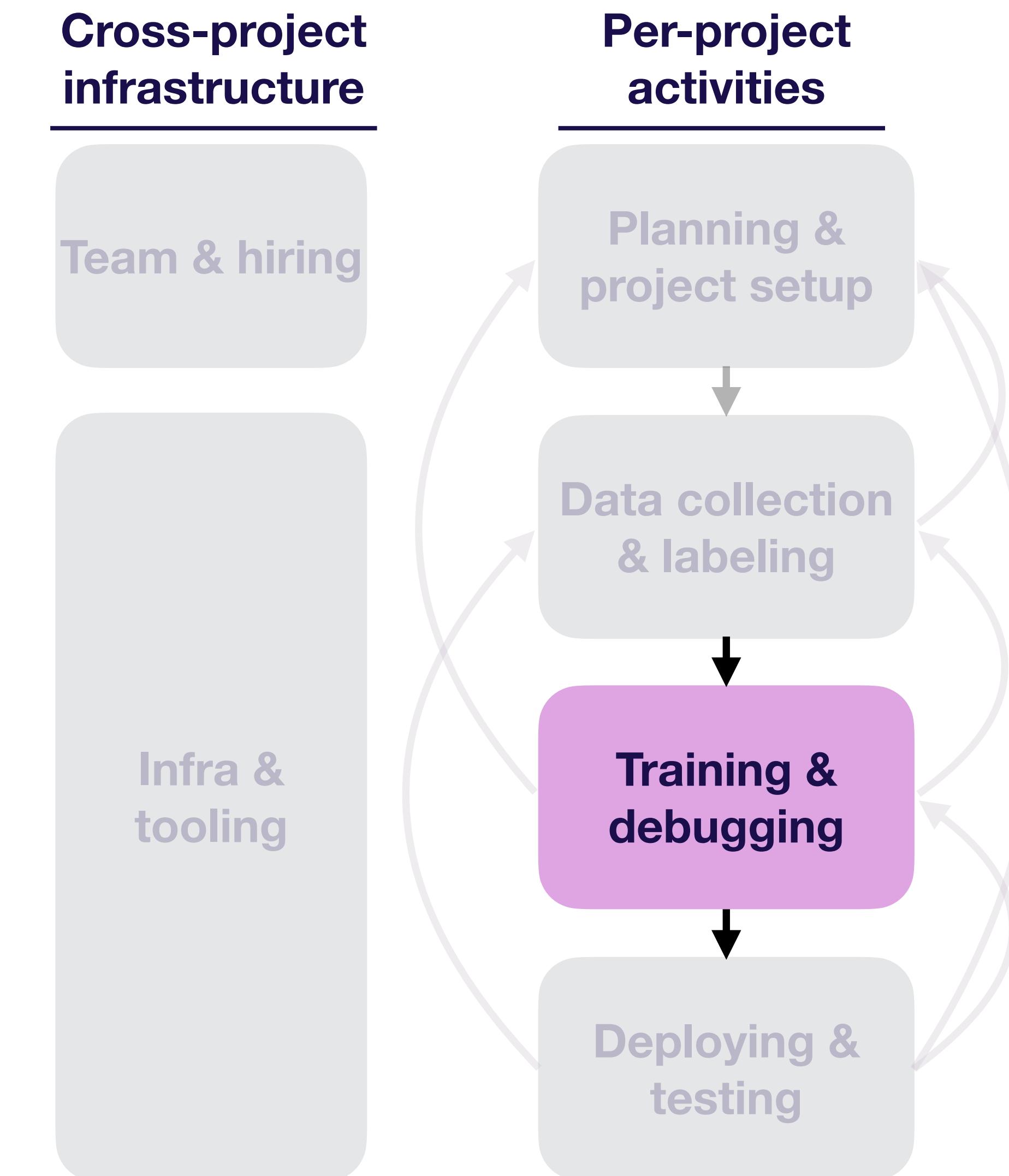
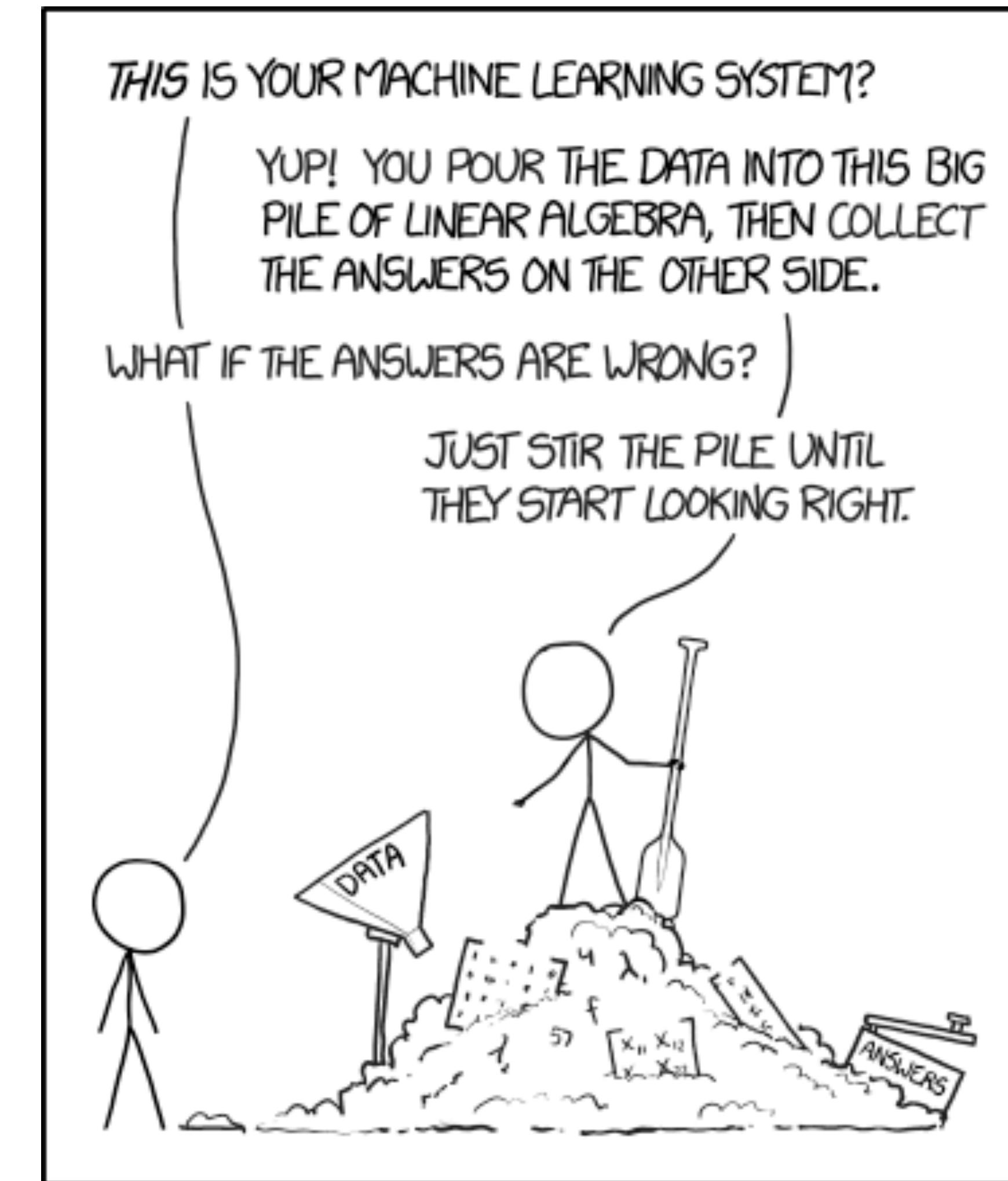


Troubleshooting Deep Neural Networks

Lifecycle of a ML project



Why talk about DL troubleshooting?



XKCD, <https://xkcd.com/1838/>

Why talk about DL troubleshooting?



Andrej Karpathy

@karpathy

Following



Debugging: first it doesn't compile. then
doesn't link. then segfaults. then gives all
zeros. then gives wrong answer. then only
maybe works

Why talk about DL troubleshooting?

Common sentiment among practitioners:

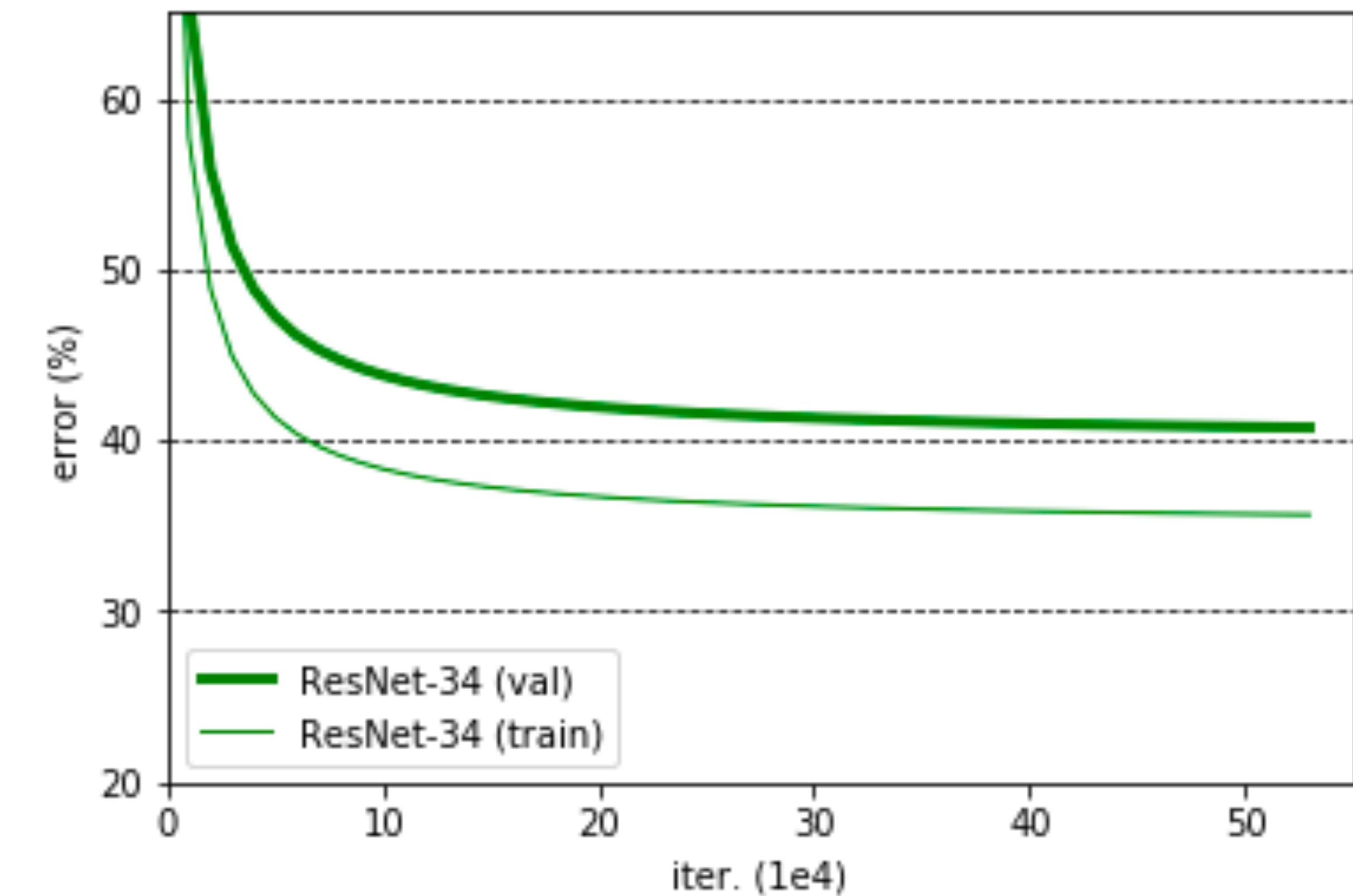
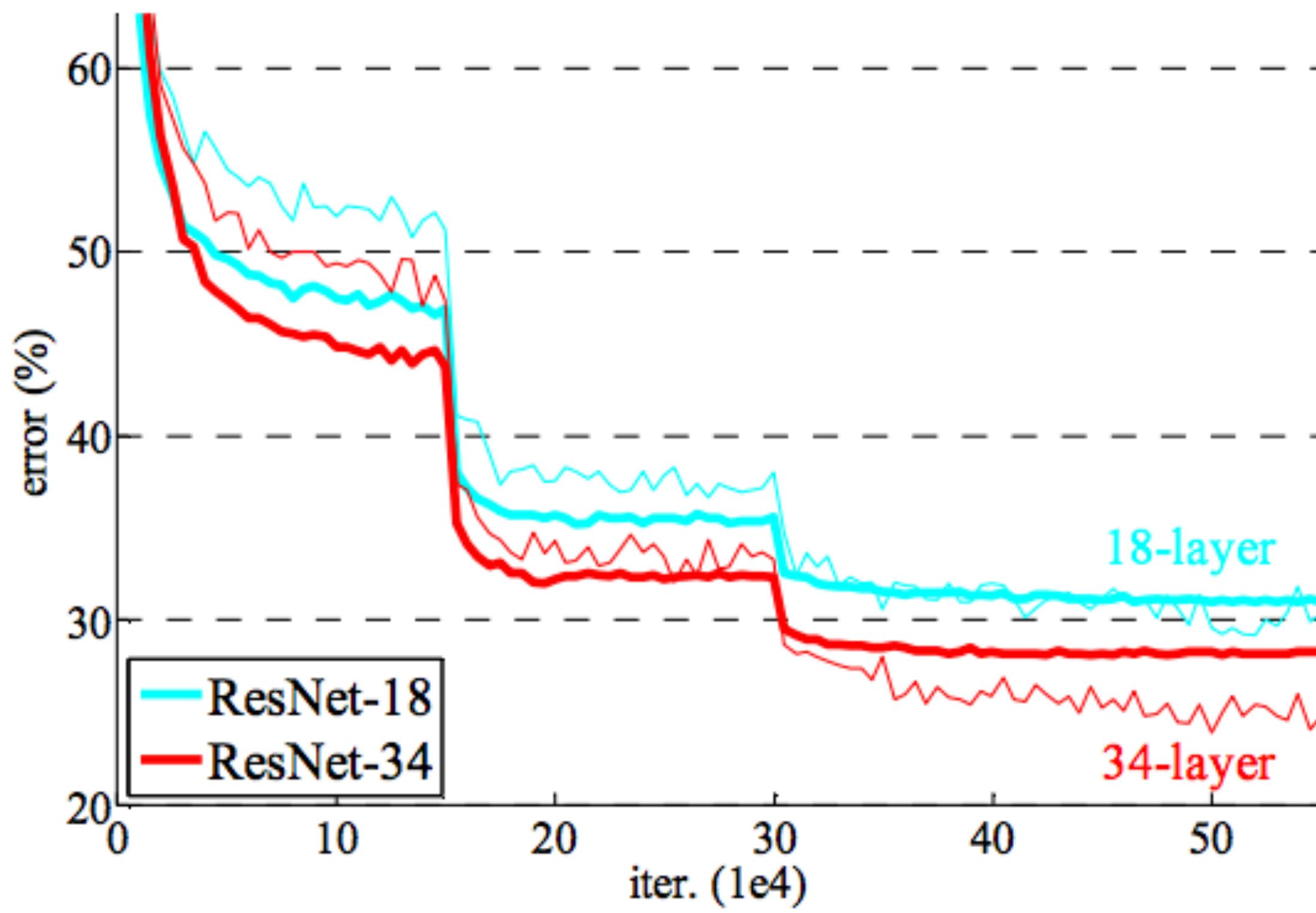
80-90% of time debugging and tuning

10-20% deriving math or implementing things

Why is DL troubleshooting so hard?

Suppose you can't reproduce a result

Your learning curve

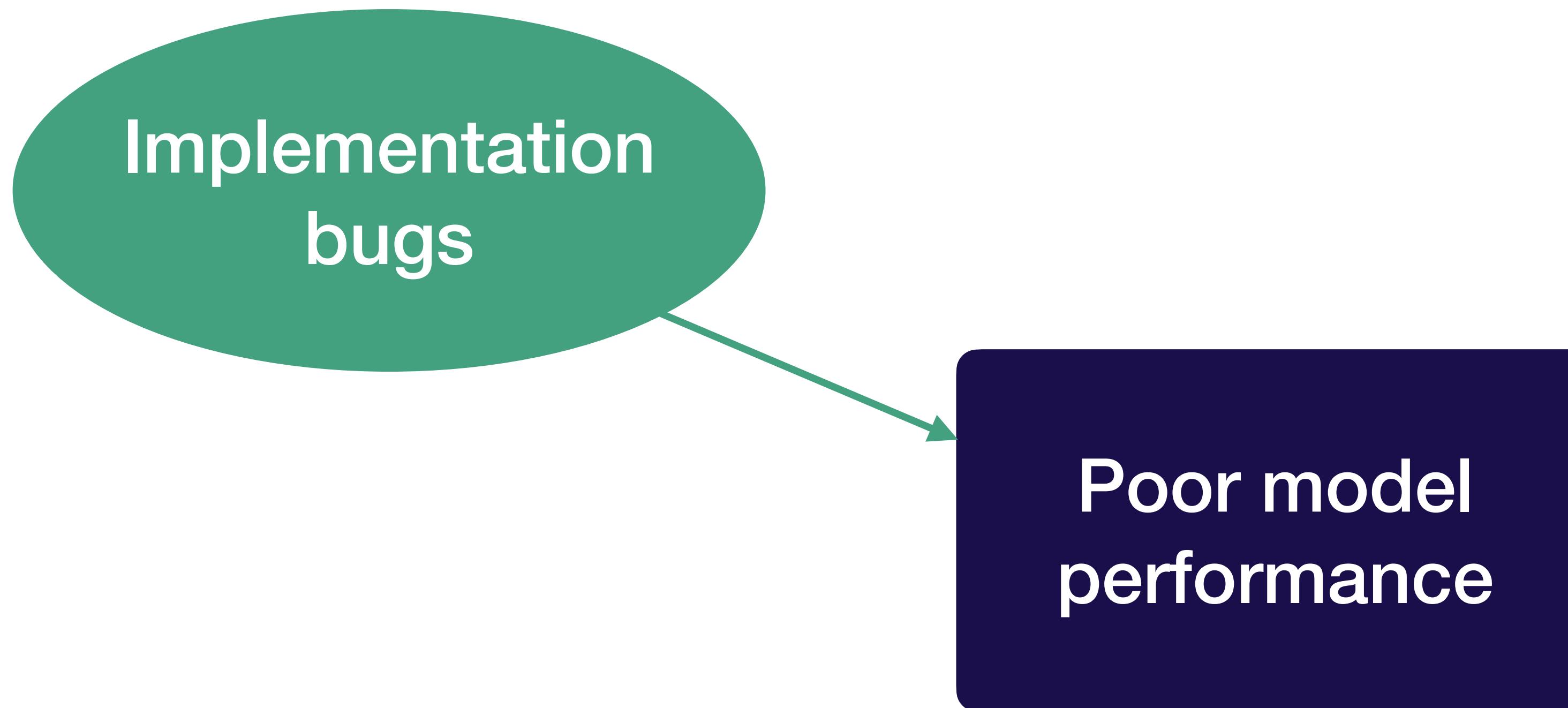


He, Kaiming, et al. "Deep residual learning for image recognition."
Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Why is your performance worse?

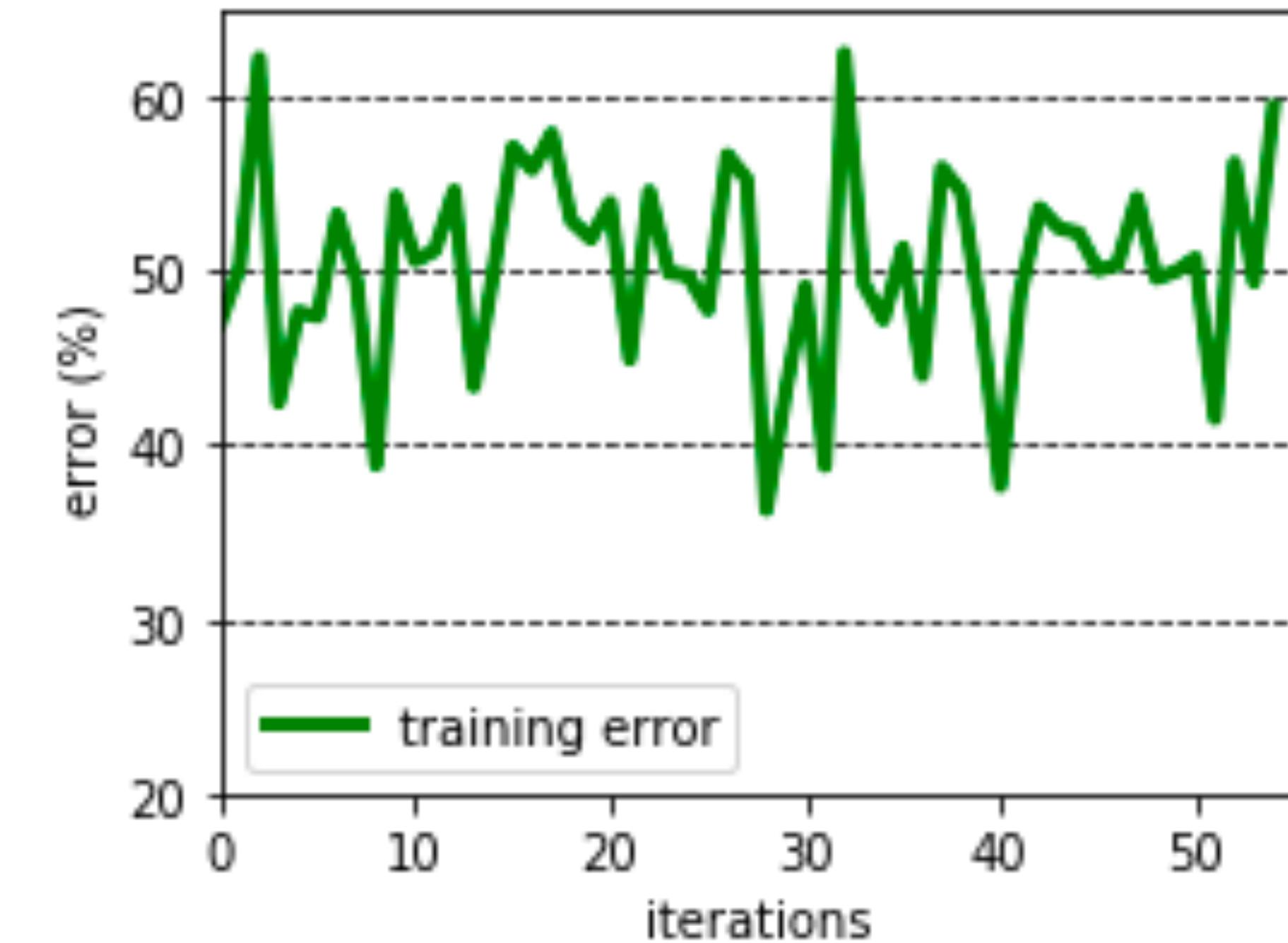
Poor model
performance

Why is your performance worse?



Most DL bugs are invisible

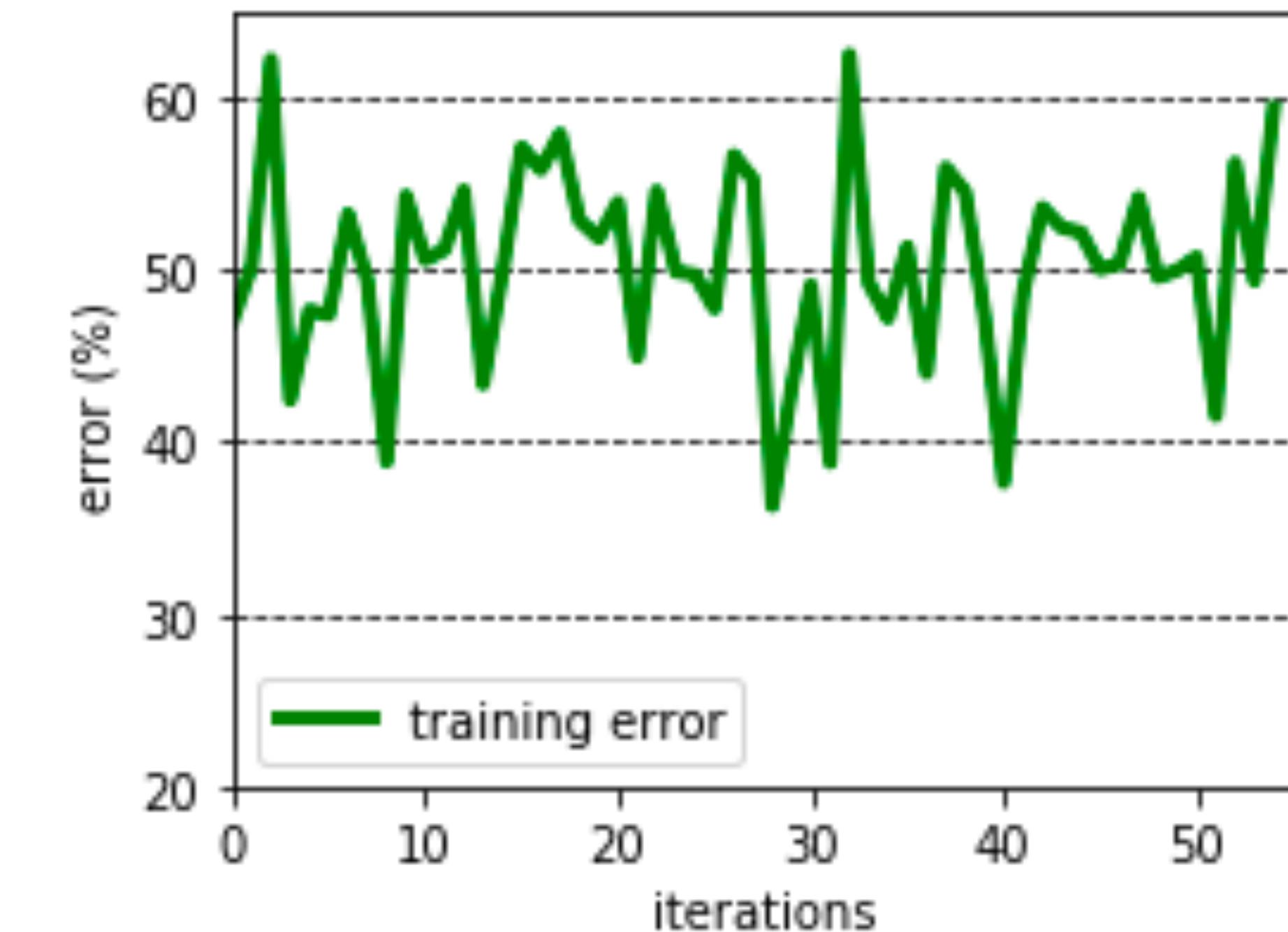
```
1 features = glob.glob('path/to/features/*')
2 labels = glob.glob('path/to/labels/*')
3 train(features, labels)
```



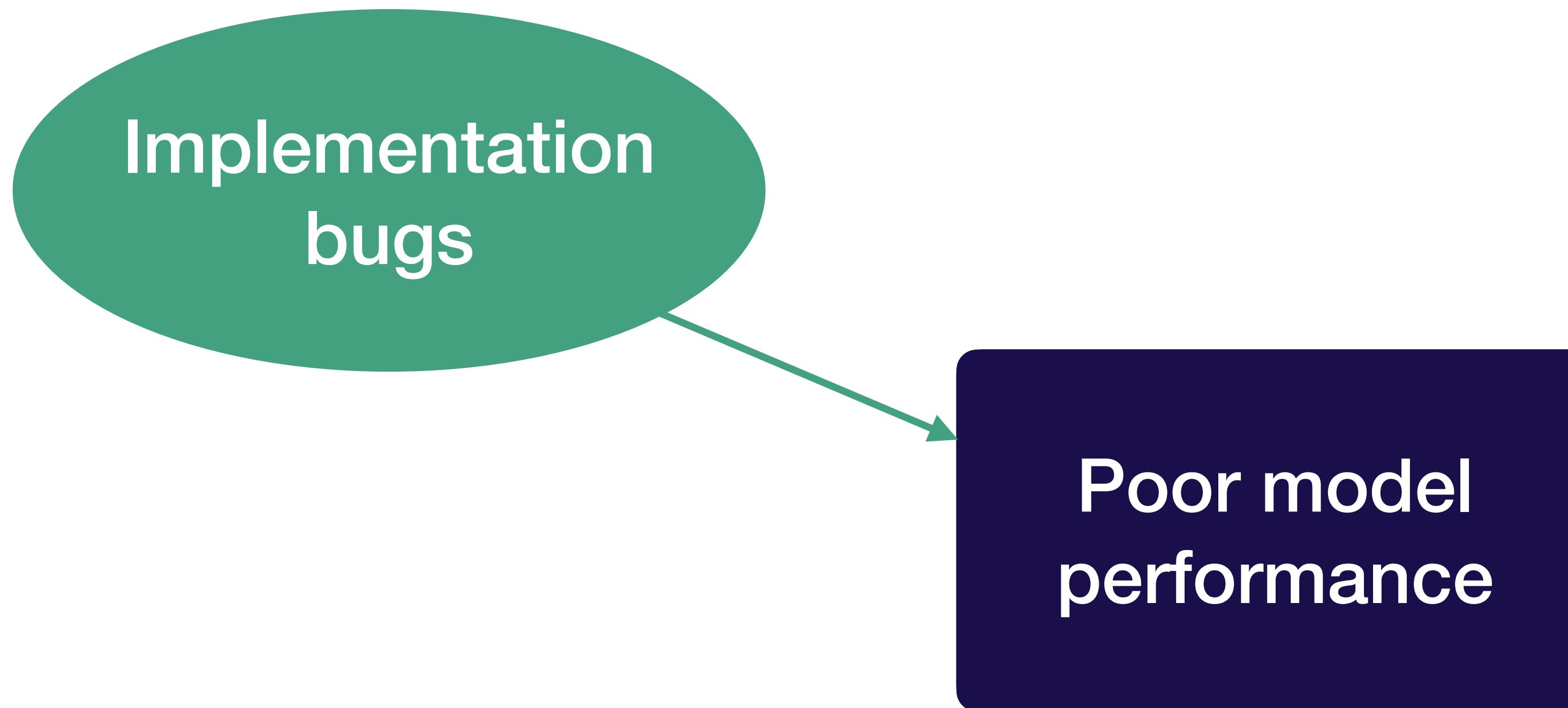
Most DL bugs are invisible

Labels out of order!

```
1 features = glob.glob('path/to/features/*')
2 labels = glob.glob('path/to/labels/*')
3 train(features, labels)
```



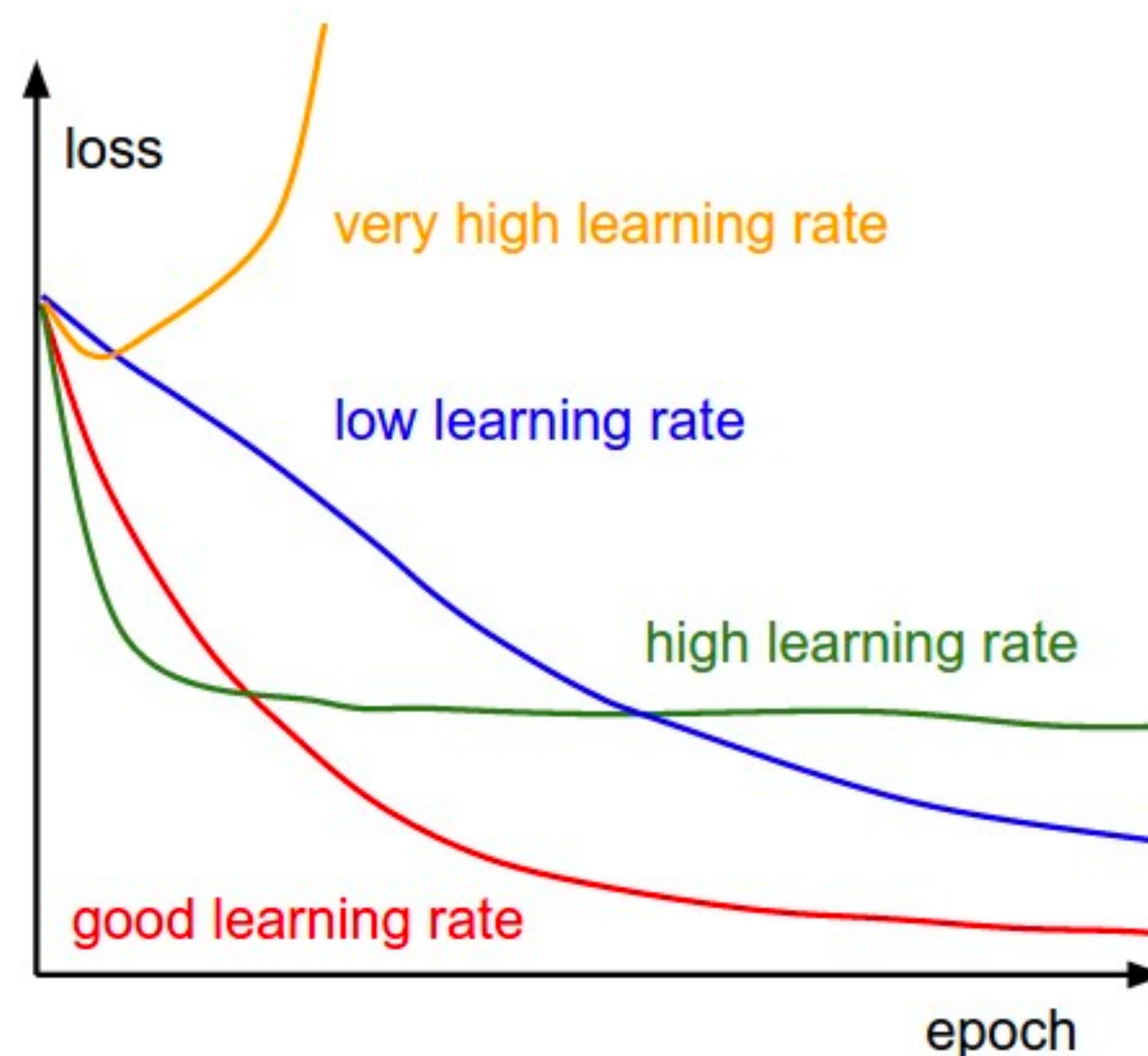
Why is your performance worse?



Why is your performance worse?

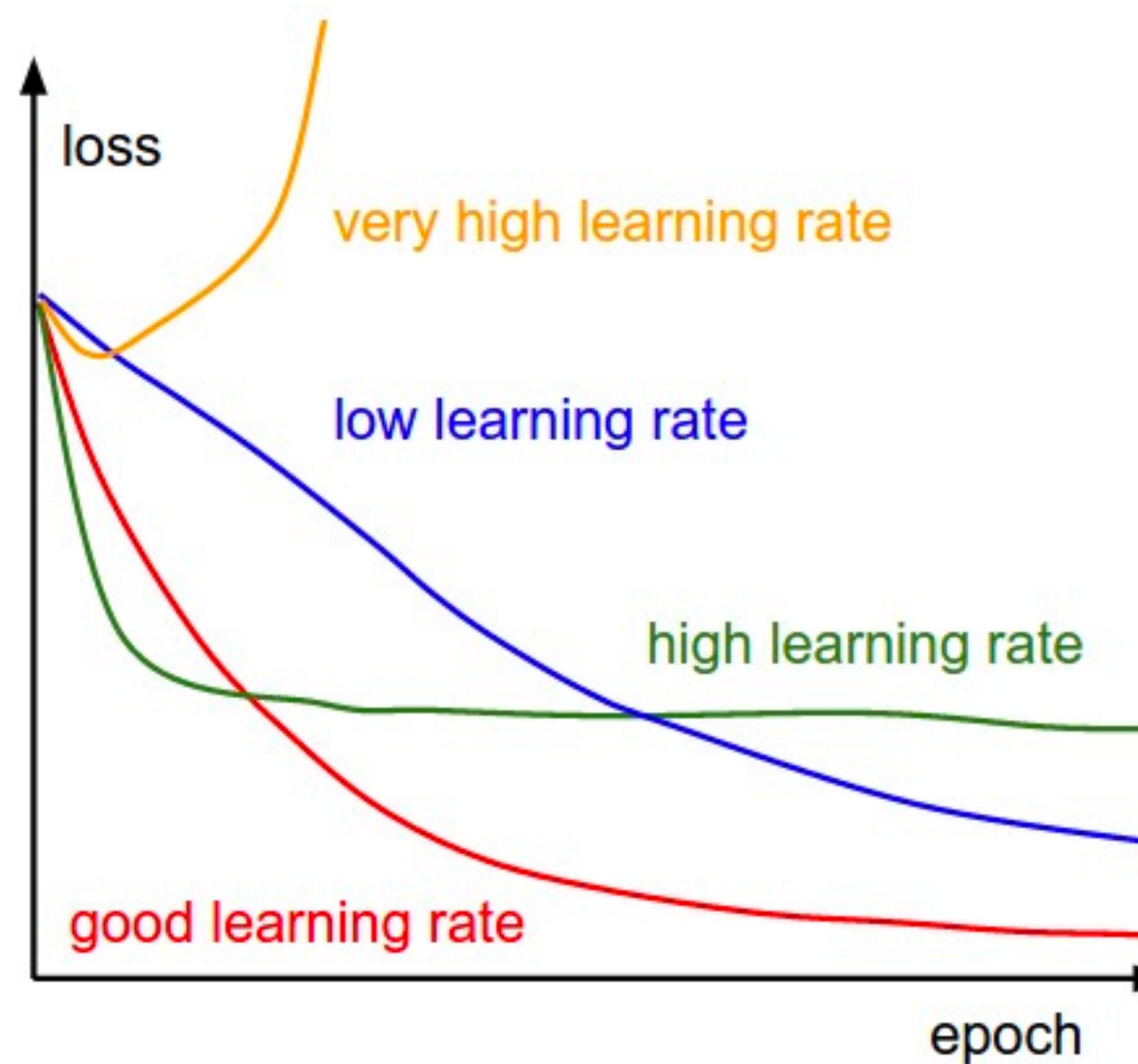


Models are sensitive to hyperparameters

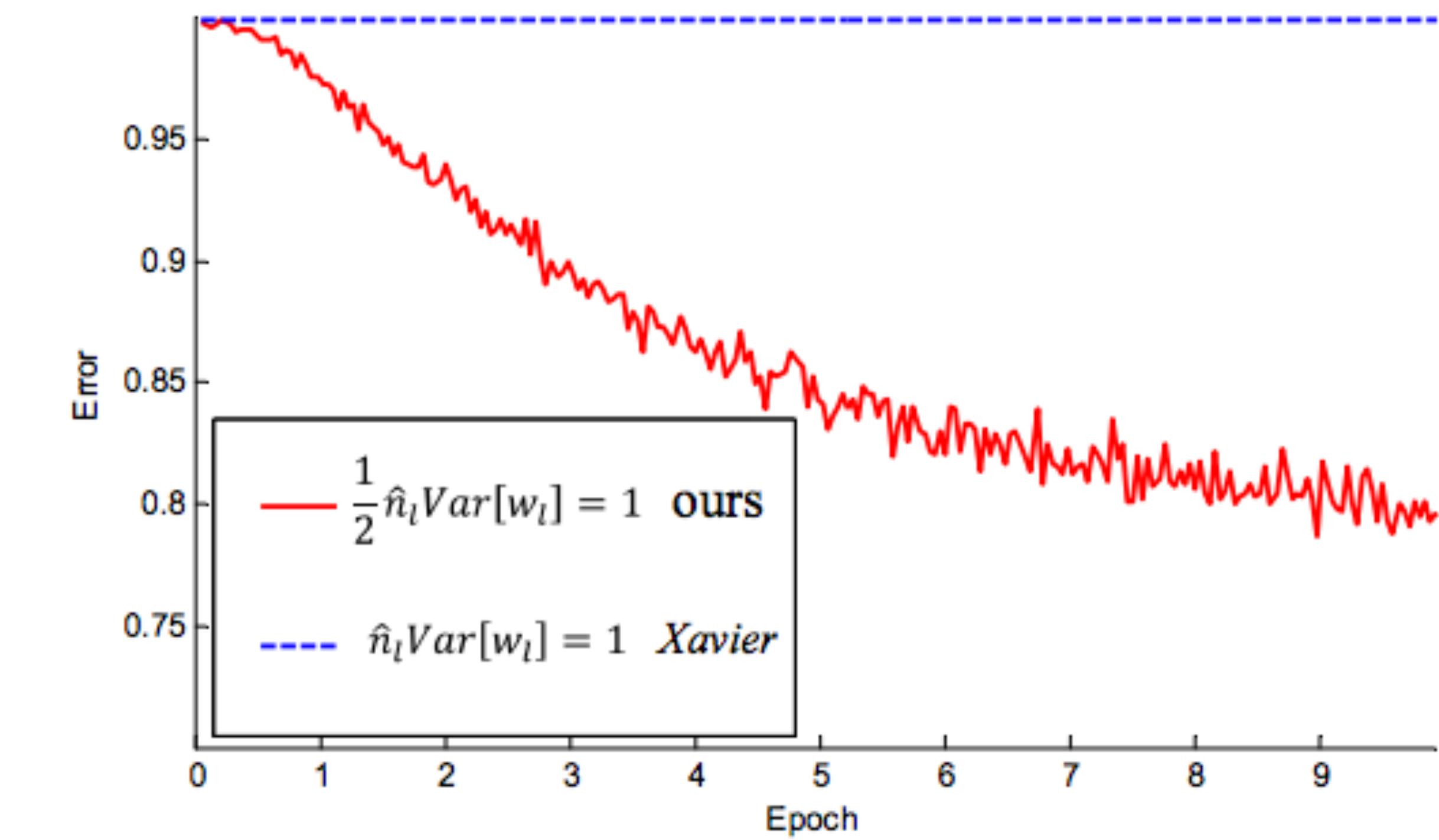


Andrey Karpathy, CS231n course notes

Models are sensitive to hyperparameters

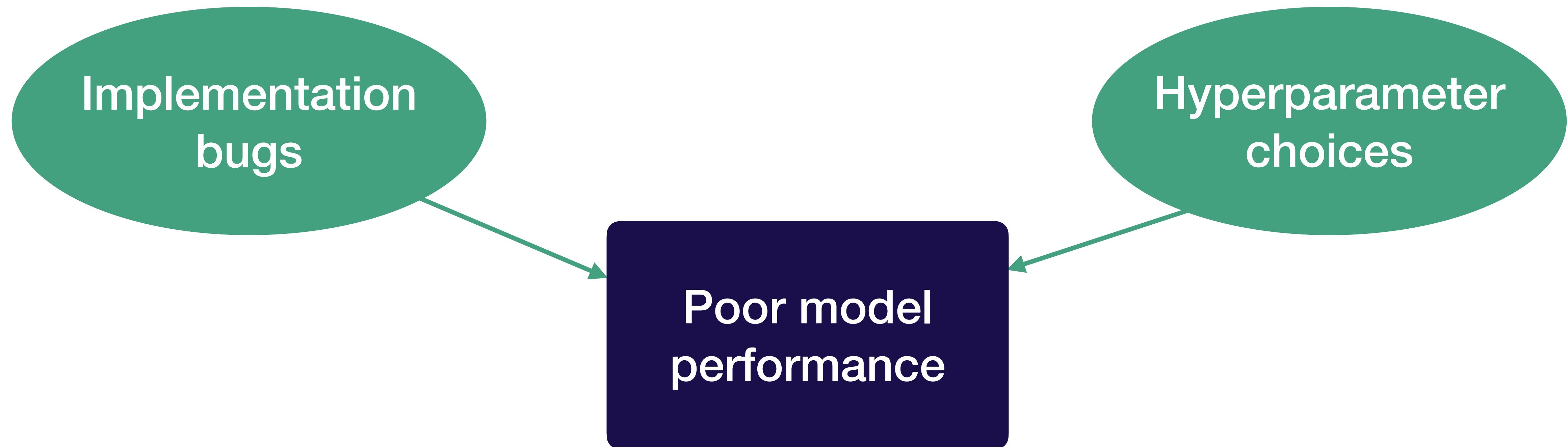


Andrej Karpathy, CS231n course notes

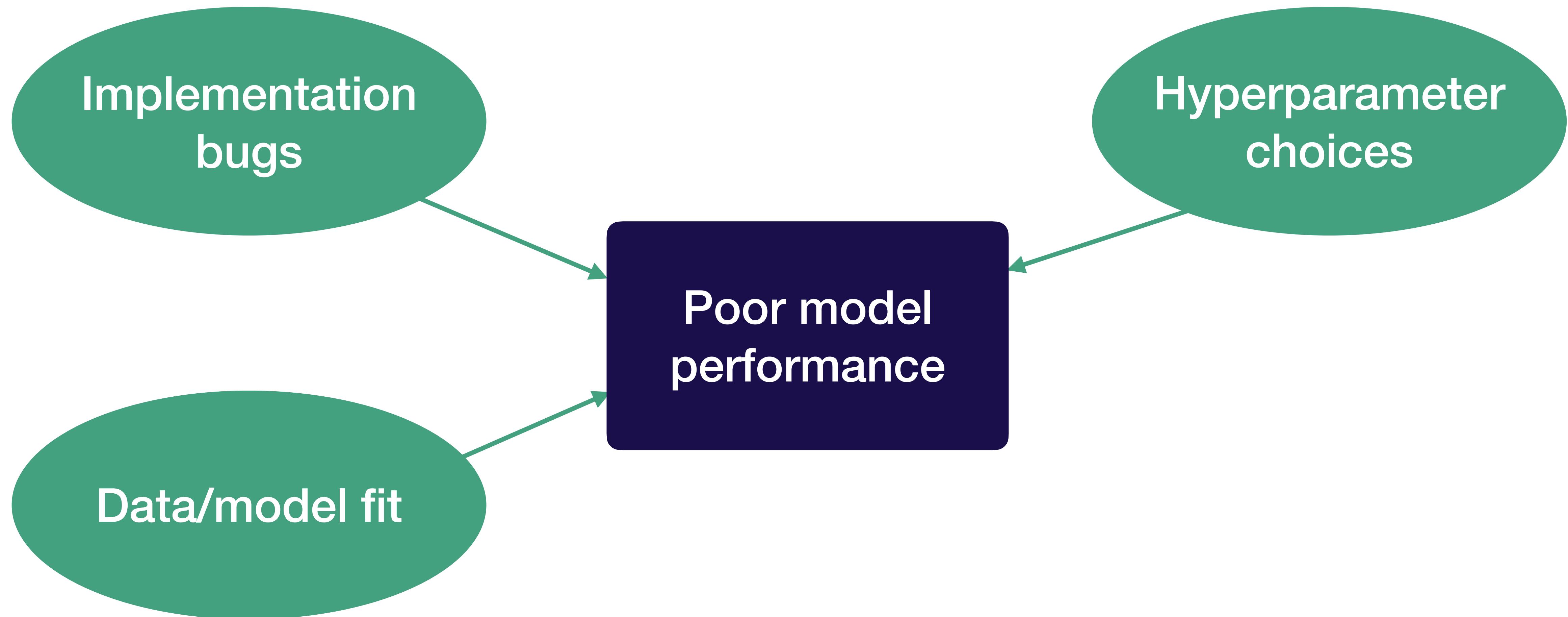


He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

Why is your performance worse?

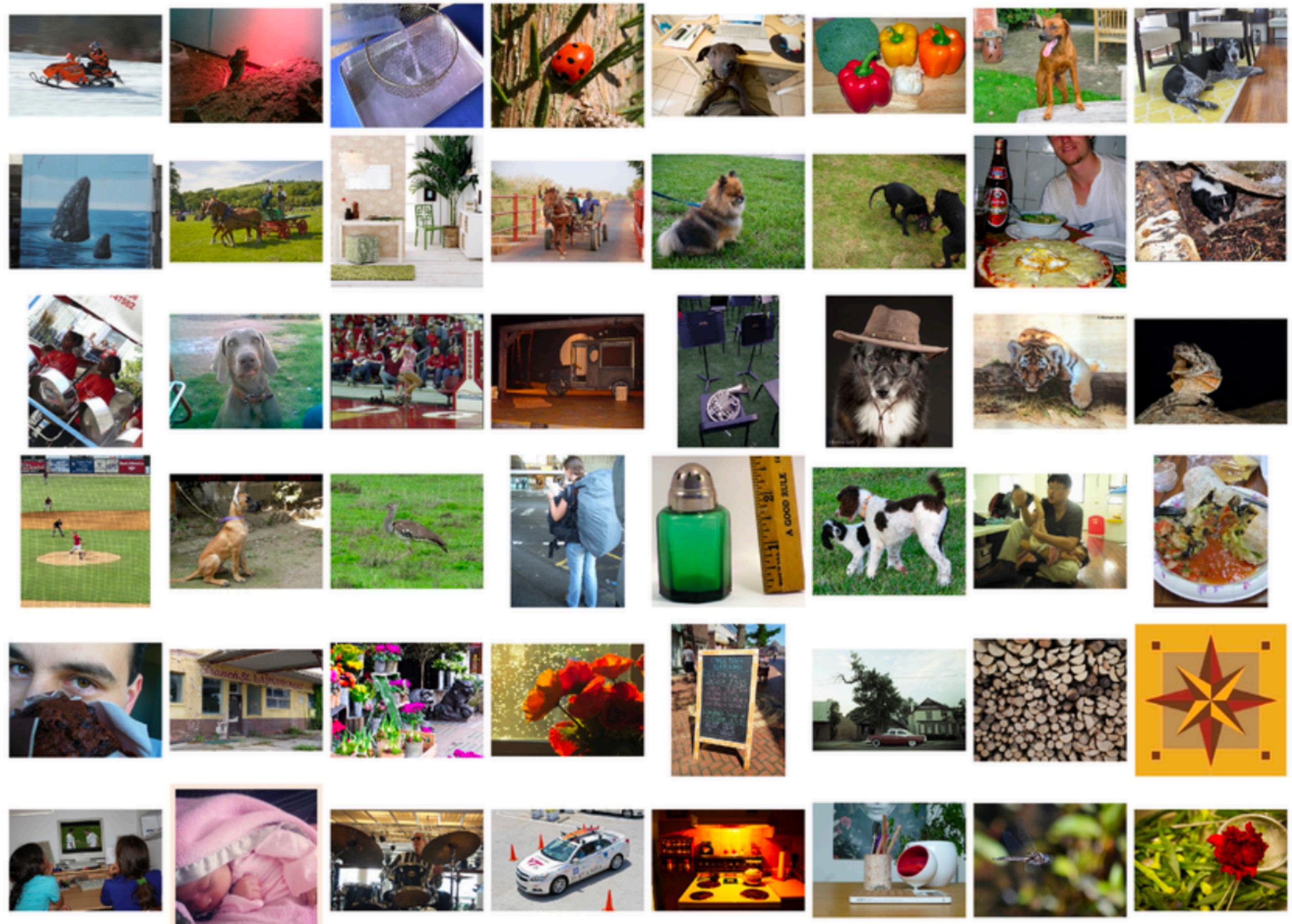


Why is your performance worse?



Data / model fit

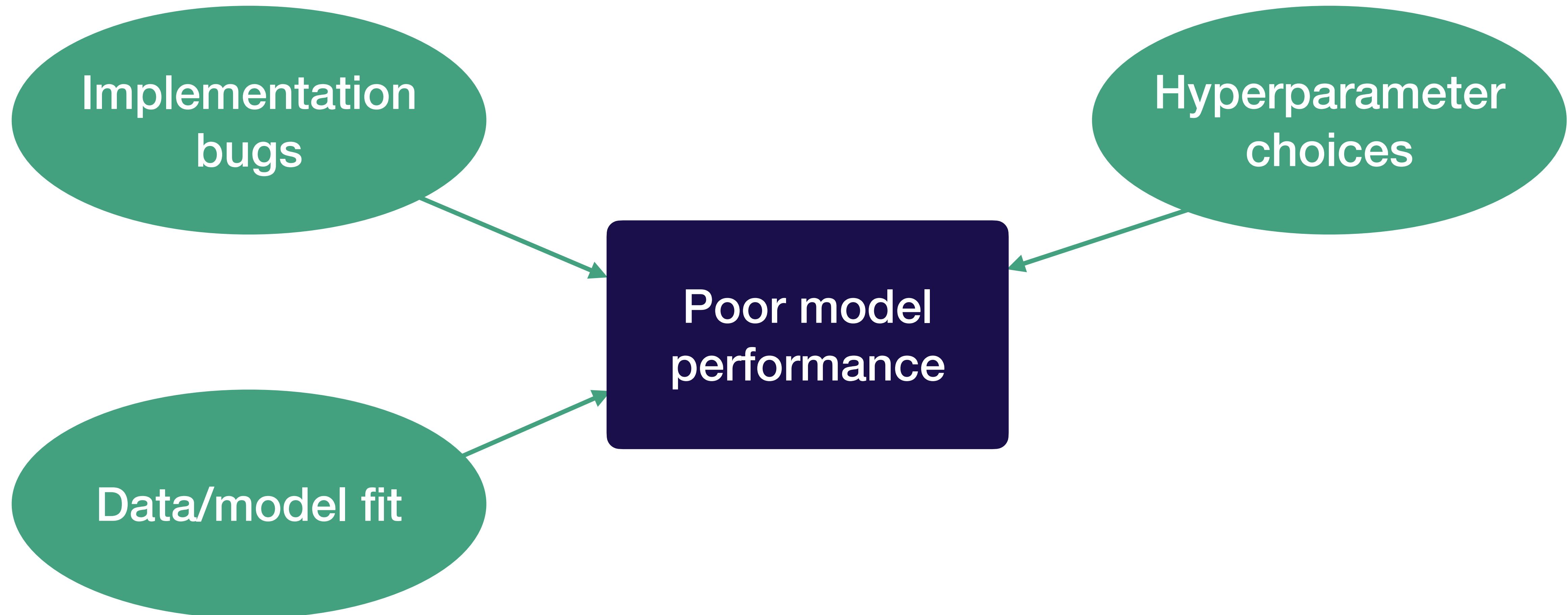
Data from the paper: ImageNet



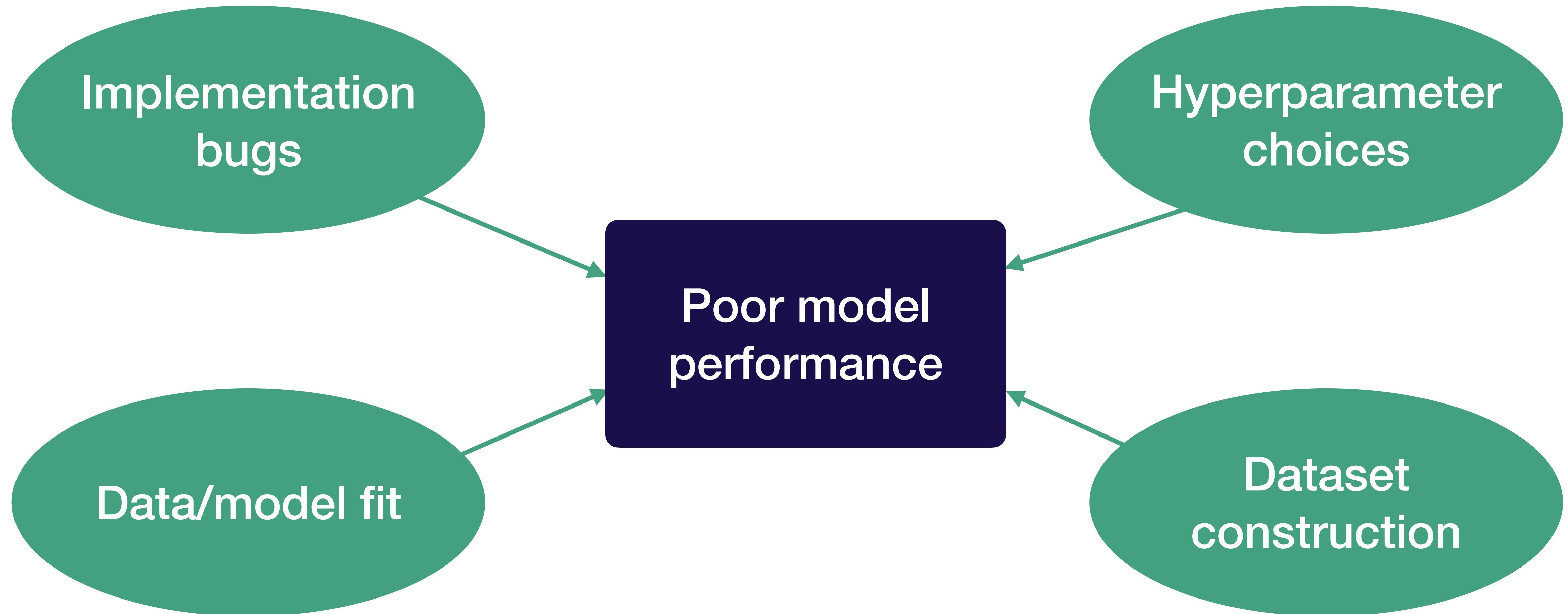
Yours: self-driving car images



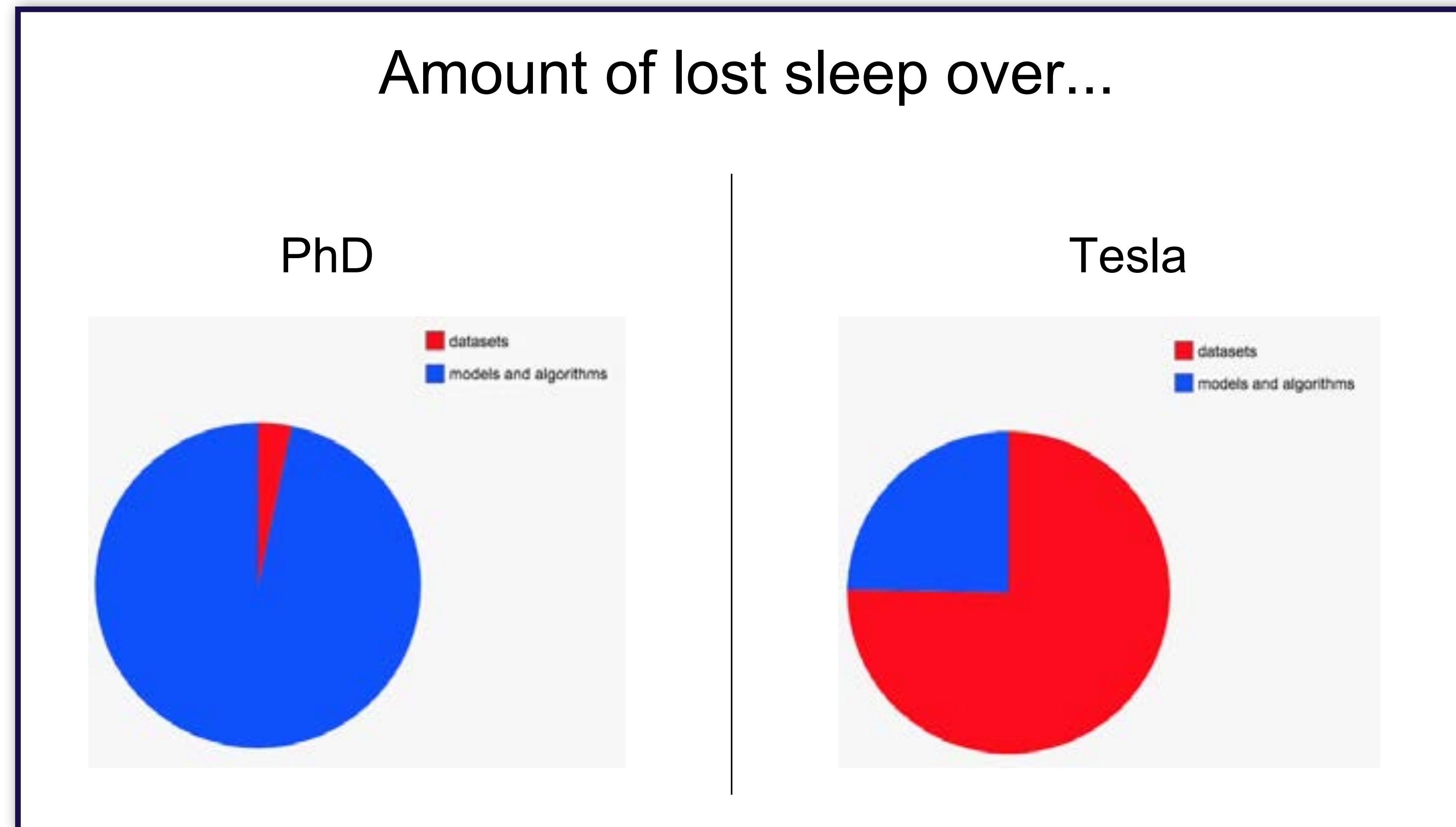
Why is your performance worse?



Why is your performance worse?



Constructing good datasets is hard



Slide from Andrej Karpathy's talk "Building the Software 2.0 Stack" at TrainAI 2018, 5/10/2018

Common dataset construction issues

- Not enough data
- Class imbalances
- Noisy labels
- Train / test from different distributions
- etc

Takeaways: why is troubleshooting hard?

- Hard to tell if you have a bug
- Lots of possible sources for the same degradation in performance
- Results can be sensitive to small changes in hyperparameters and dataset makeup

Strategy for DL troubleshooting

Key mindset for DL troubleshooting

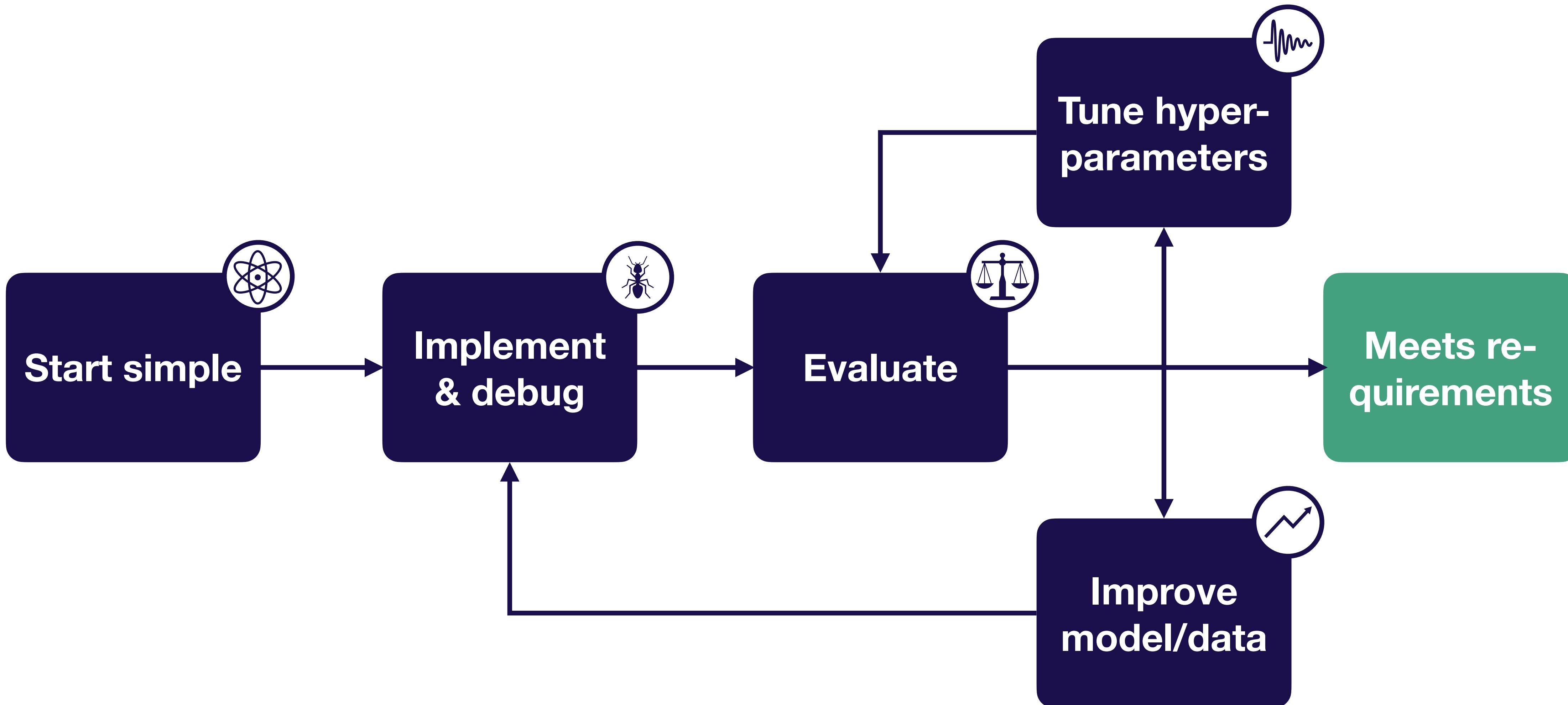
Pessimism

Key idea of DL troubleshooting

Since it's hard to disambiguate errors...

....Start simple and gradually ramp up complexity

Strategy for DL troubleshooting

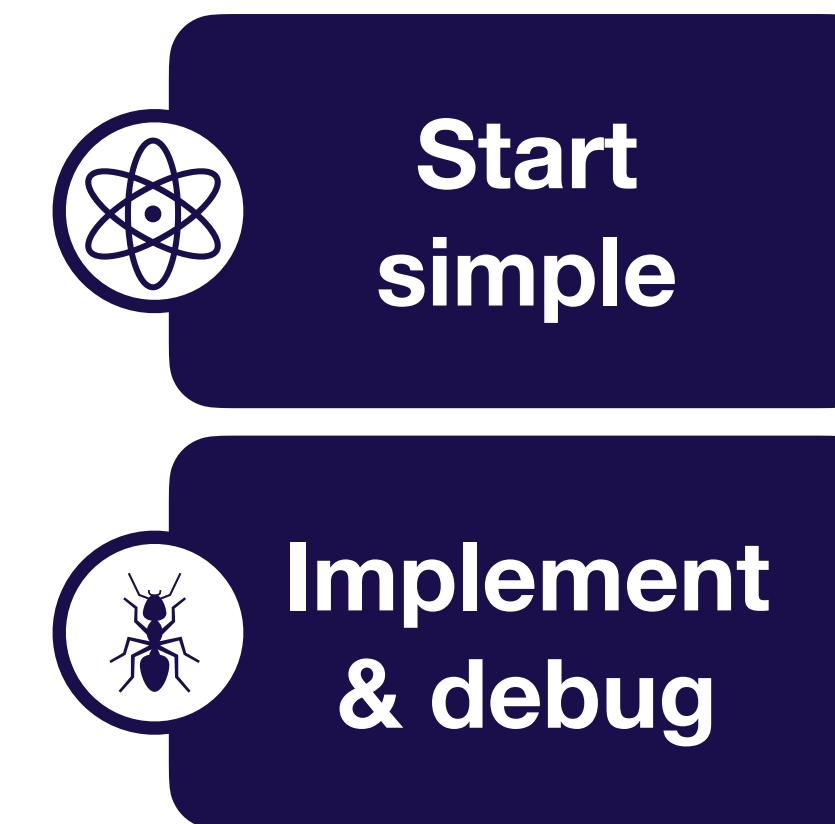


Quick summary



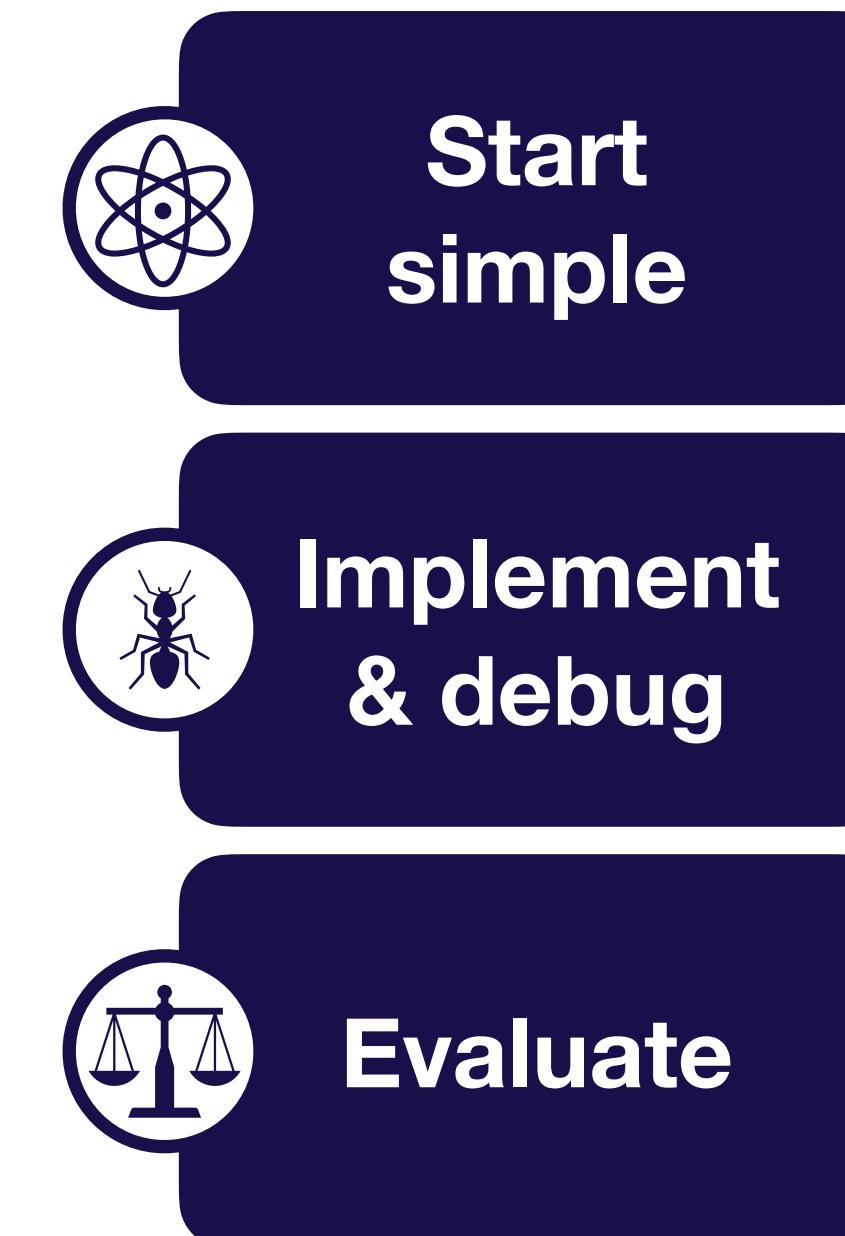
- Choose the simplest model & data possible
(e.g., LeNet on a subset of your data)

Quick summary



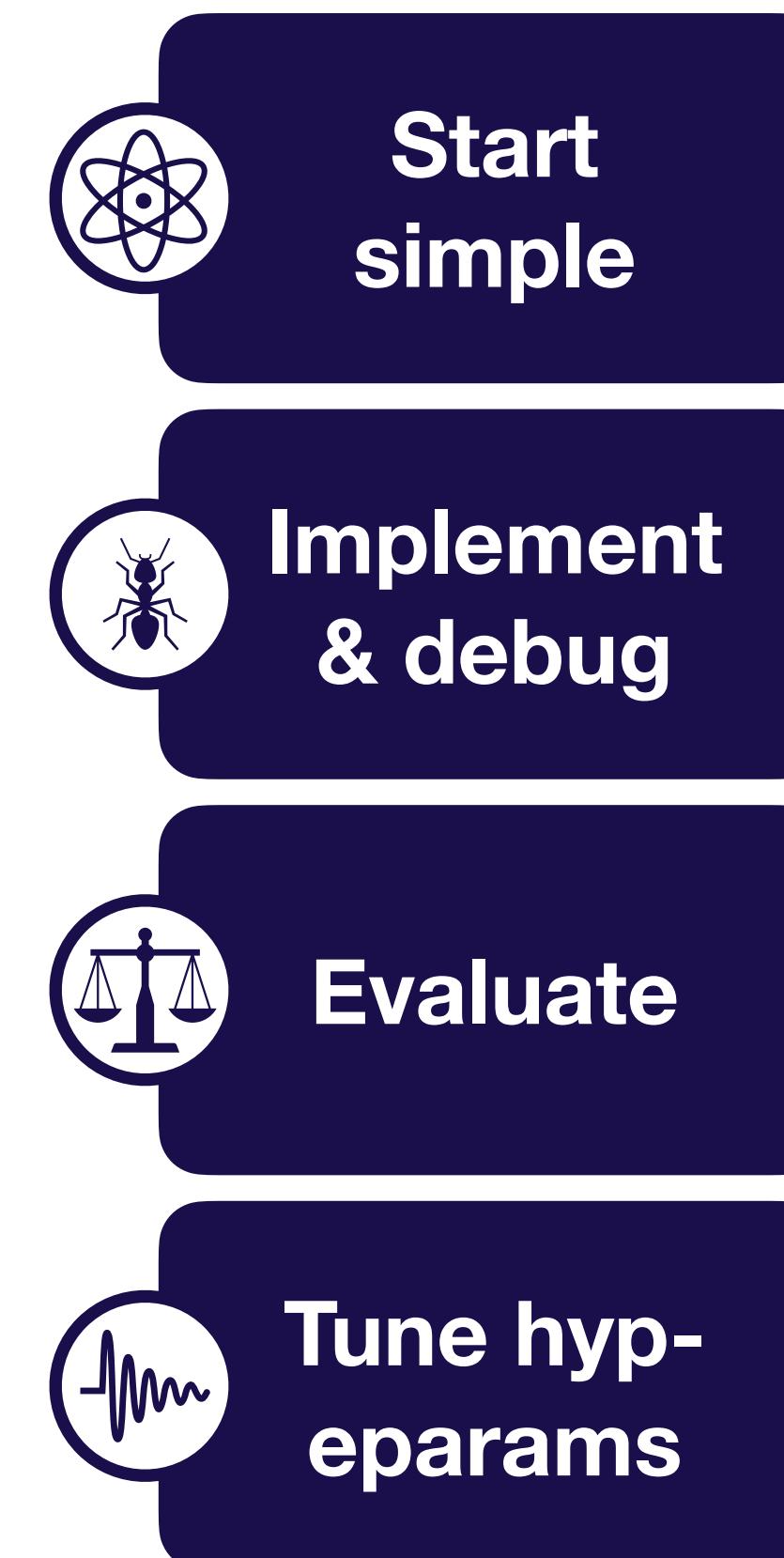
- Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
- Once model runs, overfit a single batch & reproduce a known result

Quick summary



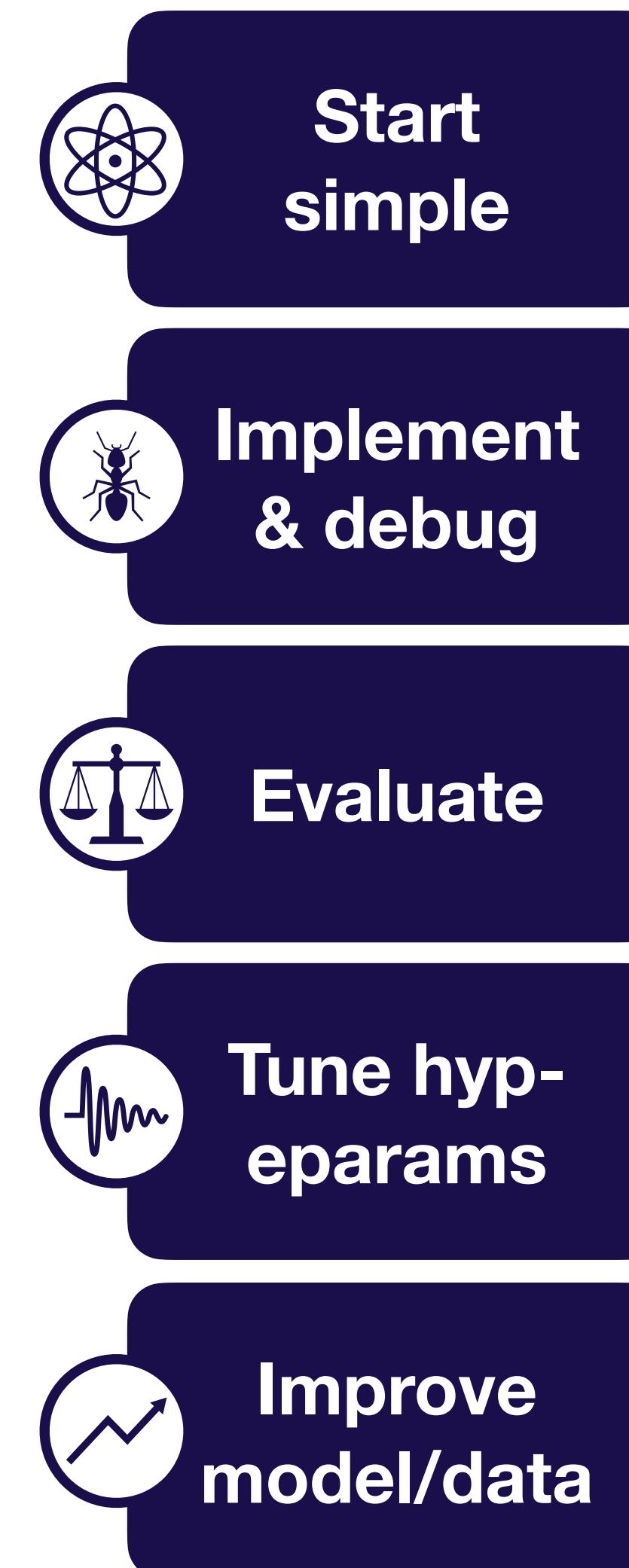
- **Choose the simplest model & data possible (e.g., LeNet on a subset of your data)**
- **Once model runs, overfit a single batch & reproduce a known result**
- **Apply the bias-variance decomposition to decide what to do next**

Quick summary



- Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
- Once model runs, overfit a single batch & reproduce a known result
- Apply the bias-variance decomposition to decide what to do next
- Use coarse-to-fine random searches

Quick summary



- **Choose the simplest model & data possible (e.g., LeNet on a subset of your data)**
- **Once model runs, overfit a single batch & reproduce a known result**
- **Apply the bias-variance decomposition to decide what to do next**
- **Use coarse-to-fine random searches**
- **Make your model bigger if you underfit; add data or regularize if you overfit**

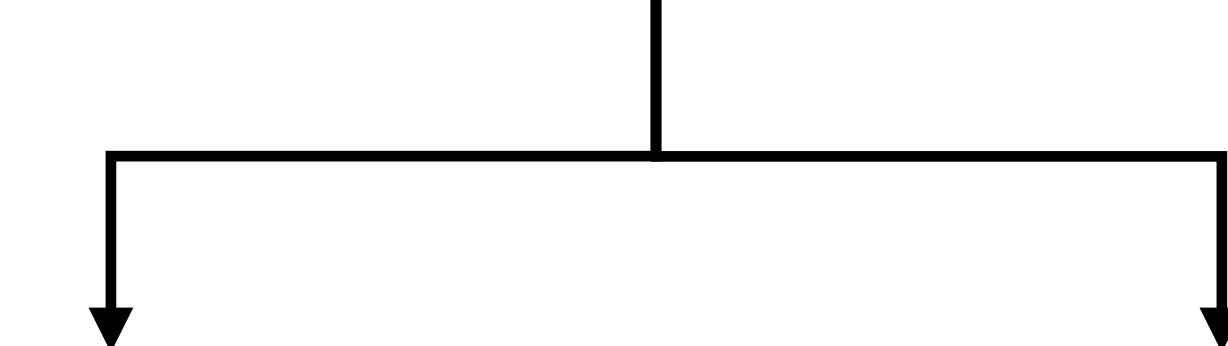
We'll assume you already have...

- Initial test set
- A single metric to improve
- Target performance based on human-level performance, published results, previous baselines, etc

We'll assume you already have...

- Initial test set
- A single metric to improve
- Target performance based on human-level performance, published results, previous baselines, etc

Running example

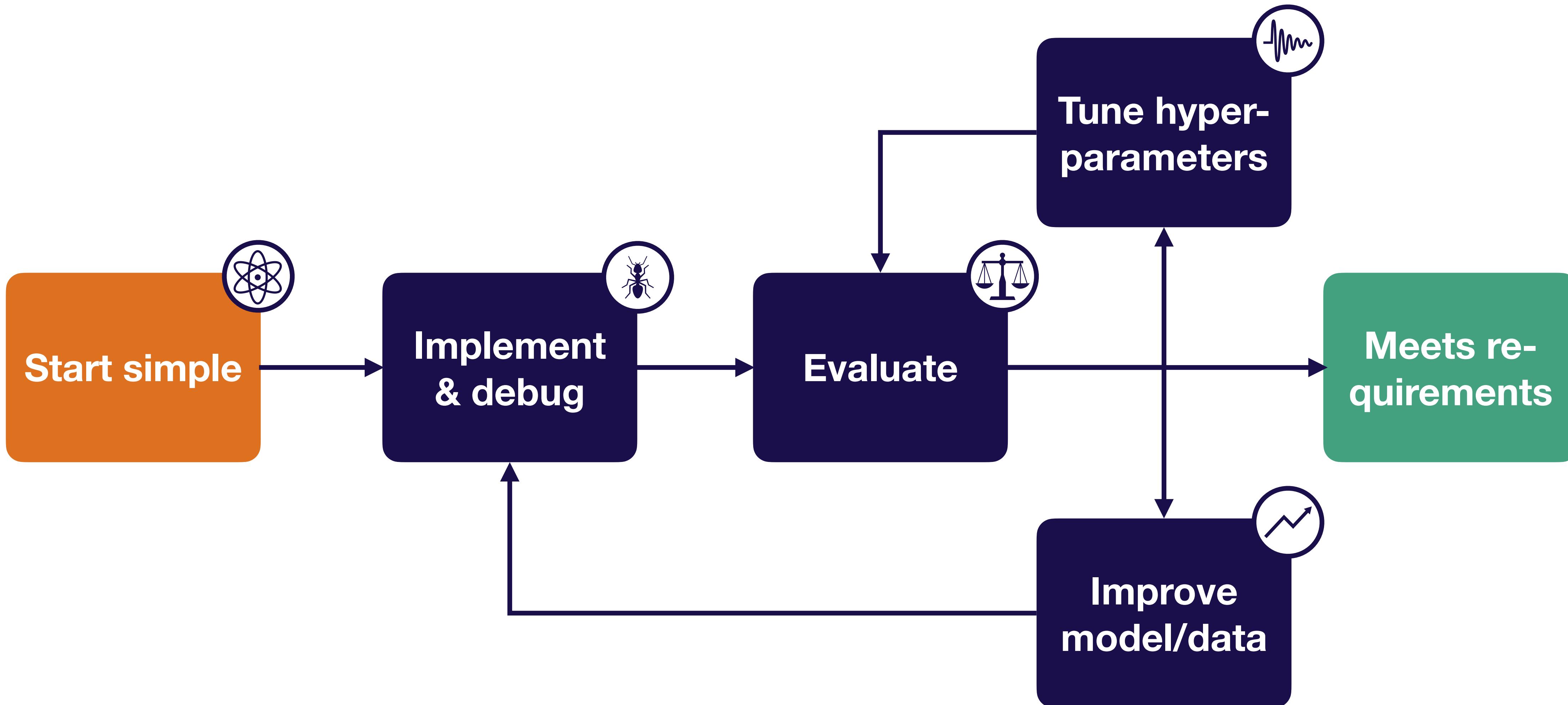


0 (no pedestrian) **1 (yes pedestrian)**

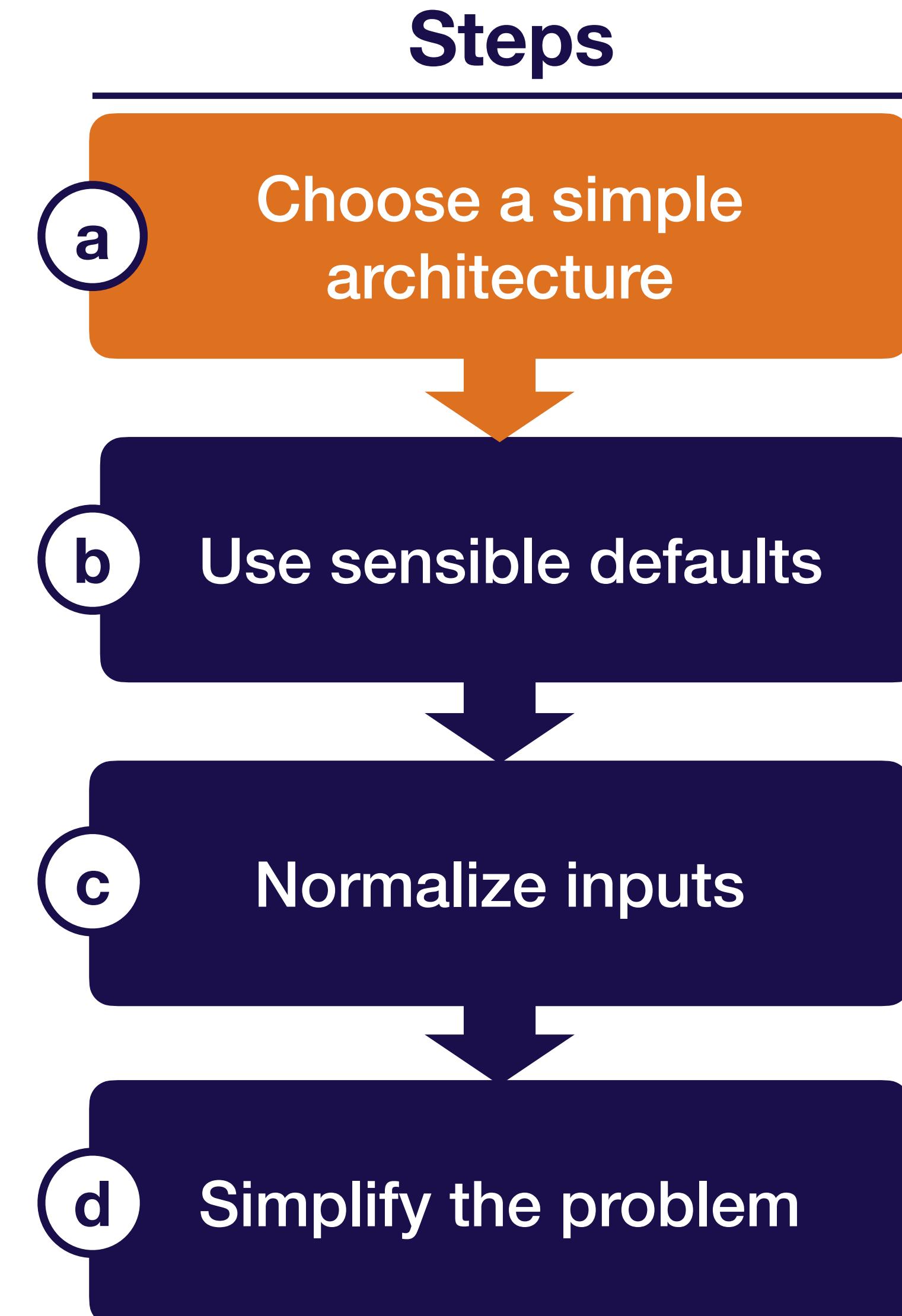
Goal: 99% classification accuracy

Questions?

Strategy for DL troubleshooting



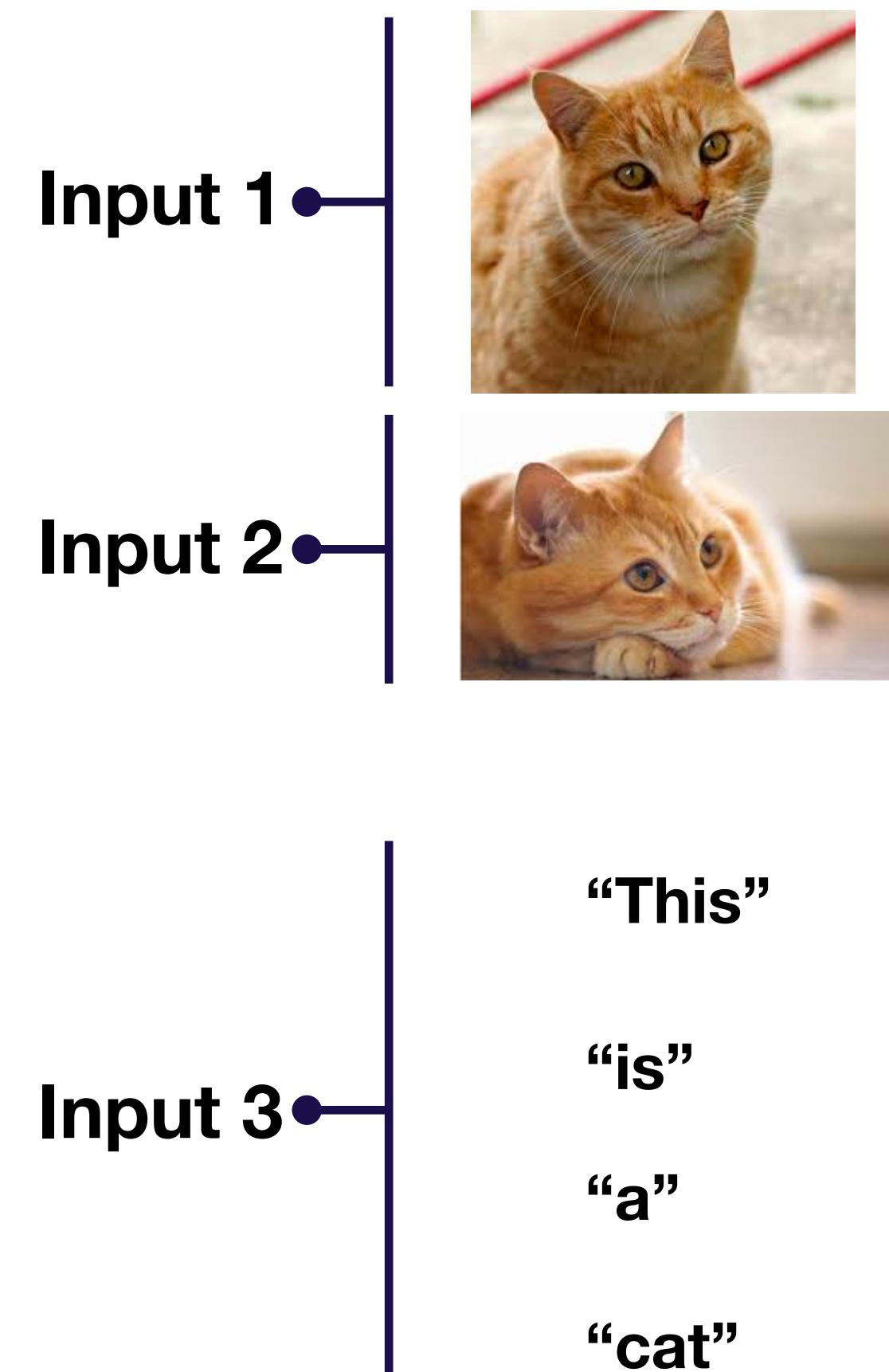
Starting simple



Demystifying architecture selection

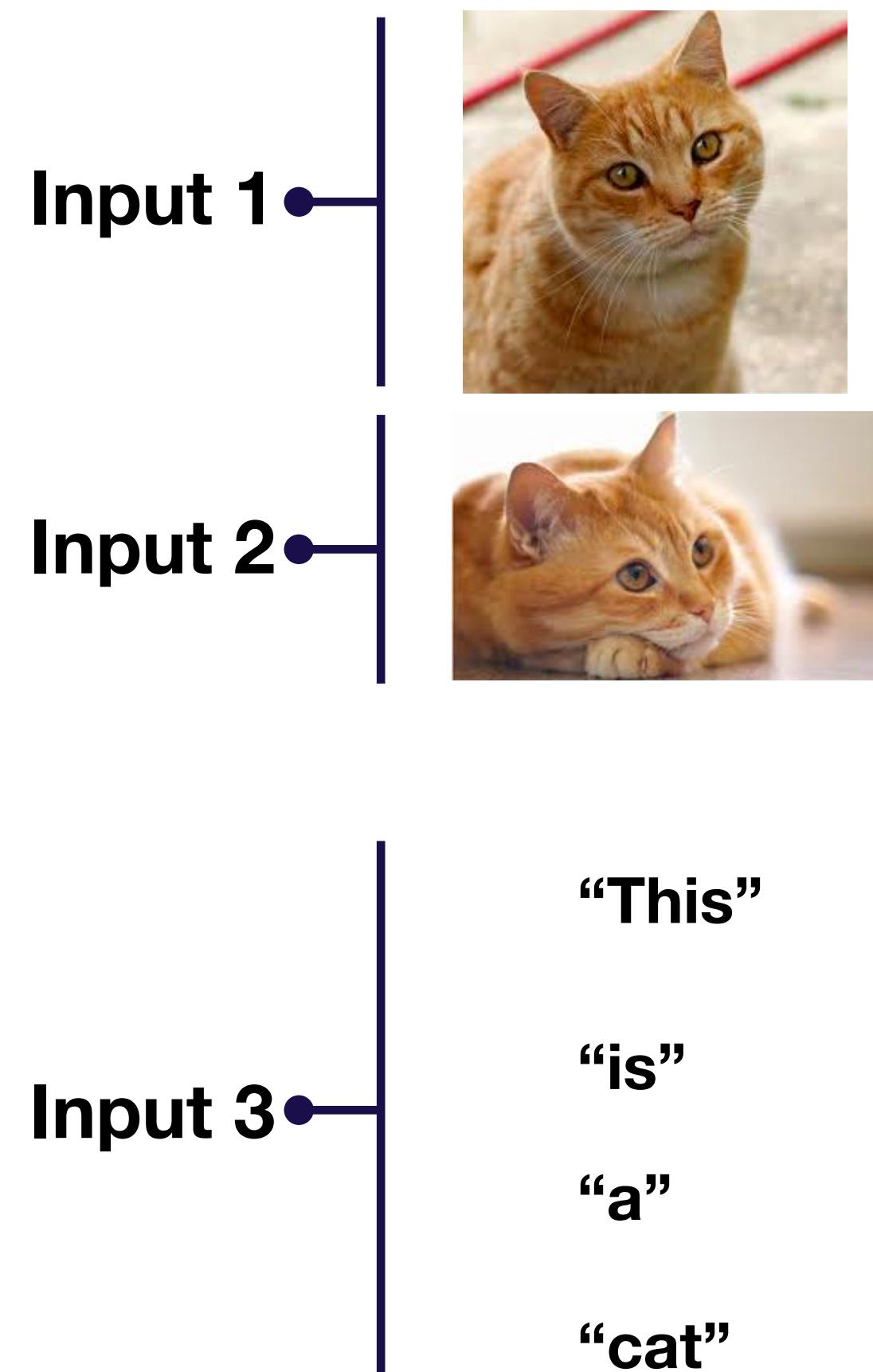
	Start here	Consider using this later
Images	LeNet-like architecture	ResNet
Sequences	LSTM with one hidden layer (or temporal convs)	Attention model or WaveNet-like model
Other	Fully connected neural net with one hidden layer	Problem-dependent

Dealing with multiple input modalities



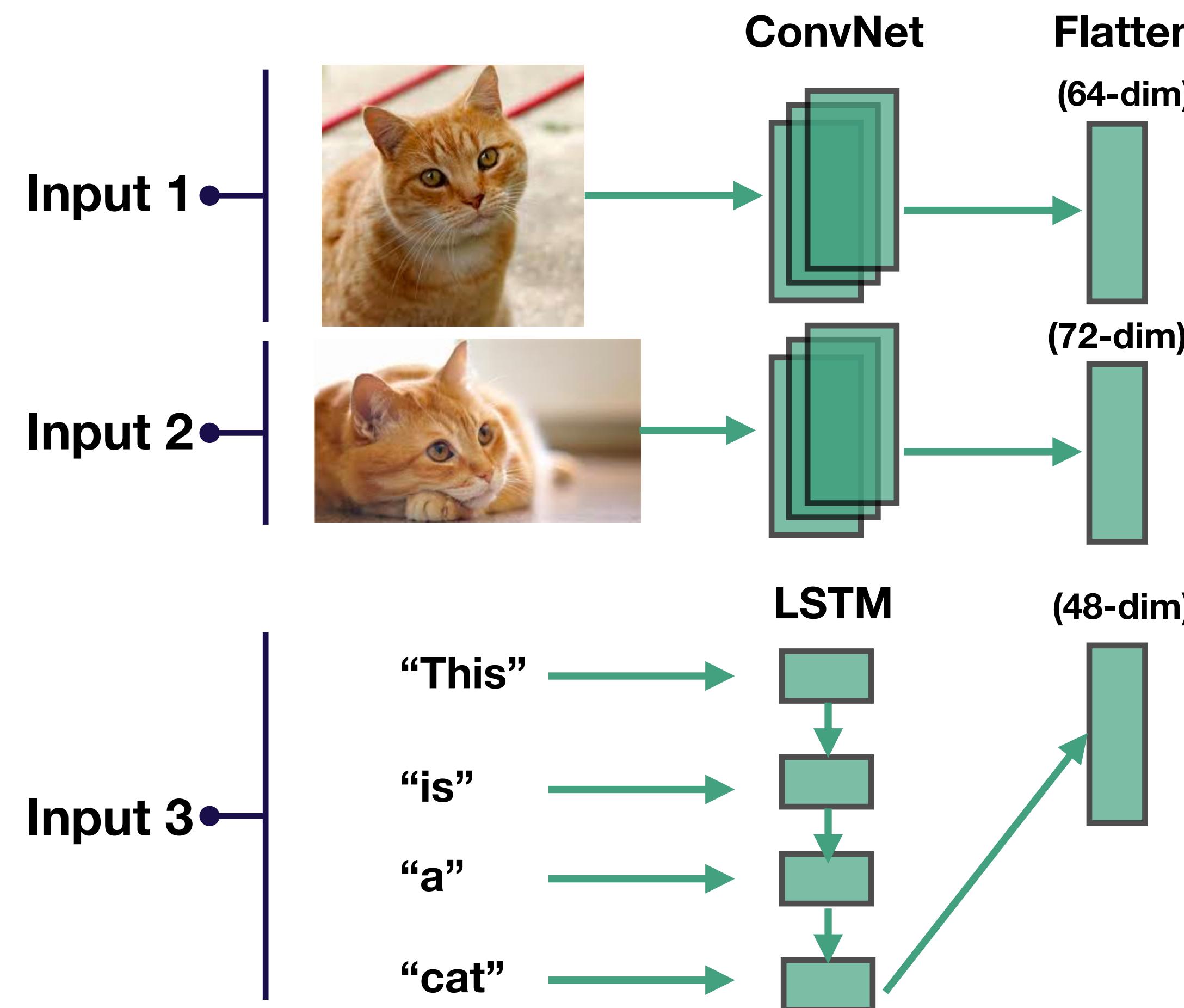
Dealing with multiple input modalities

1. Map each into a lower dimensional feature space



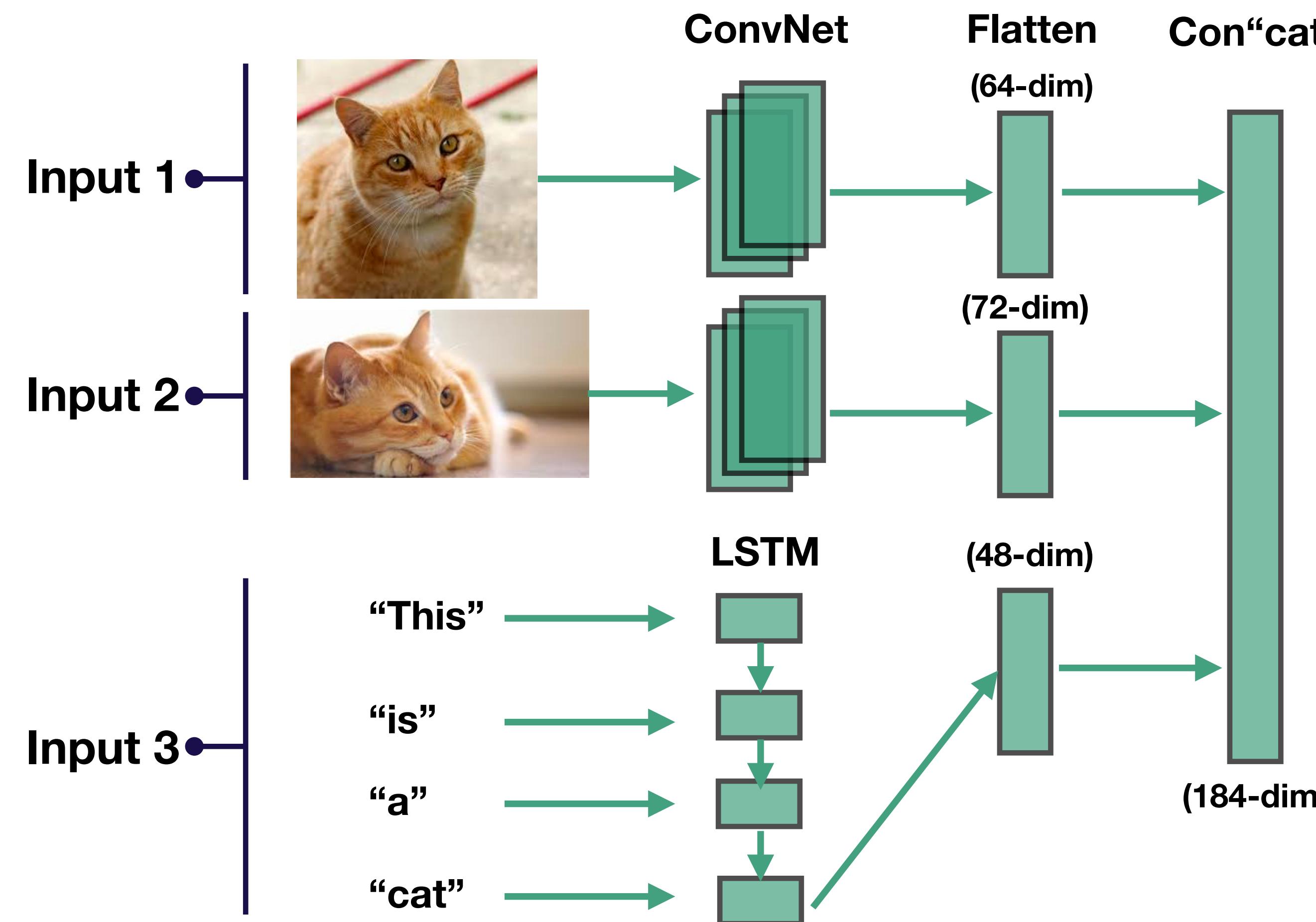
Dealing with multiple input modalities

1. Map each into a lower dimensional feature space



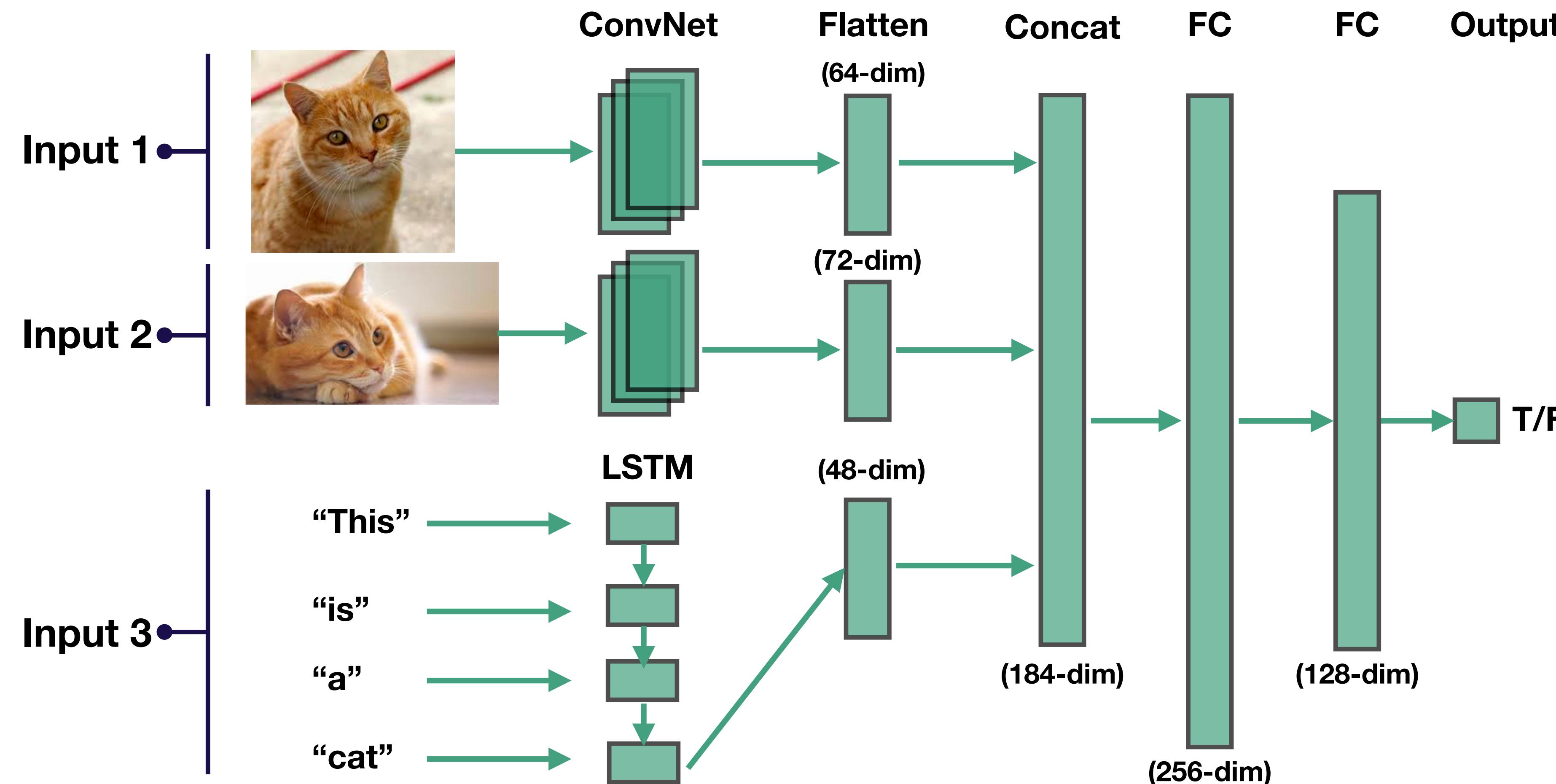
Dealing with multiple input modalities

2. Concatenate

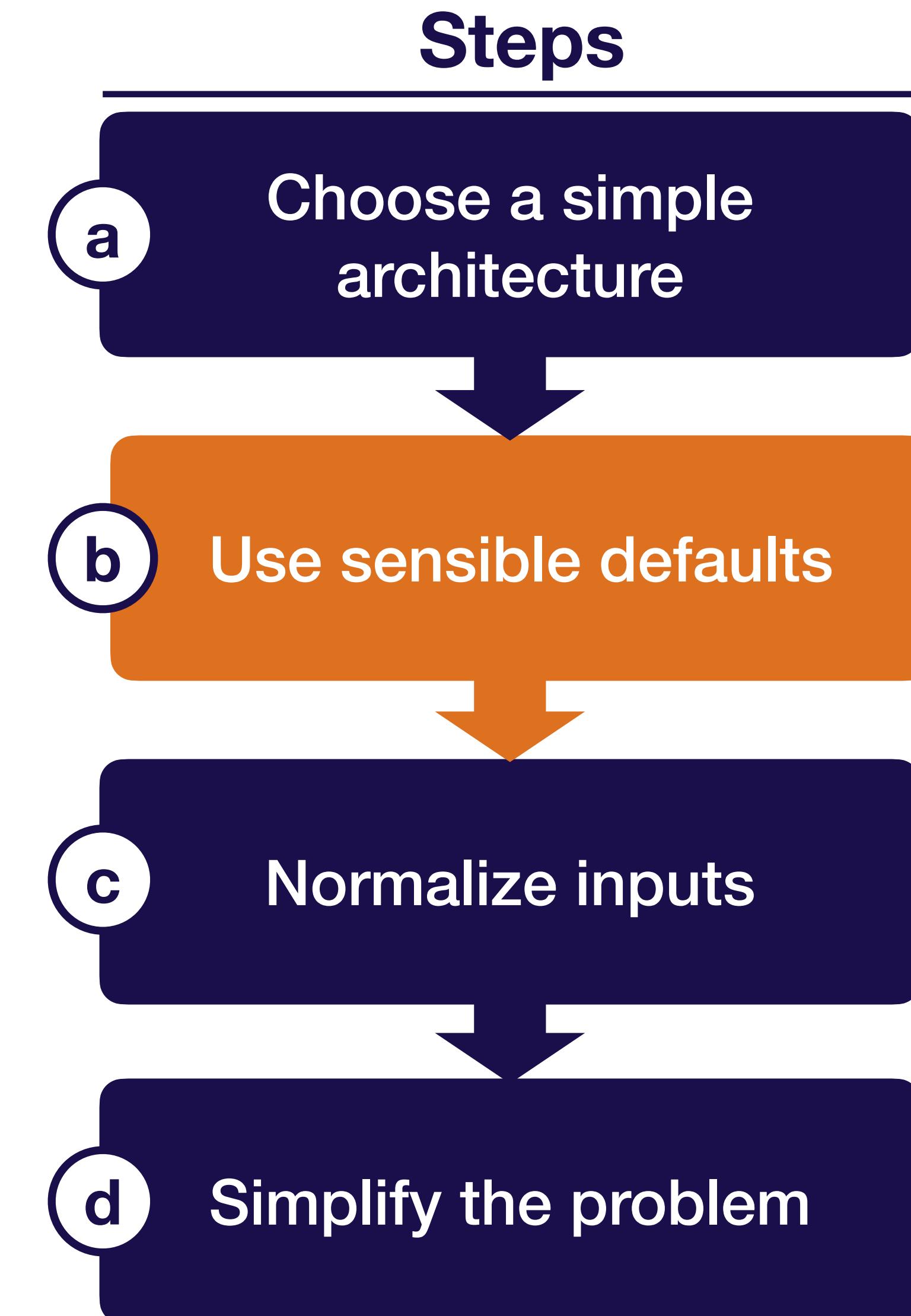


Dealing with multiple input modalities

3. Pass through fully connected layers to output



Starting simple



Recommended network / optimizer defaults

- **Optimizer:** Adam optimizer with learning rate 3e-4
- **Activations:** relu (FC and Conv models), tanh (LSTMs)
- **Initialization:** He et al. normal (relu), Glorot normal (tanh)
- **Regularization:** None
- **Data normalization:** None

Definitions of recommended initializers

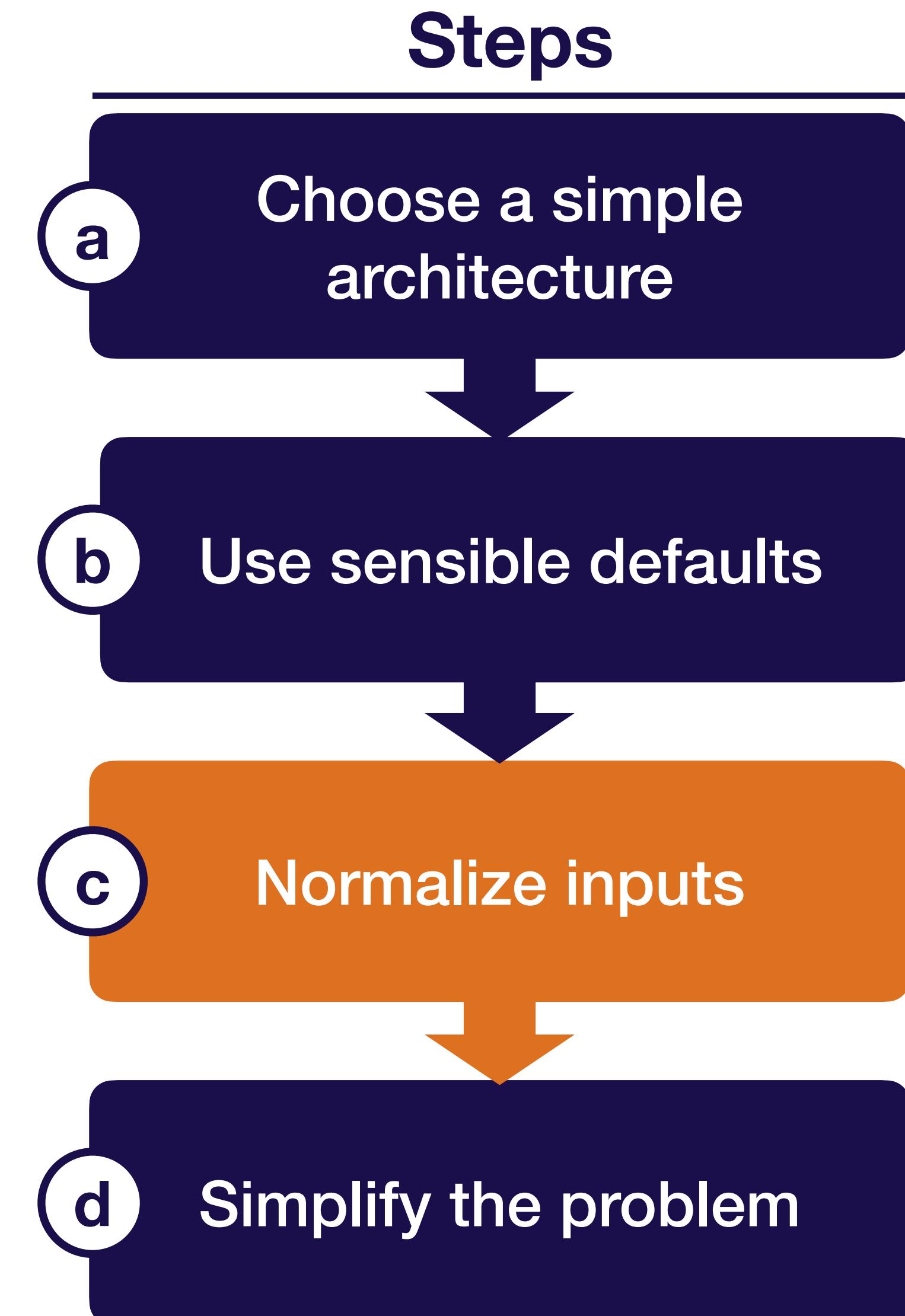
- (n is the number of inputs, m is the number of outputs)
- He et al. normal (used for ReLU)

$$\mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

- Glorot normal (used for tanh)

$$\mathcal{N}\left(0, \sqrt{\frac{2}{n+m}}\right)$$

Starting simple

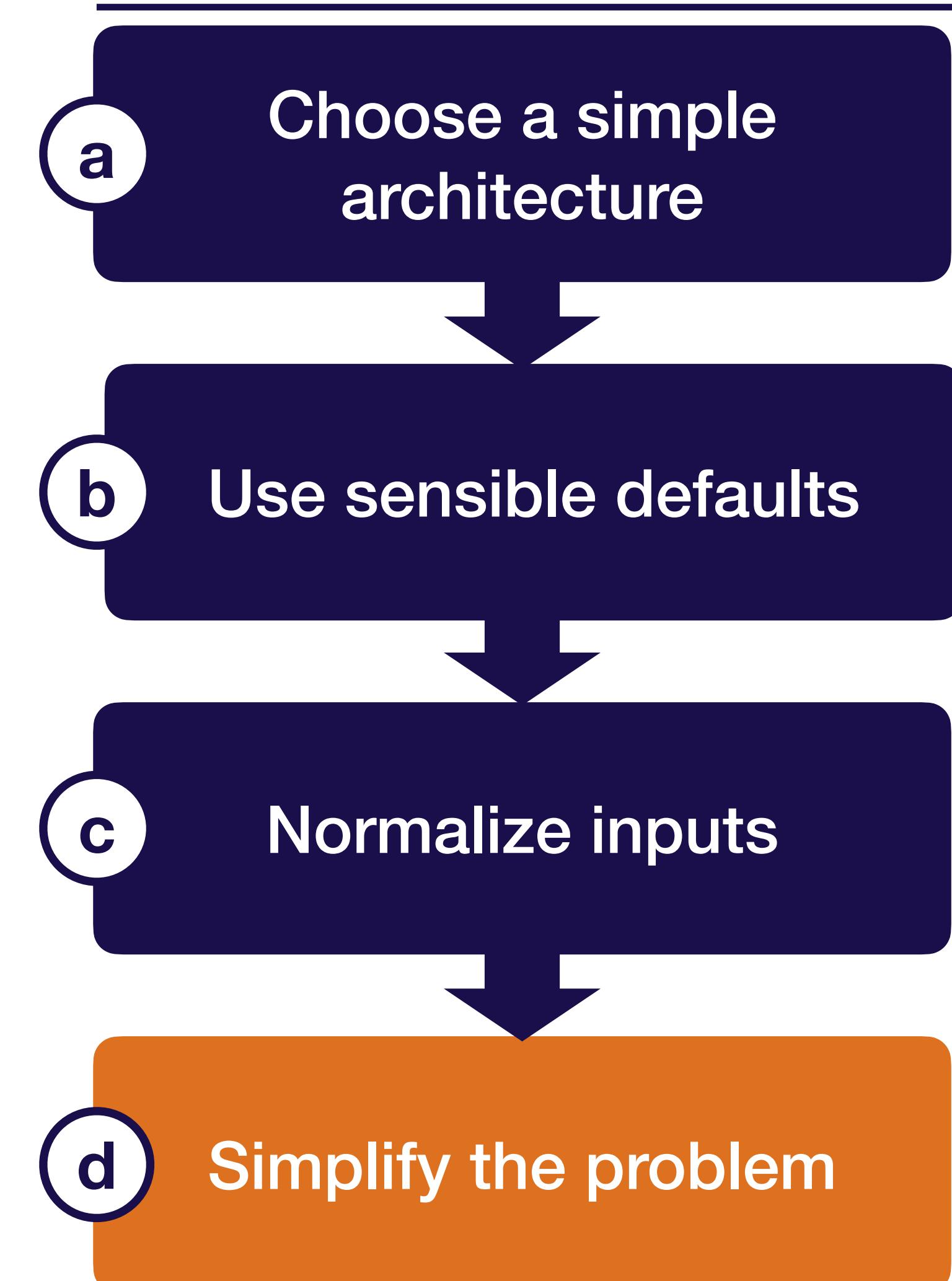


Important to normalize scale of input data

- Subtract mean and divide by variance
- For images, fine to scale values to [0, 1] or [-0.5, 0.5]
(e.g., by dividing by 255)
[Careful, make sure your library doesn't do it for you!]

Starting simple

Steps



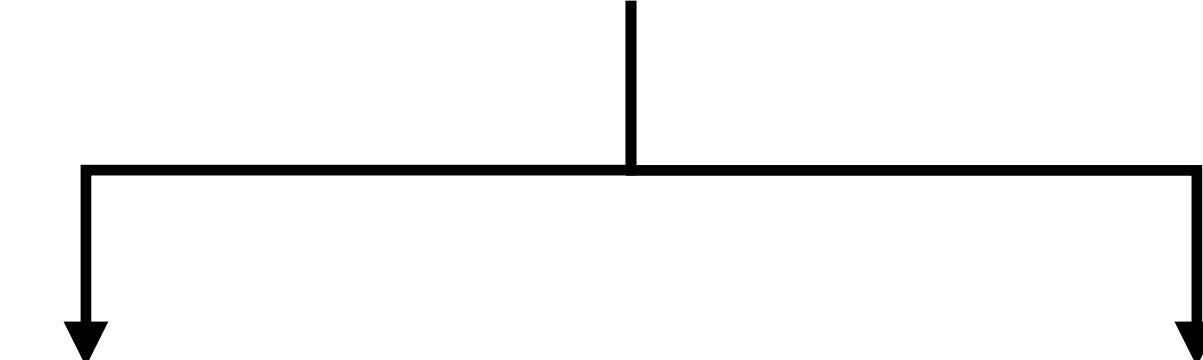
Consider simplifying the problem as well

- Start with a small training set (~10,000 examples)
- Use a fixed number of objects, classes, image size, etc.
- Create a simpler synthetic training set

Simplest model for pedestrian detection

Running example

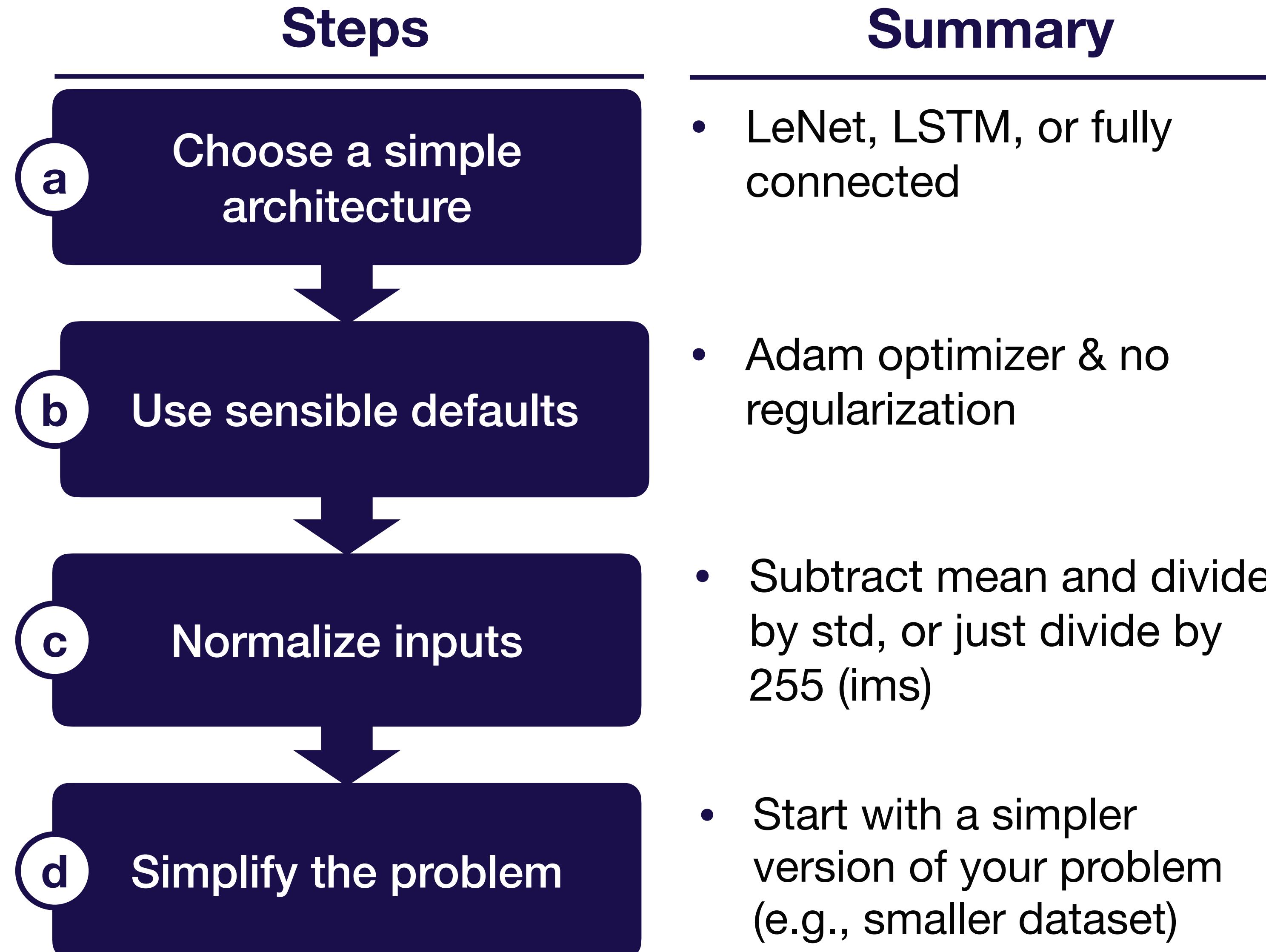
- Start with a subset of 10,000 images for training for test
- Use a LeNet architecture with sigmoid cross-entropy loss
- Adam optimizer with LR 3e-4
- No regularization



0 (no pedestrian) **1 (yes pedestrian)**

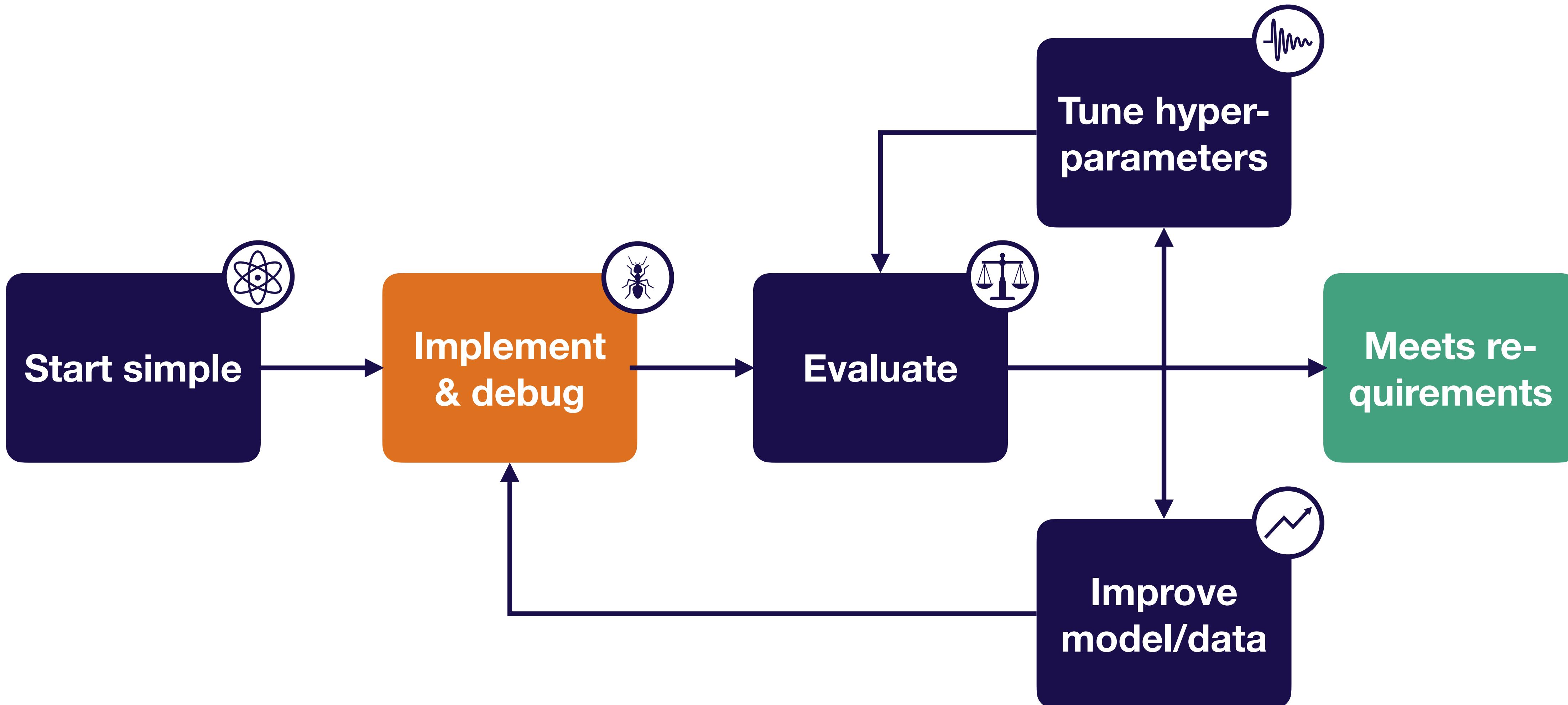
Goal: 99% classification accuracy

Starting simple

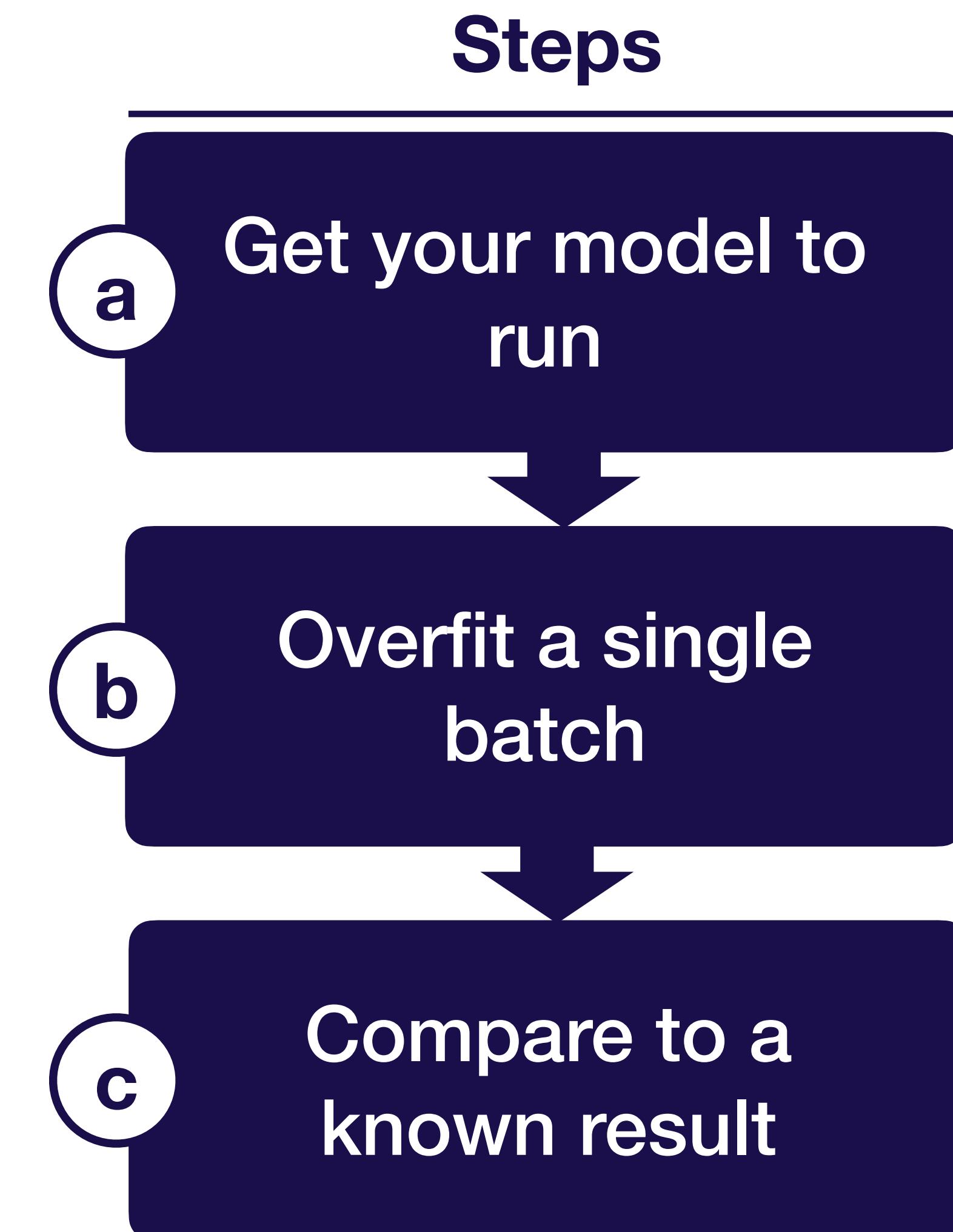


Questions?

Strategy for DL troubleshooting



Implementing bug-free DL models



Preview: the five most common DL bugs

- **Incorrect shapes for your tensors**
 - Can fail silently! E.g., accidental broadcasting: `x.shape = (None,), y.shape = (None, 1), (x+y).shape = (None, None)`
- **Pre-processing inputs incorrectly**
 - E.g., Forgetting to normalize, or too much pre-processing
- **Incorrect input to your loss function**
 - E.g., softmaxed outputs to a loss that expects logits
- **Forgot to set up train mode for the net correctly**
 - E.g., toggling train/eval, controlling batch norm dependencies
- **Numerical instability - inf/NaN**
 - Often stems from using an exp, log, or div operation



General advice for implementing your model

Lightweight implementation

- Minimum possible new lines of code for v1
- Rule of thumb: <200 lines
- (Tested infrastructure components are fine)

Use off-the-shelf components, e.g.,

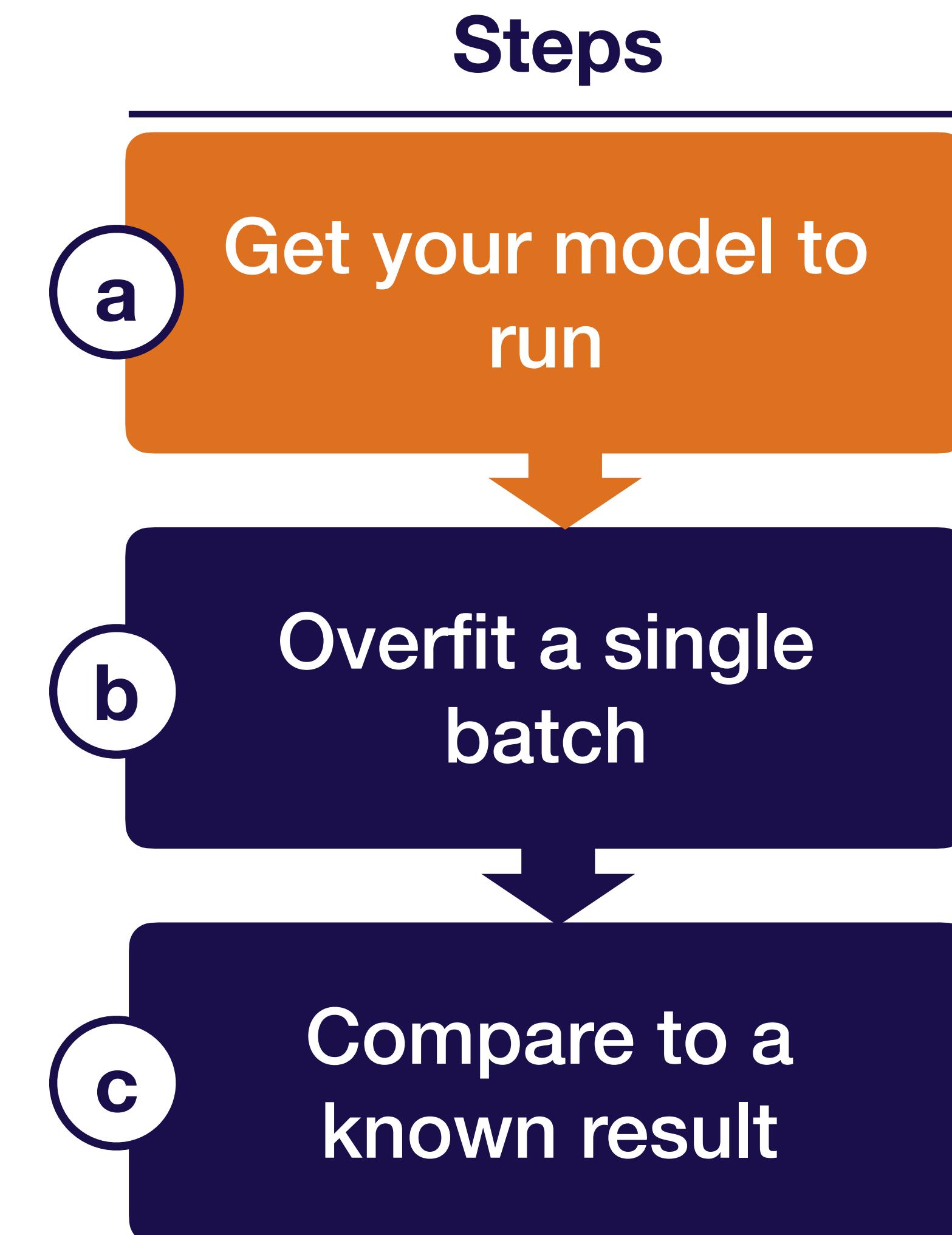
- Keras
- `tf.layers.dense(...)` instead of `tf.nn.relu(tf.matmul(W, x))`
- `tf.losses.cross_entropy(...)` instead of writing out the exp

Build complicated data pipelines later

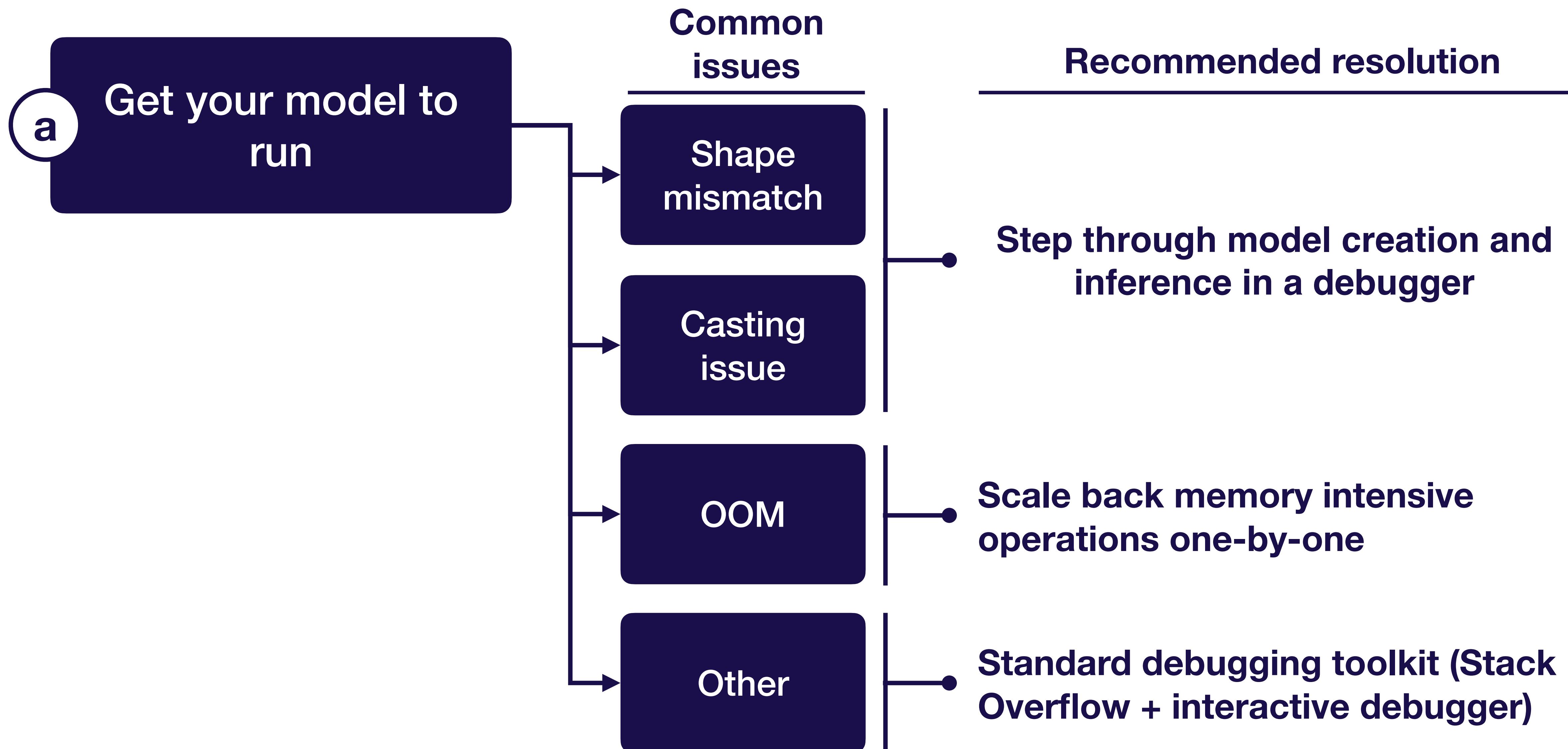
- Start with a dataset you can load into memory



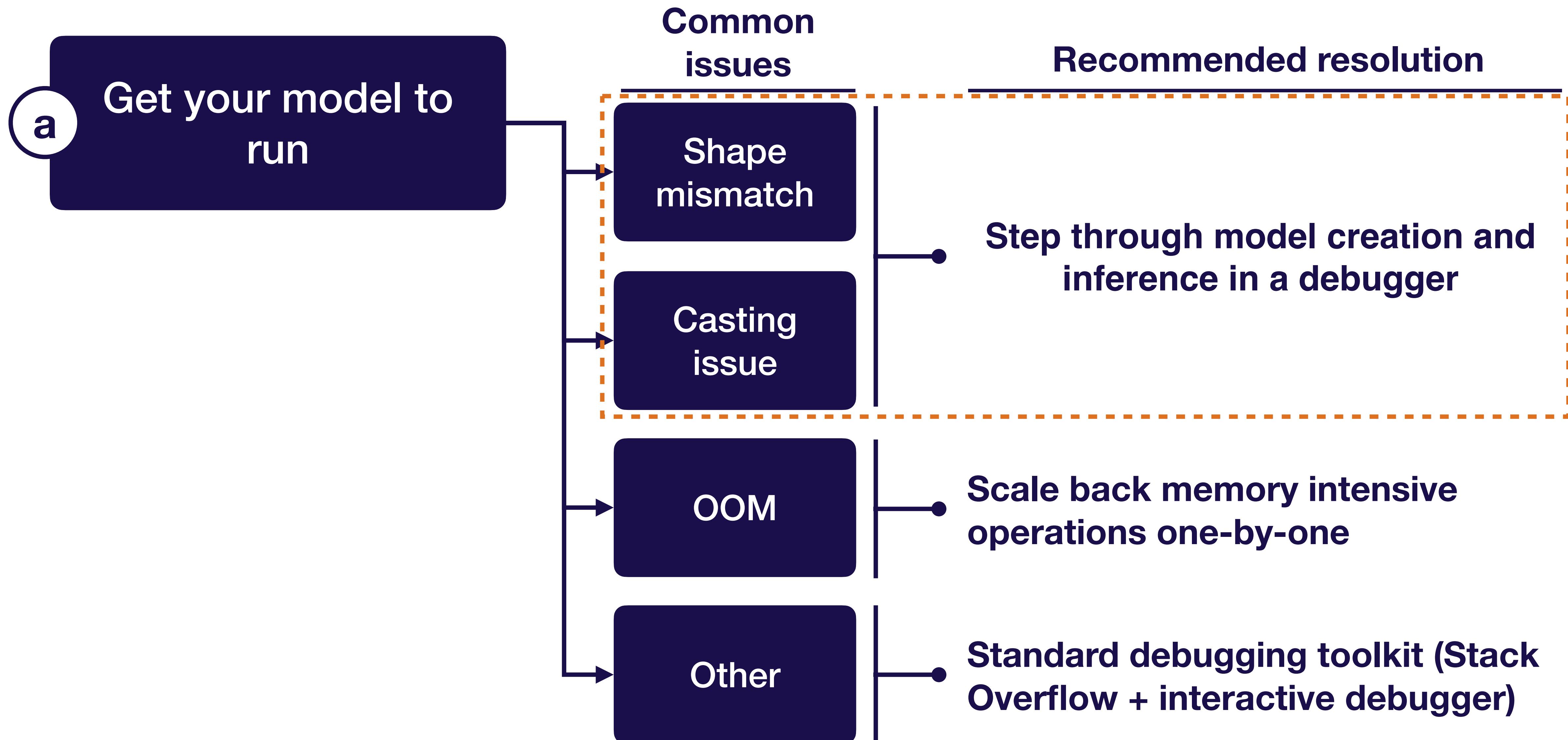
Implementing bug-free DL models



Implementing bug-free DL models



Implementing bug-free DL models



Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 1: step through graph creation

```
2 # Option 1: step through graph creation
3 import ipdb; ipdb.set_trace()
4
5 for i in range(num_layers):
6     out = layers.fully_connected(out, 50)
7
```

```
josh at MacBook-Pro-9 in ~/projects
$ python test.py
> /Users/josh/projects/test.py(5)<module>()
    3 h = tf.placeholder(tf.float32, (None, 100))
    4 import ipdb; ipdb.set_trace()
----> 5 w = tf.layers.dense(h)

ipdb>
```



Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 2: step into training loop

```
9 # Option 2: step into training loop
10 sess = tf.Session()
11 for i in range(num_epochs):
12     import ipdb; ipdb.set_trace()
13     loss_, _ = sess.run([loss, train_op])
14
```



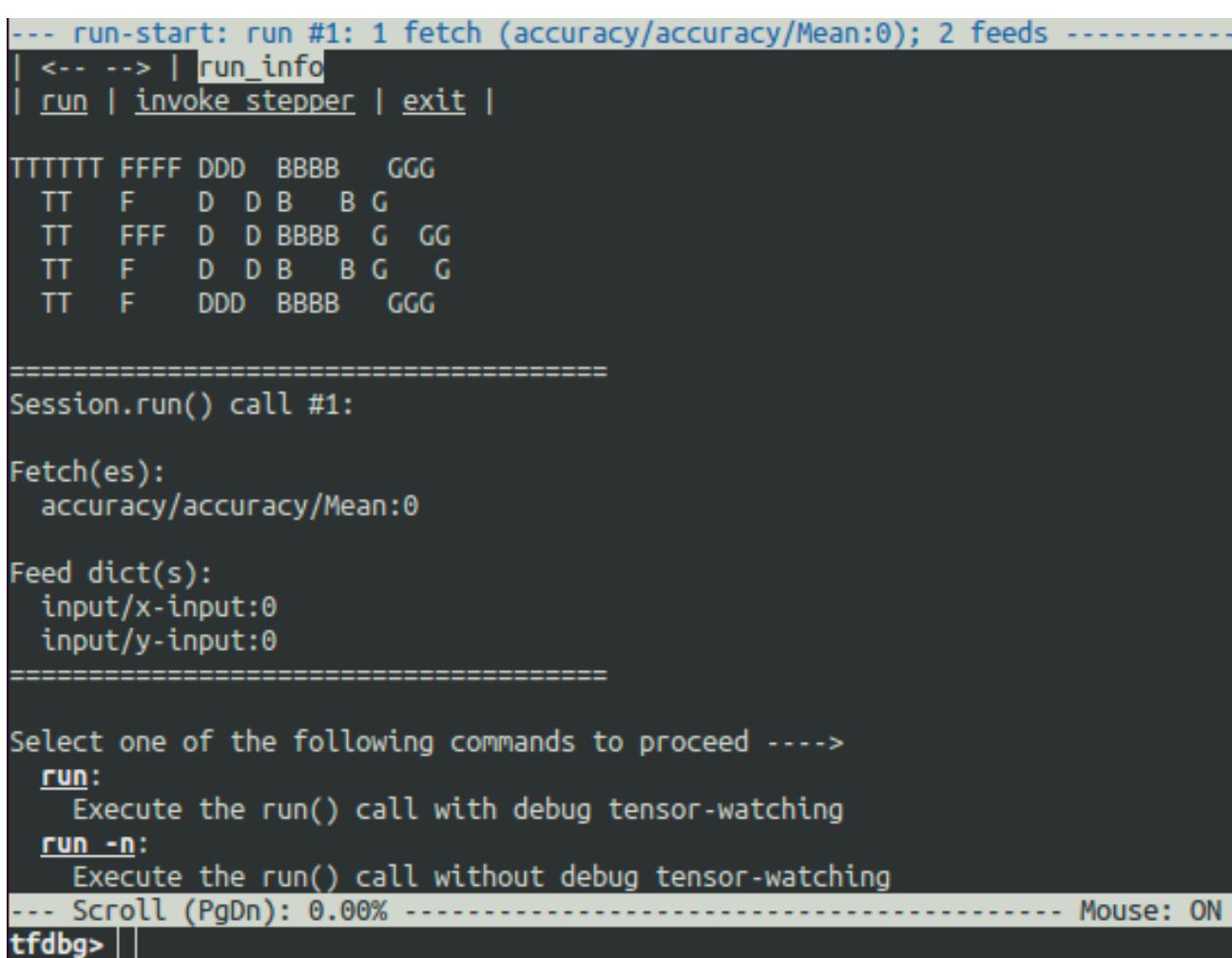
Evaluate tensors using `sess.run(...)`

Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 3: use tfdb

```
python -m tensorflow.python.debug.examples.debug_mnist --debug
```

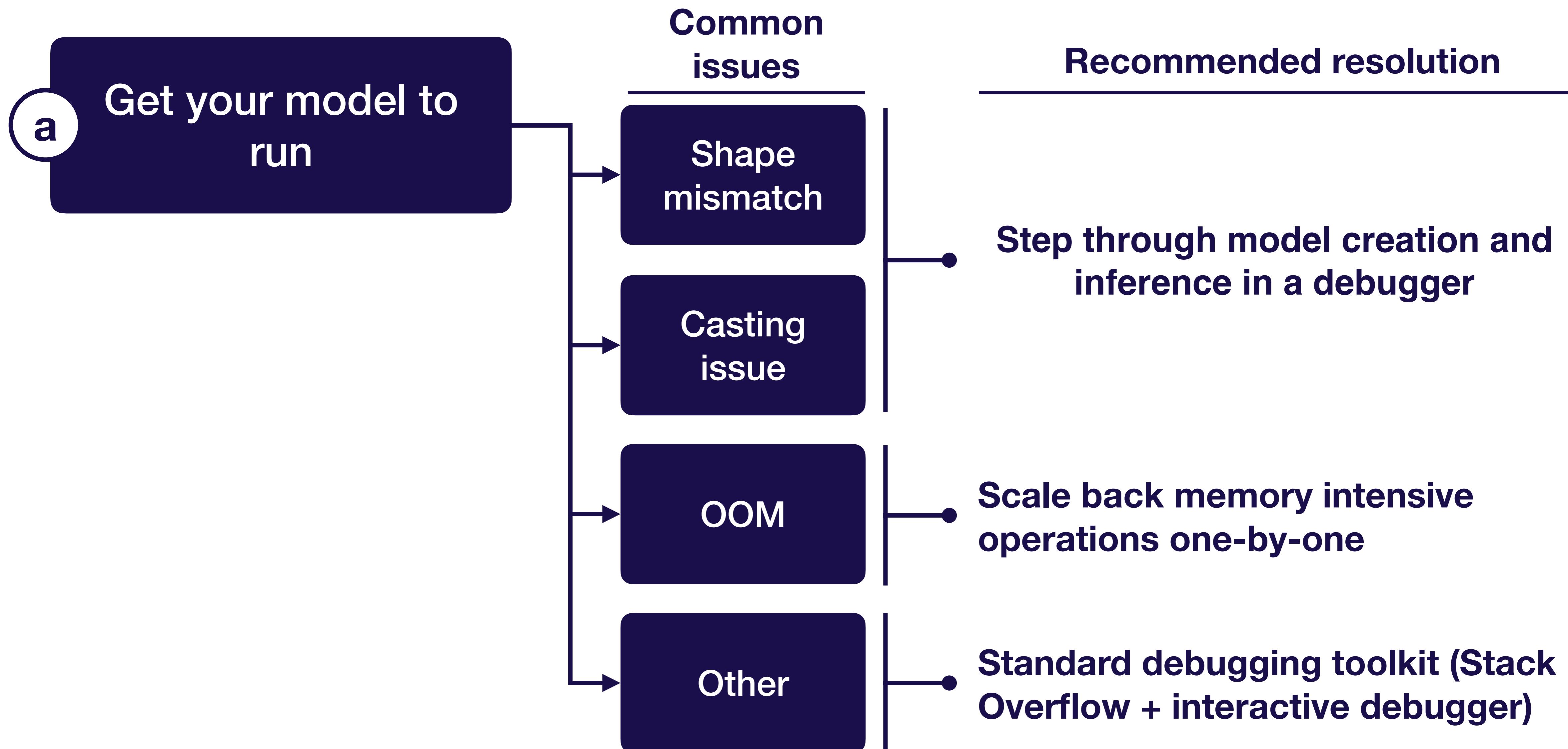


```
-- run-start: run #1: 1 fetch (accuracy/accuracy/Mean:0); 2 feeds -----
| <-- --> | run_info
| run | invoke stepper | exit |
TTTTTT FFFF DDD BBBB GGG
TT F D D B B G
TT FFF D D BBBB G GG
TT F D D B B G G
TT F DDD BBBB GGG
=====
Session.run() call #1:
Fetch(es):
  accuracy/accuracy/Mean:0
Feed dict(s):
  input/x-input:0
  input/y-input:0
=====
Select one of the following commands to proceed ---->
  run:
    Execute the run() call with debug tensor-watching
  run -n:
    Execute the run() call without debug tensor-watching
-- Scroll (PgDn): 0.00% ----- Mouse: ON -----
tfdbg> [ ]
```

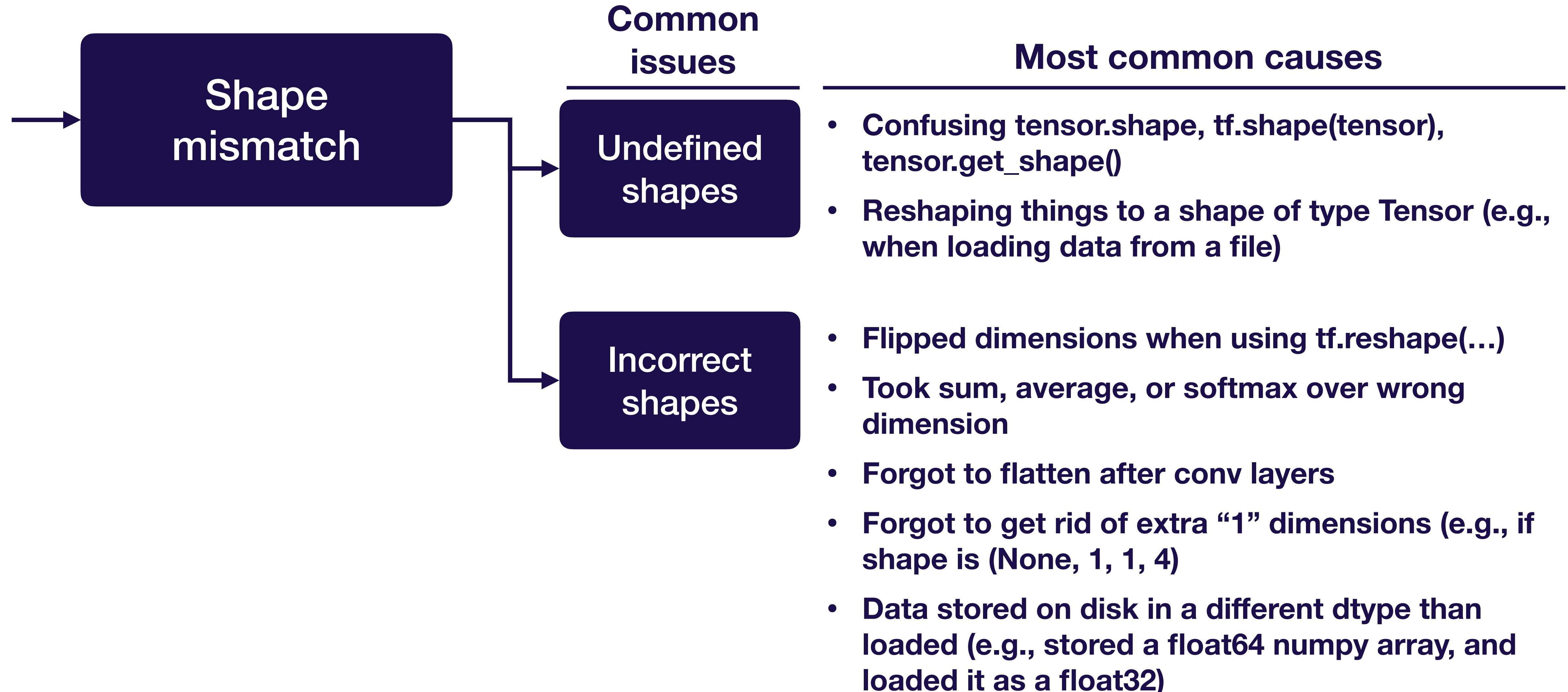
**Stops
execution at
each
sess.run(...)
and lets you
inspect**



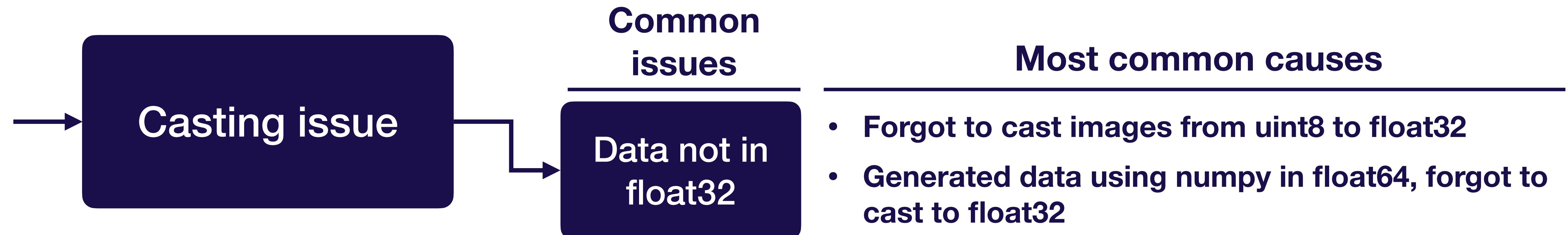
Implementing bug-free DL models



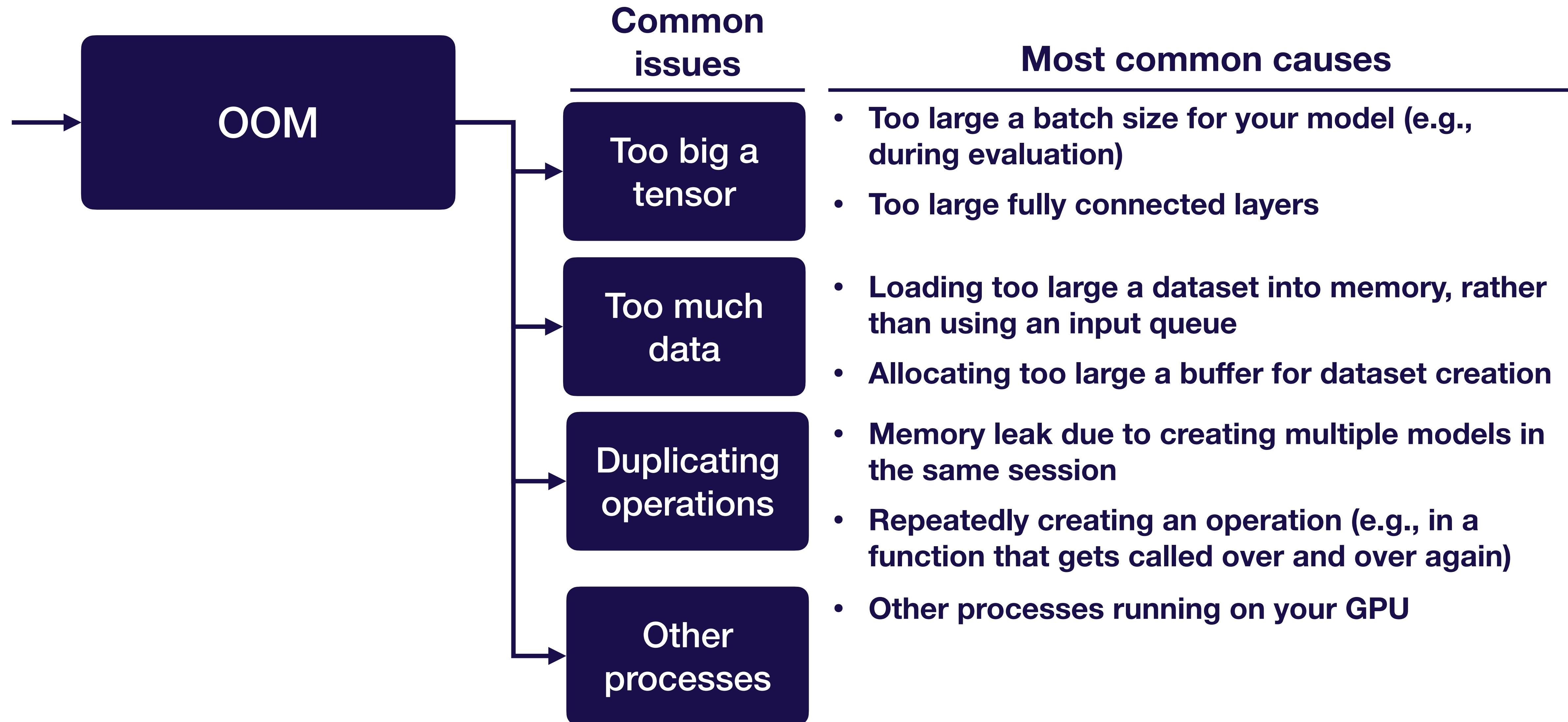
Implementing bug-free DL models



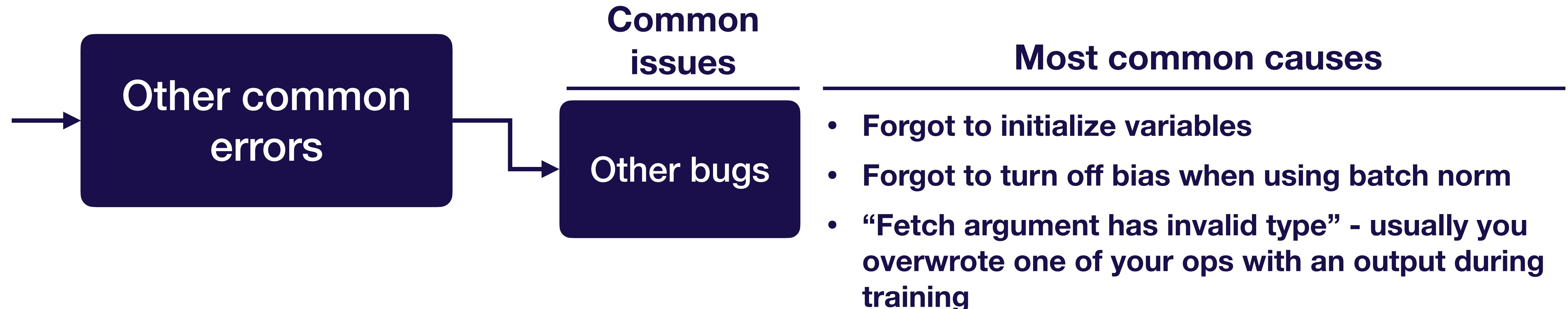
Implementing bug-free DL models



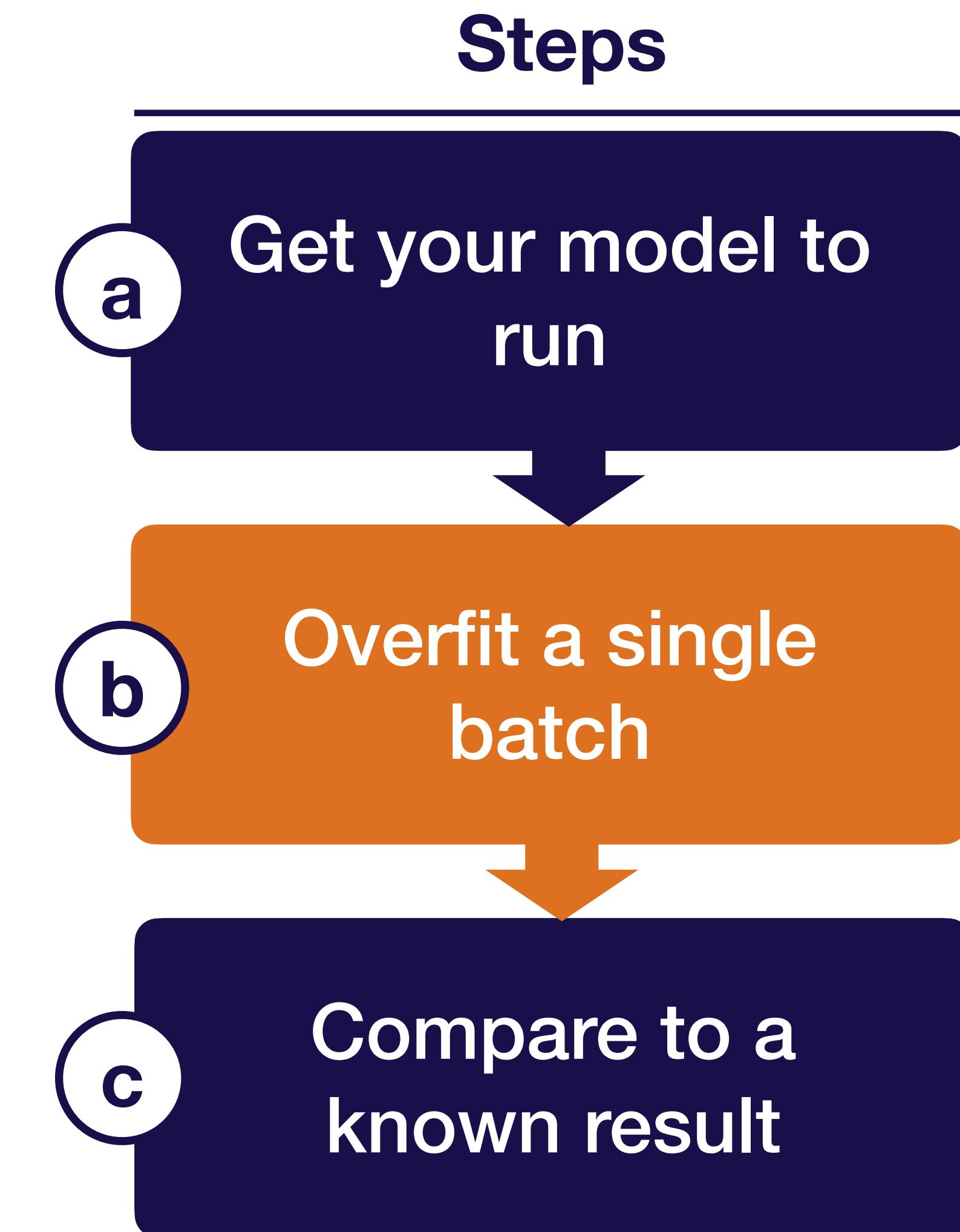
Implementing bug-free DL models



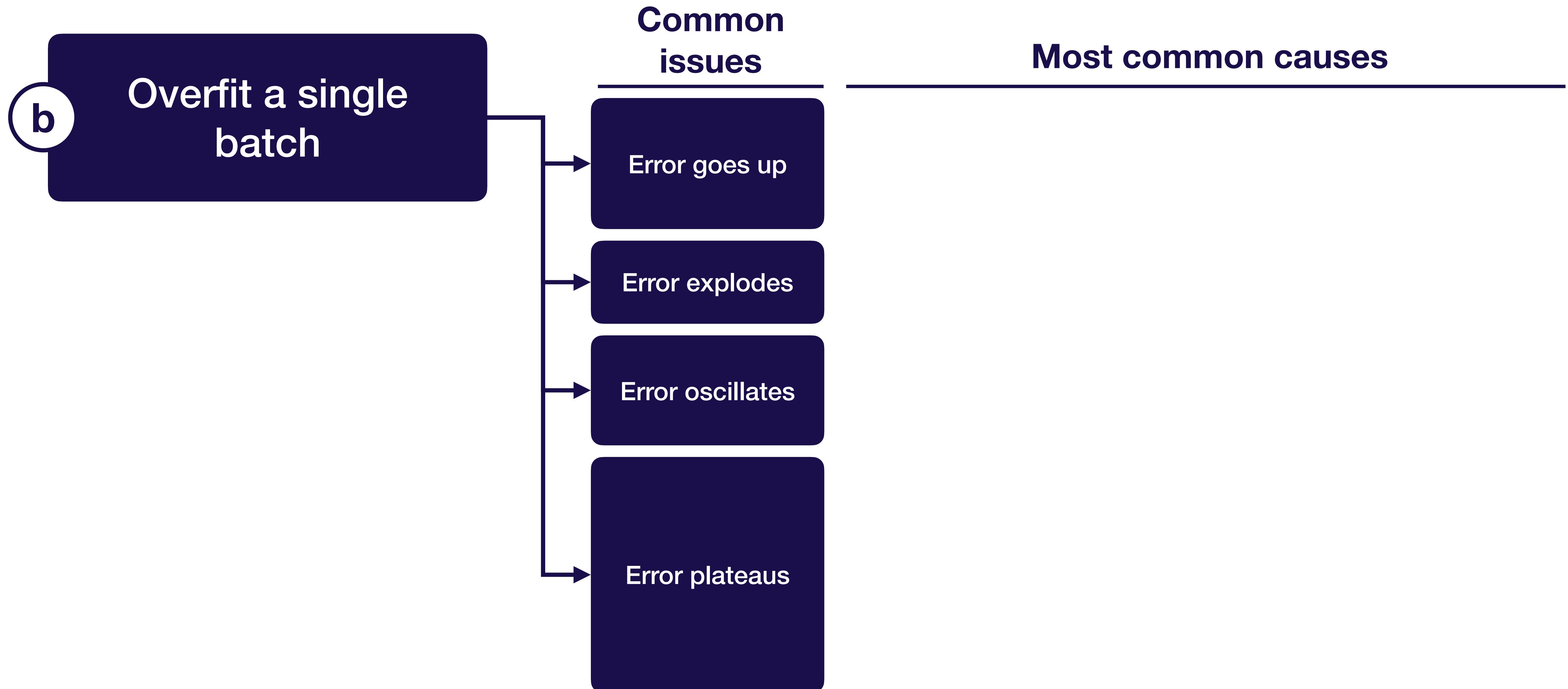
Implementing bug-free DL models



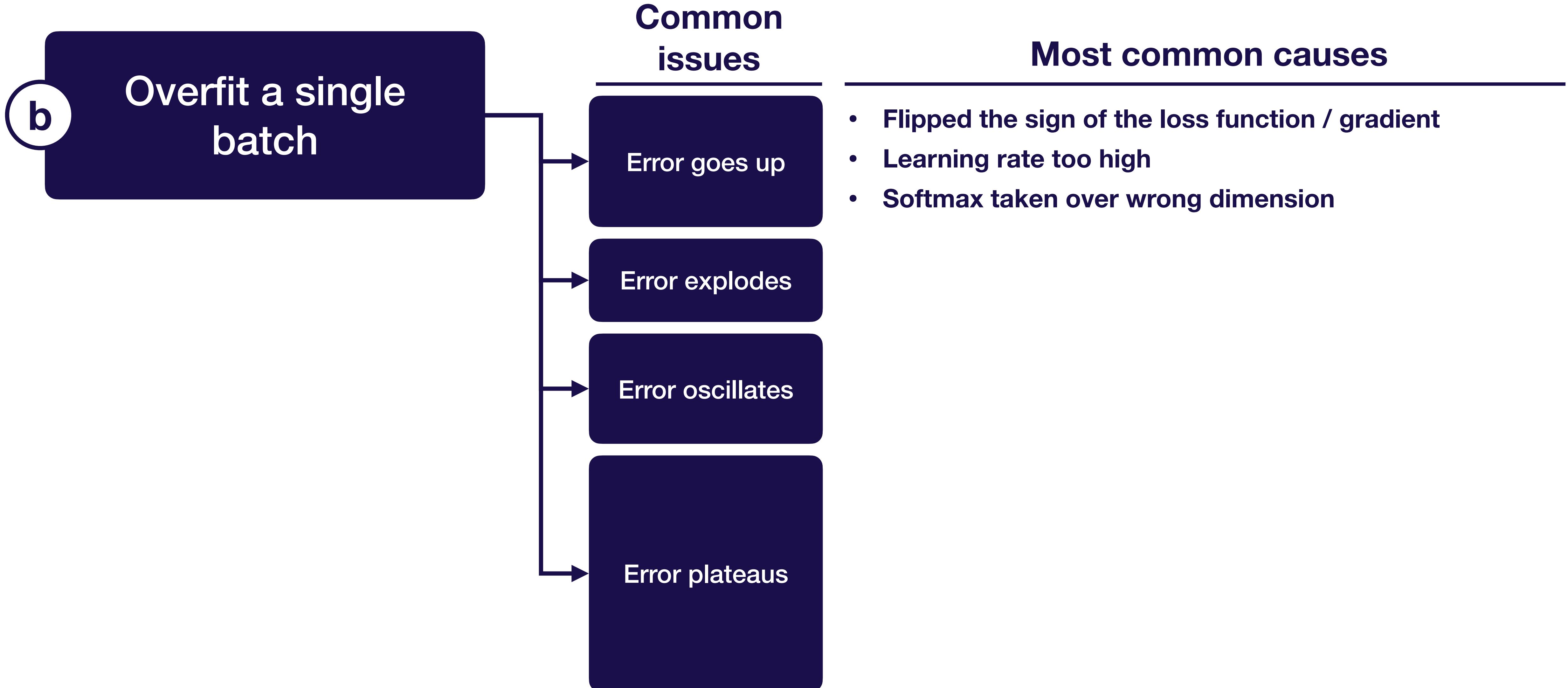
Implementing bug-free DL models



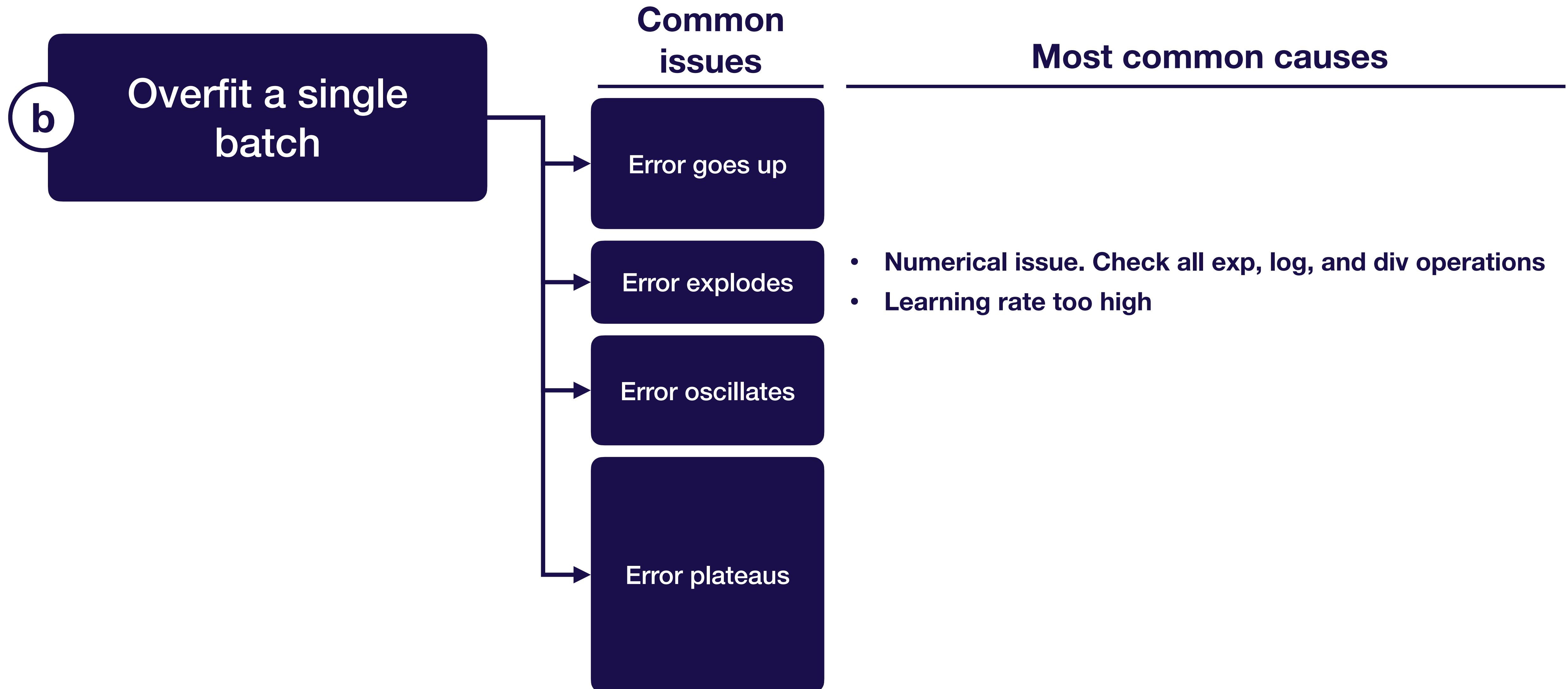
Implementing bug-free DL models



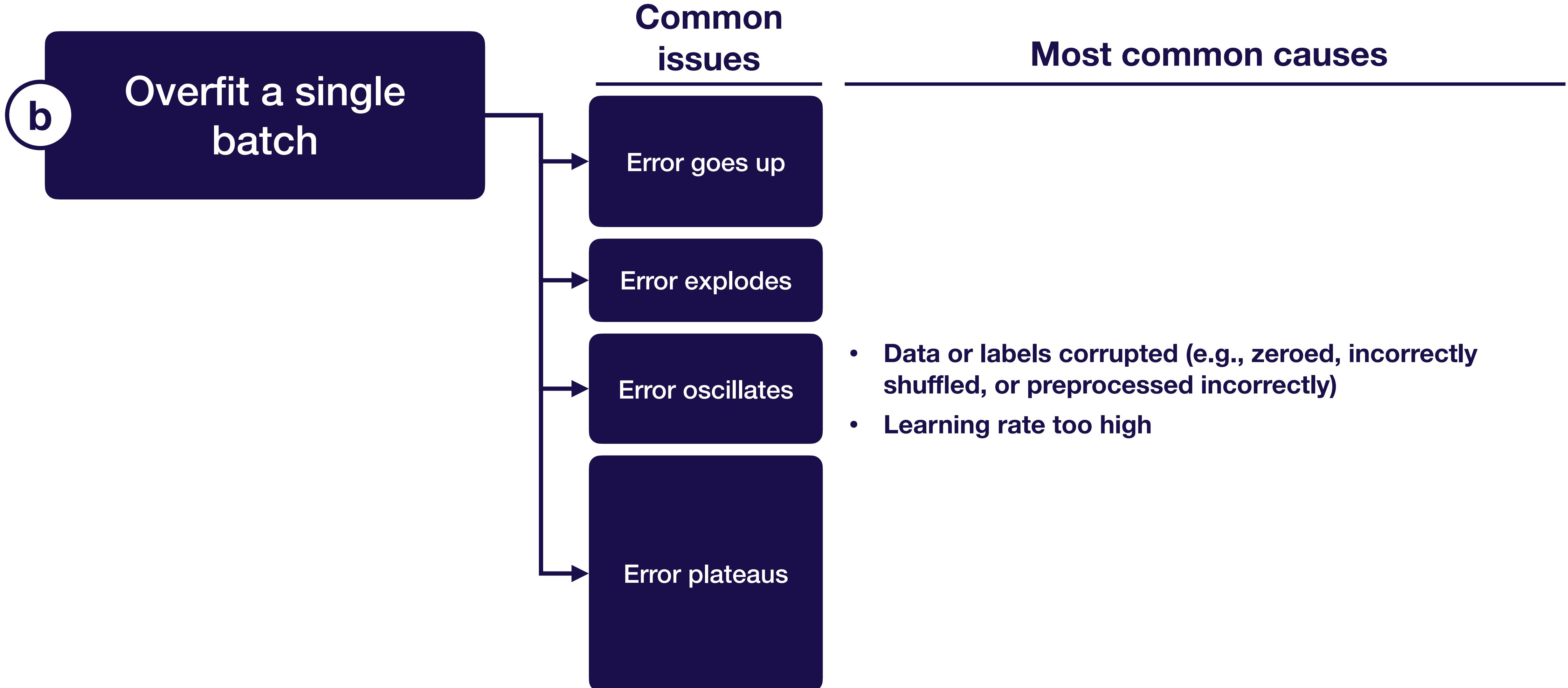
Implementing bug-free DL models



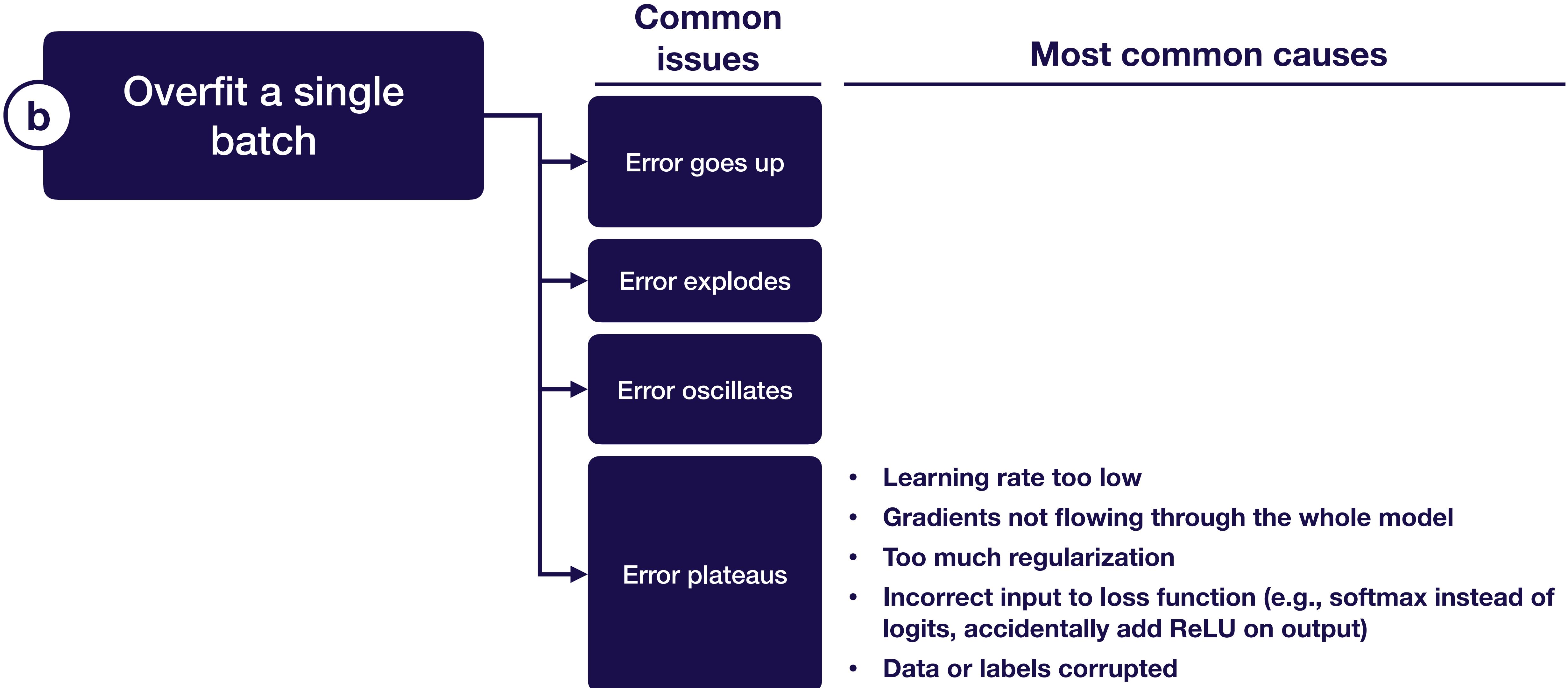
Implementing bug-free DL models



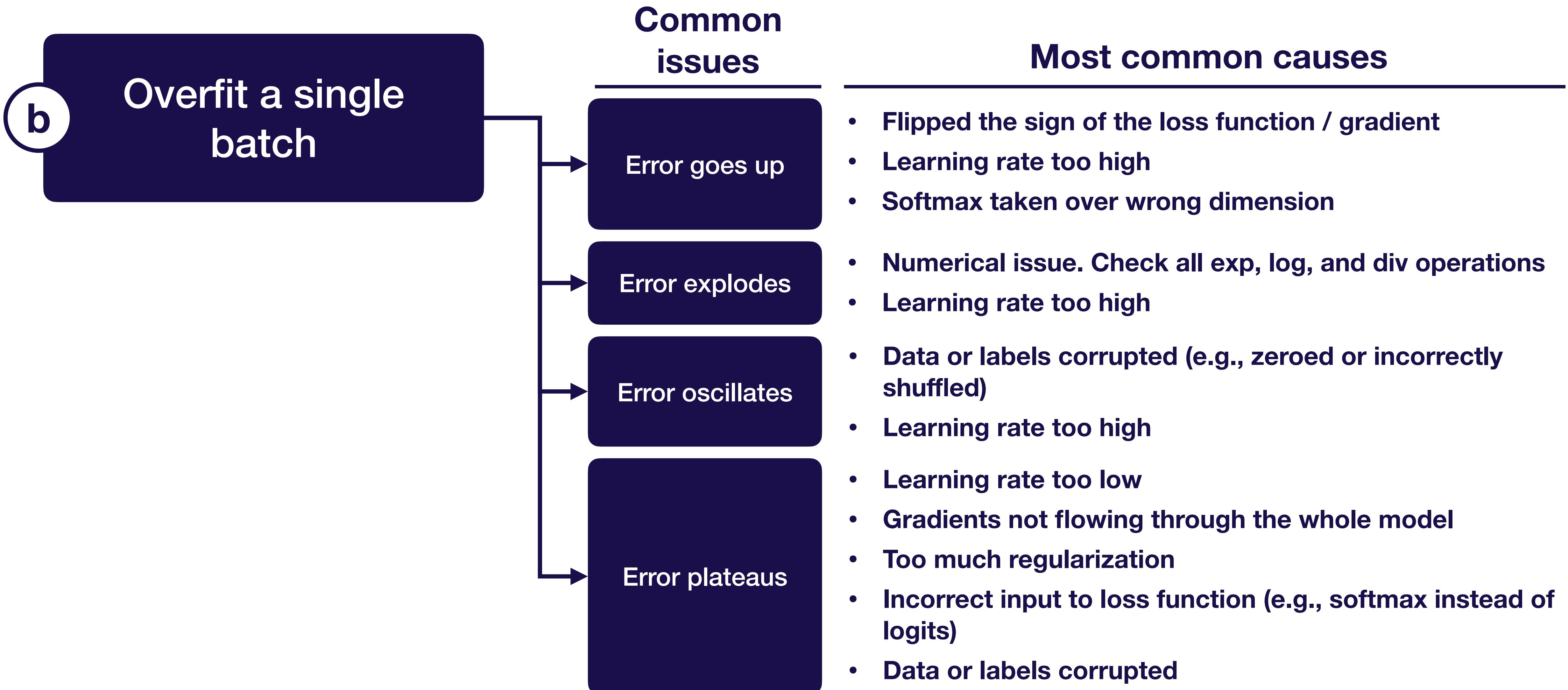
Implementing bug-free DL models



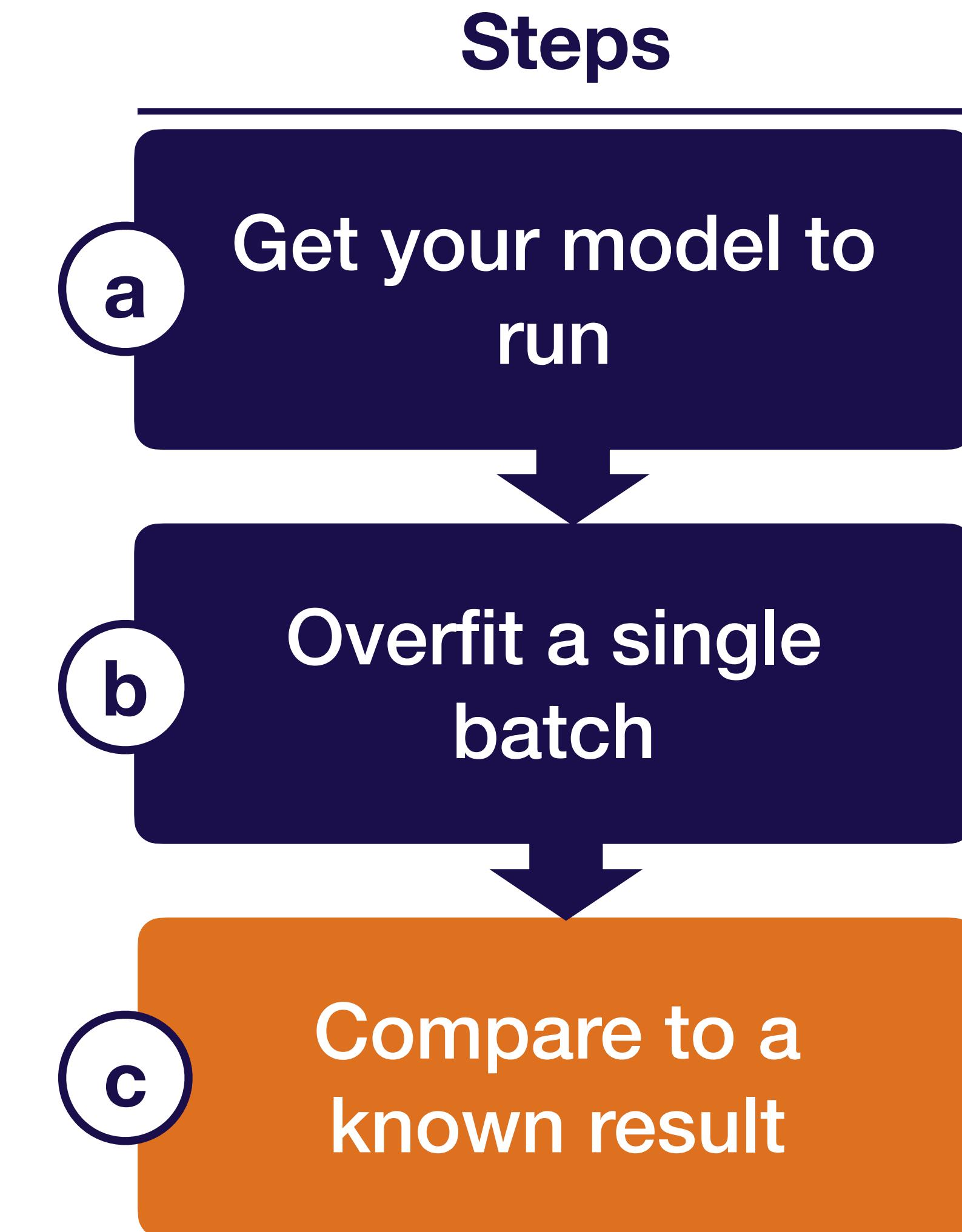
Implementing bug-free DL models



Implementing bug-free DL models



Implementing bug-free DL models



Hierarchy of known results

More useful

- Official model implementation evaluated on similar dataset to yours

You can:

- Walk through code line-by-line and ensure you have the same output
- Ensure your performance is up to par with expectations

Less useful



Hierarchy of known results

More useful

- Official model implementation evaluated on benchmark (e.g., MNIST)

You can:

- Walk through code line-by-line and ensure you have the same output

Less useful



Hierarchy of known results

More useful



- Unofficial model implementation

You can:

- Same as before, but with lower confidence

Less useful



Hierarchy of known results

More useful

- Results from a paper (with no code)

You can:

- Ensure your performance is up to par with expectations

Less useful



Hierarchy of known results

More useful

You can:

- Make sure your model performs well in a simpler setting
- Results from your model on a benchmark dataset (e.g., MNIST)

Less useful



Hierarchy of known results

More useful



Less useful

You can:

- Get a general sense of what kind of performance can be expected
- Results from a similar model on a similar dataset



Hierarchy of known results

More useful



Less useful

You can:

- Make sure your model is learning anything at all

- Super simple baselines (e.g., average of outputs or linear regression)



Hierarchy of known results

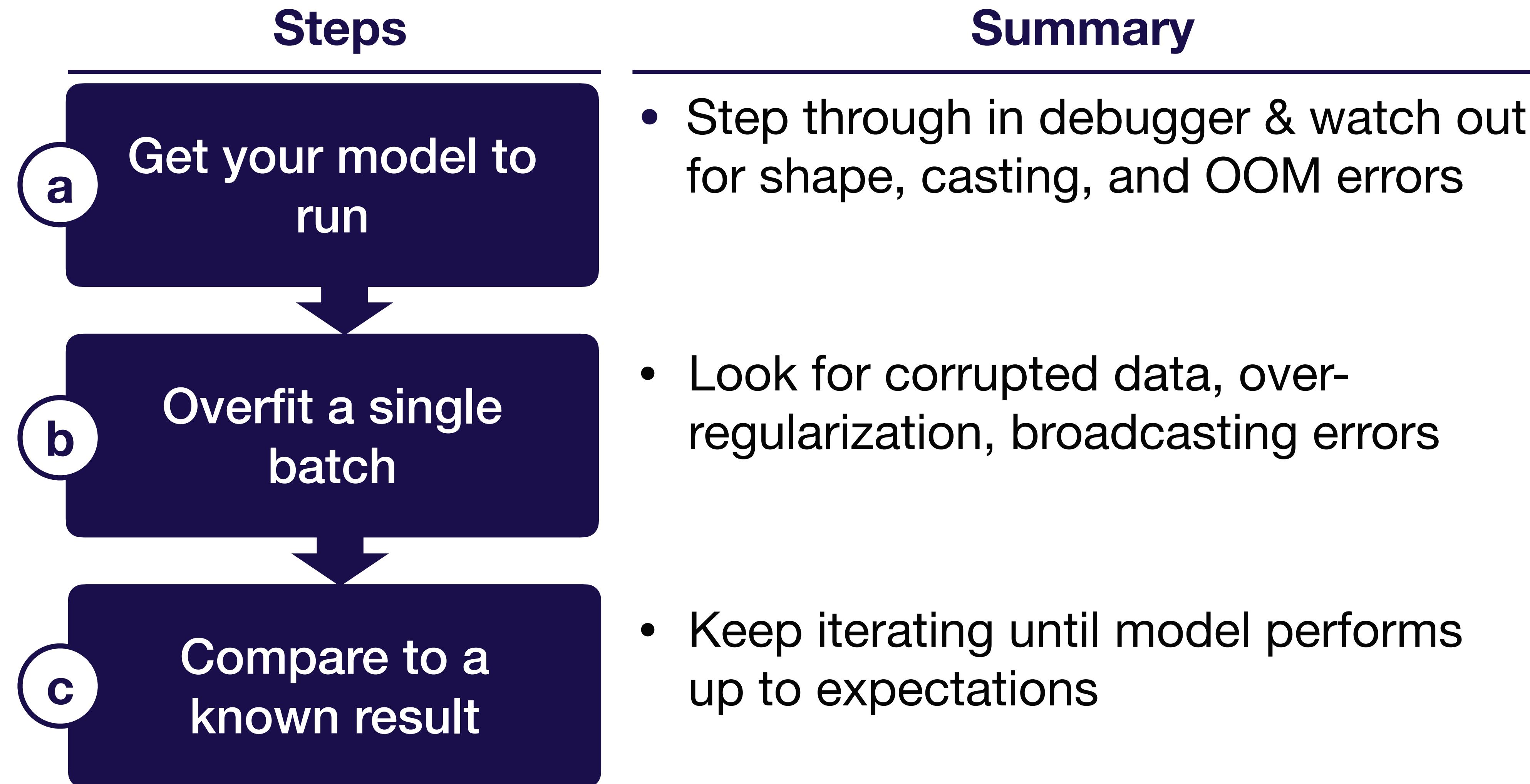
More useful

- Official model implementation evaluated on similar dataset to yours
- Official model implementation evaluated on benchmark (e.g., MNIST)
- Unofficial model implementation
- Results from the paper (with no code)
- Results from your model on a benchmark dataset (e.g., MNIST)
- Results from a similar model on a similar dataset
- Super simple baselines (e.g., average of outputs or linear regression)

Less useful



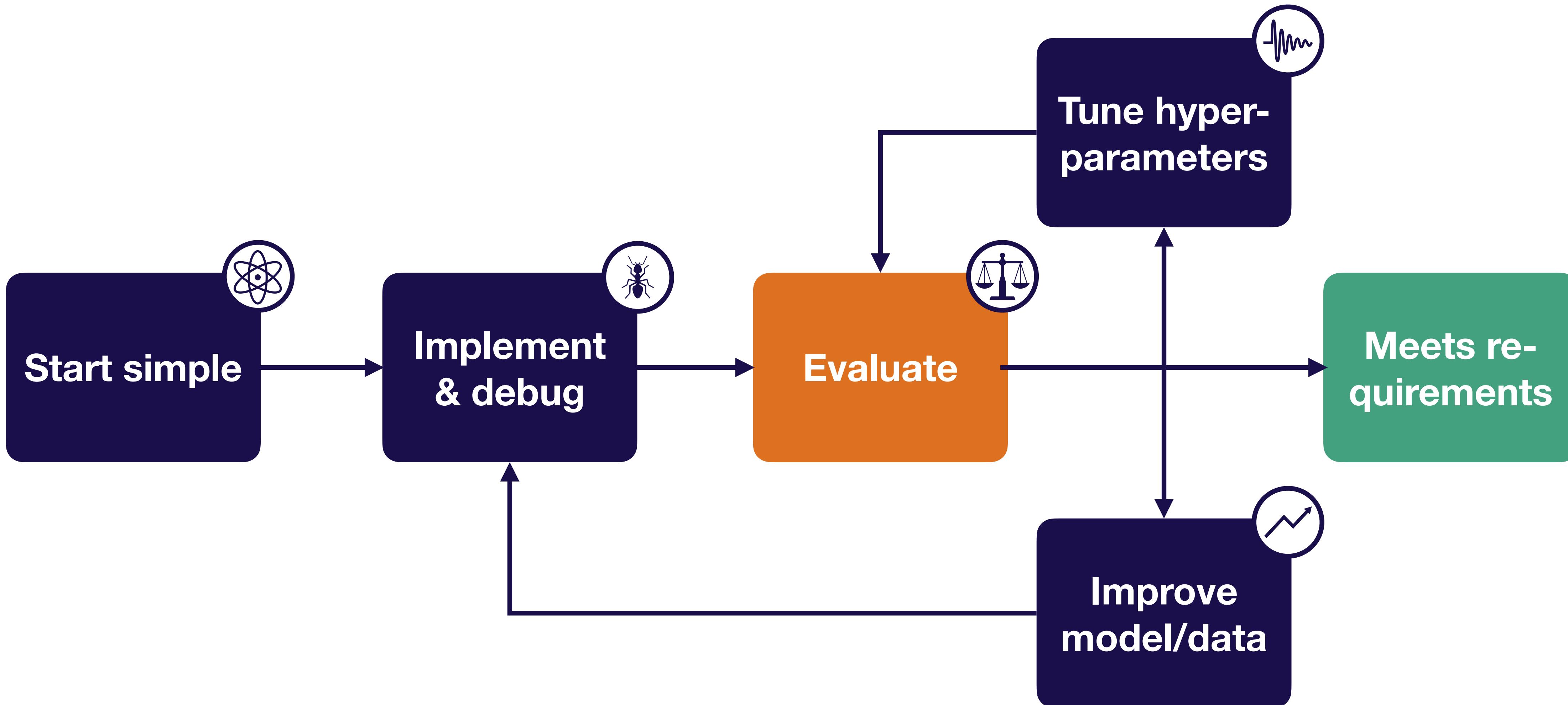
Summary: how to implement & debug



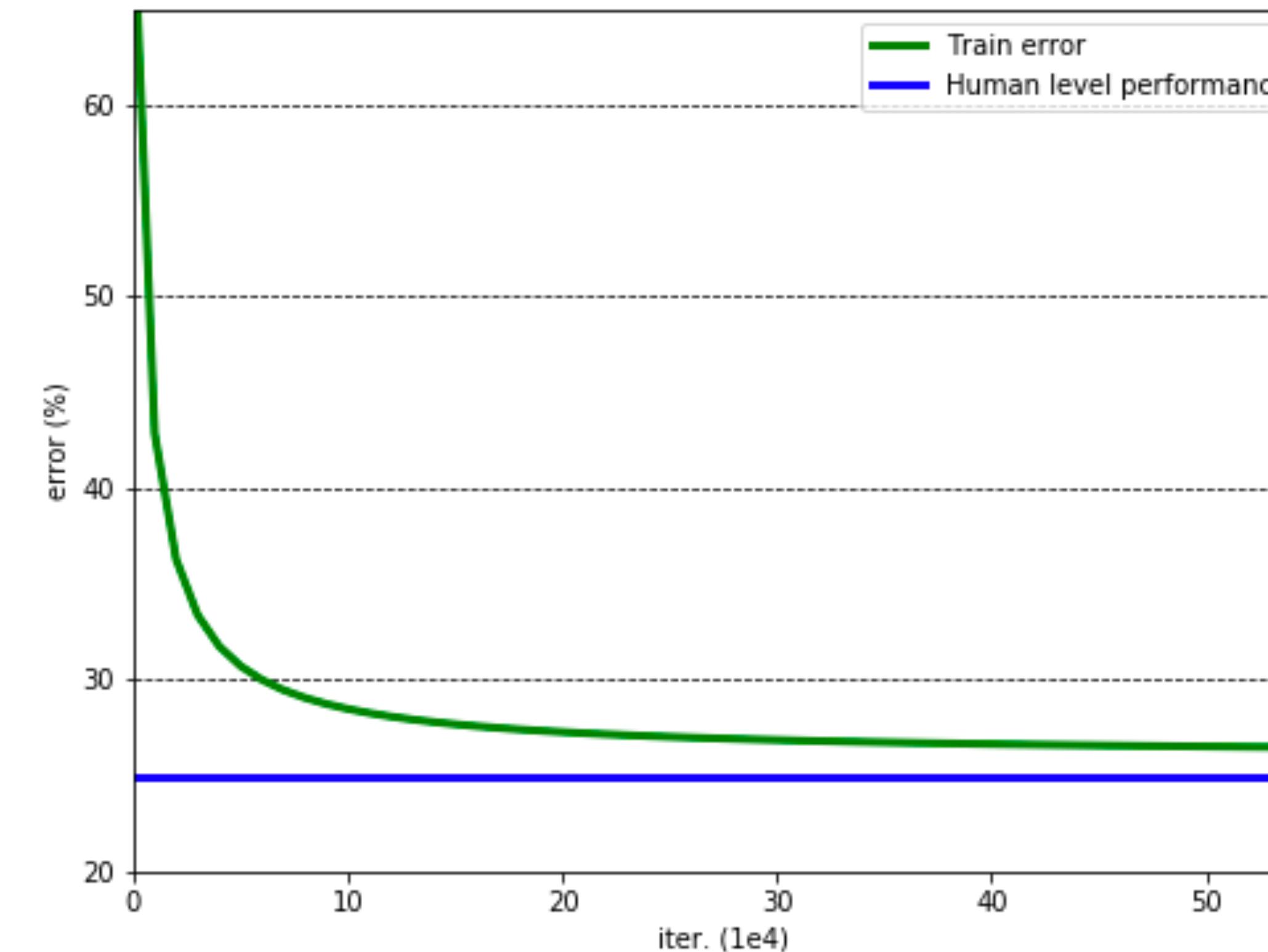
Questions?



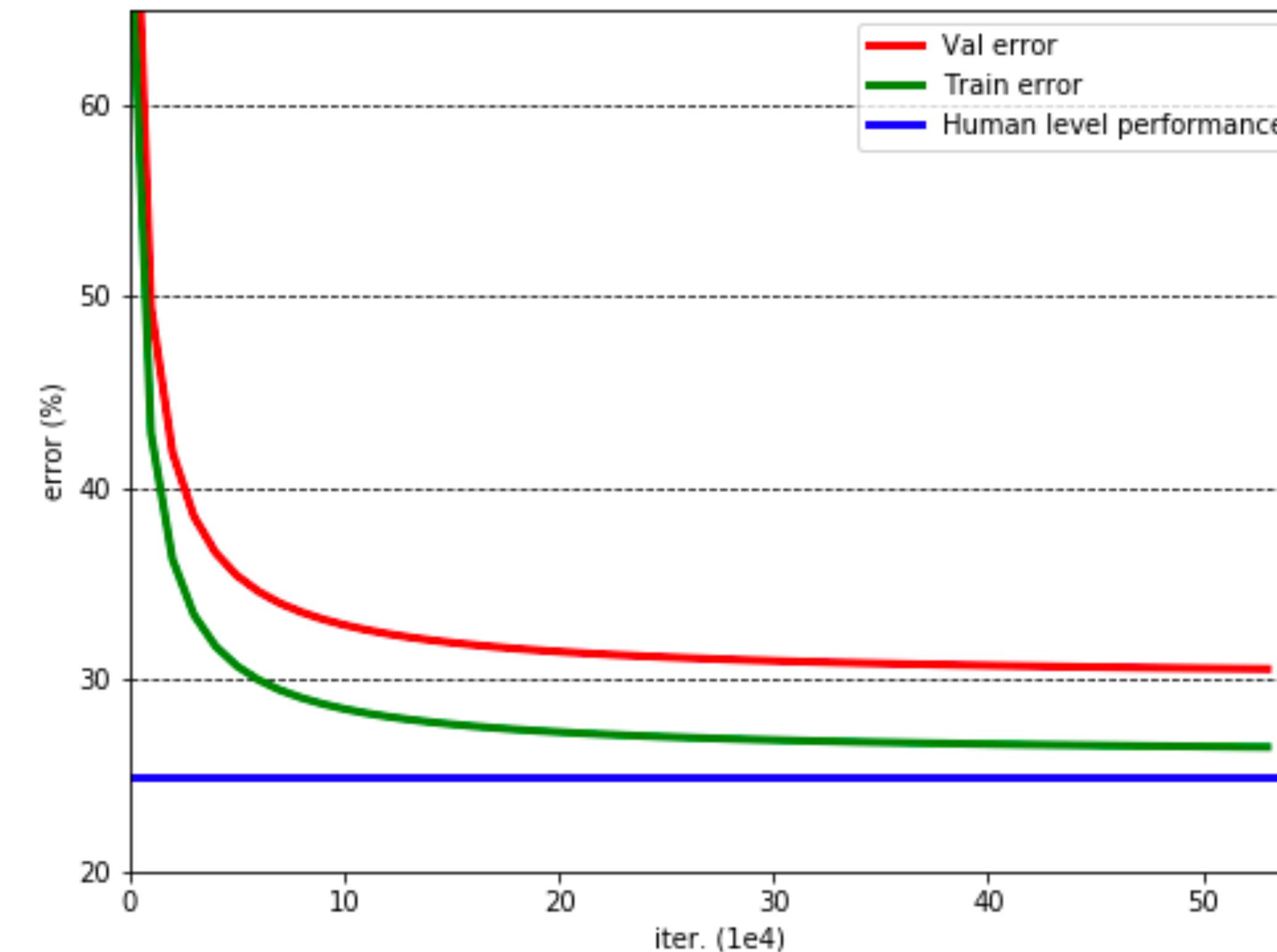
Strategy for DL troubleshooting



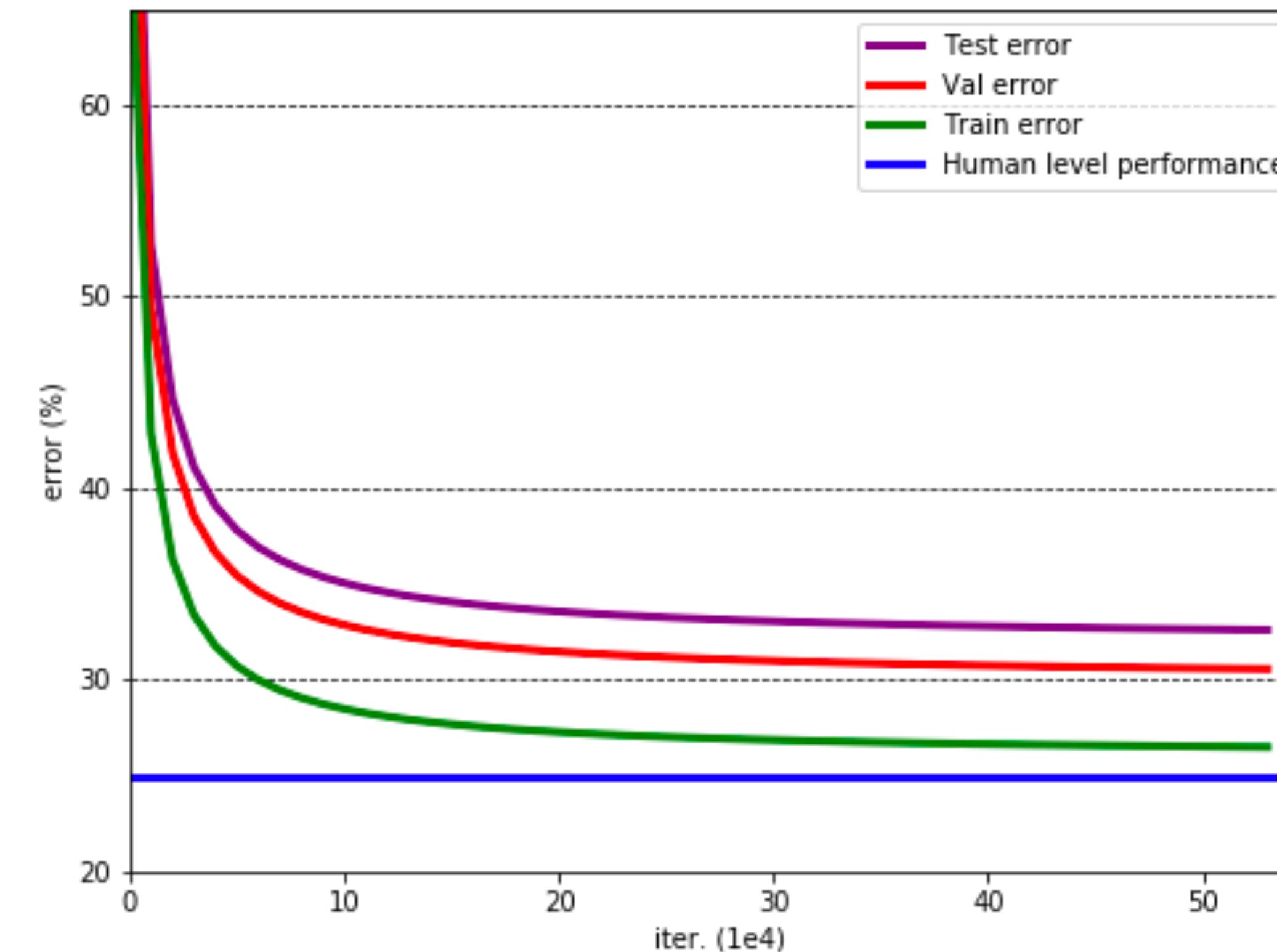
Bias-variance decomposition



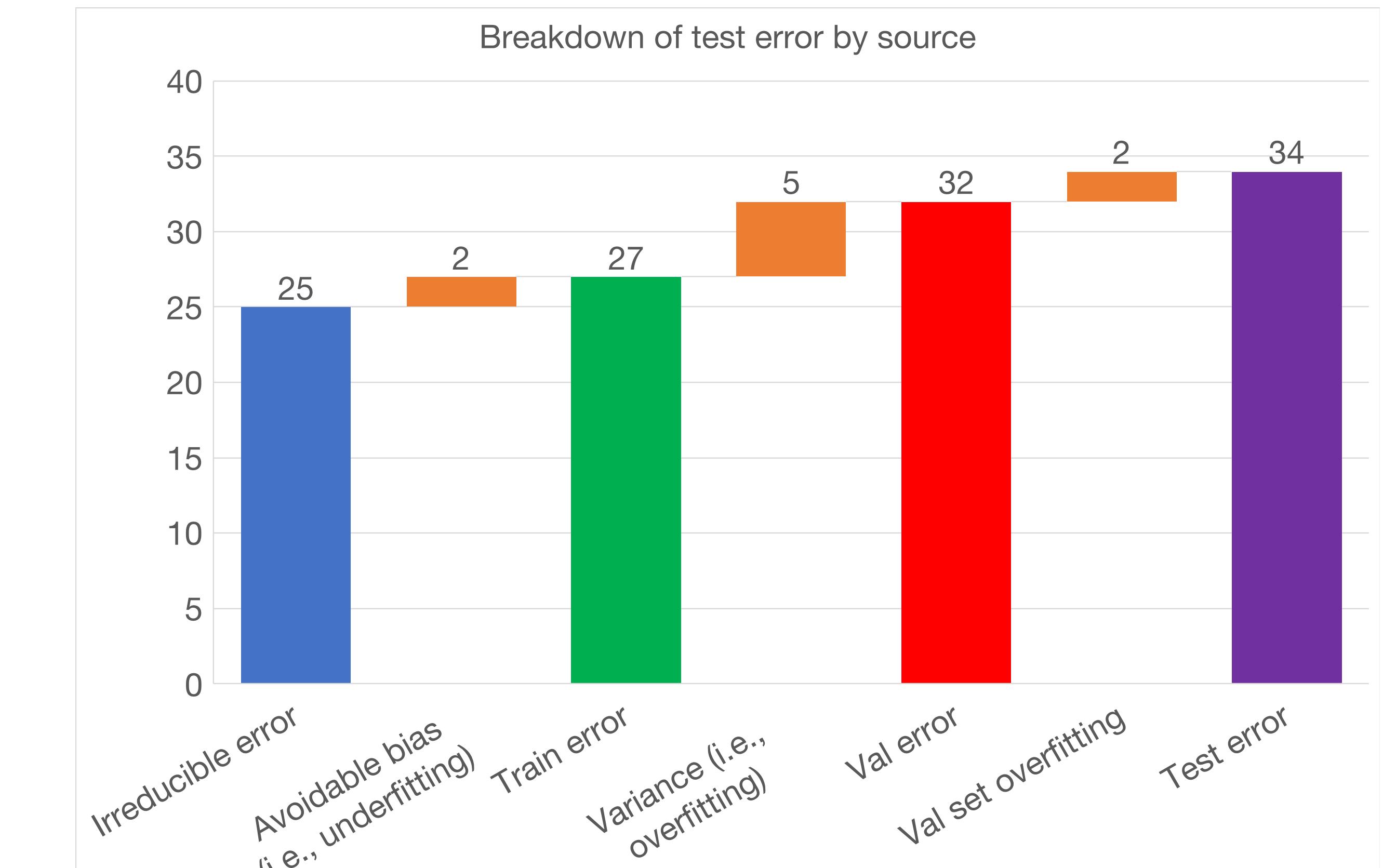
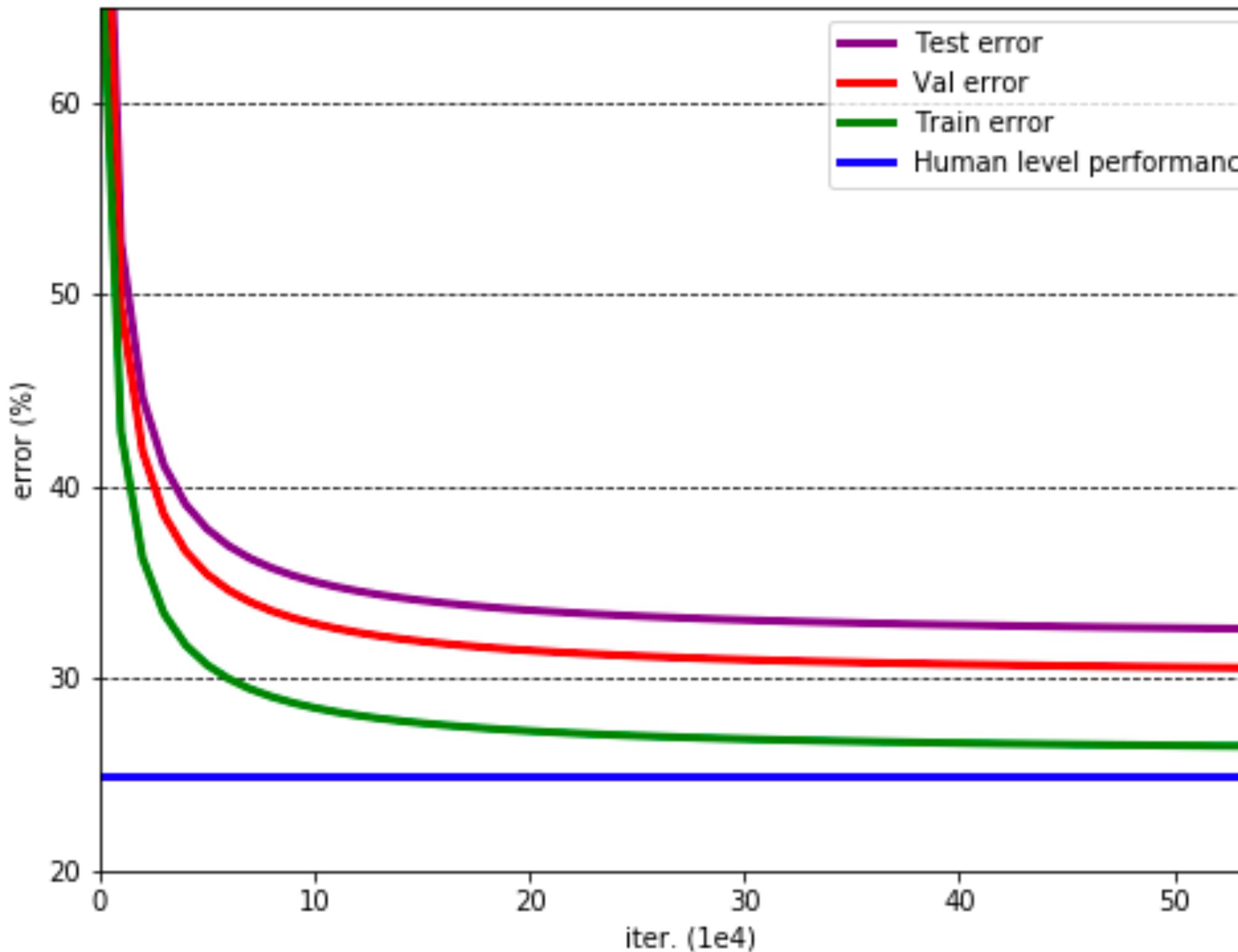
Bias-variance decomposition



Bias-variance decomposition



Bias-variance decomposition



Bias-variance decomposition

- **Test error = irreducible error + bias + variance + val overfitting**
- This assumes train, val, and test all come from the same distribution.
What if not?

Handling distribution shift

Train data

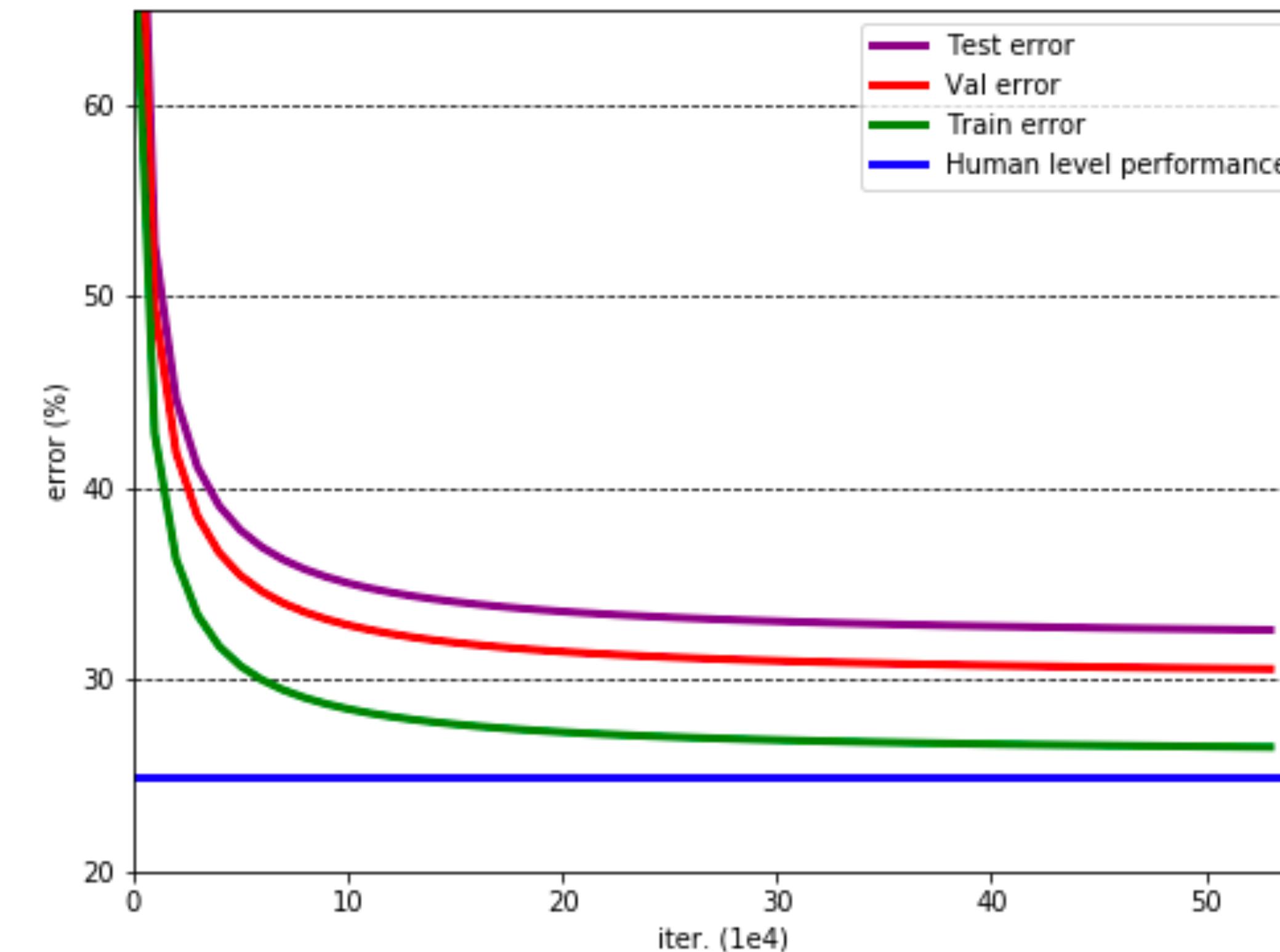


Test data

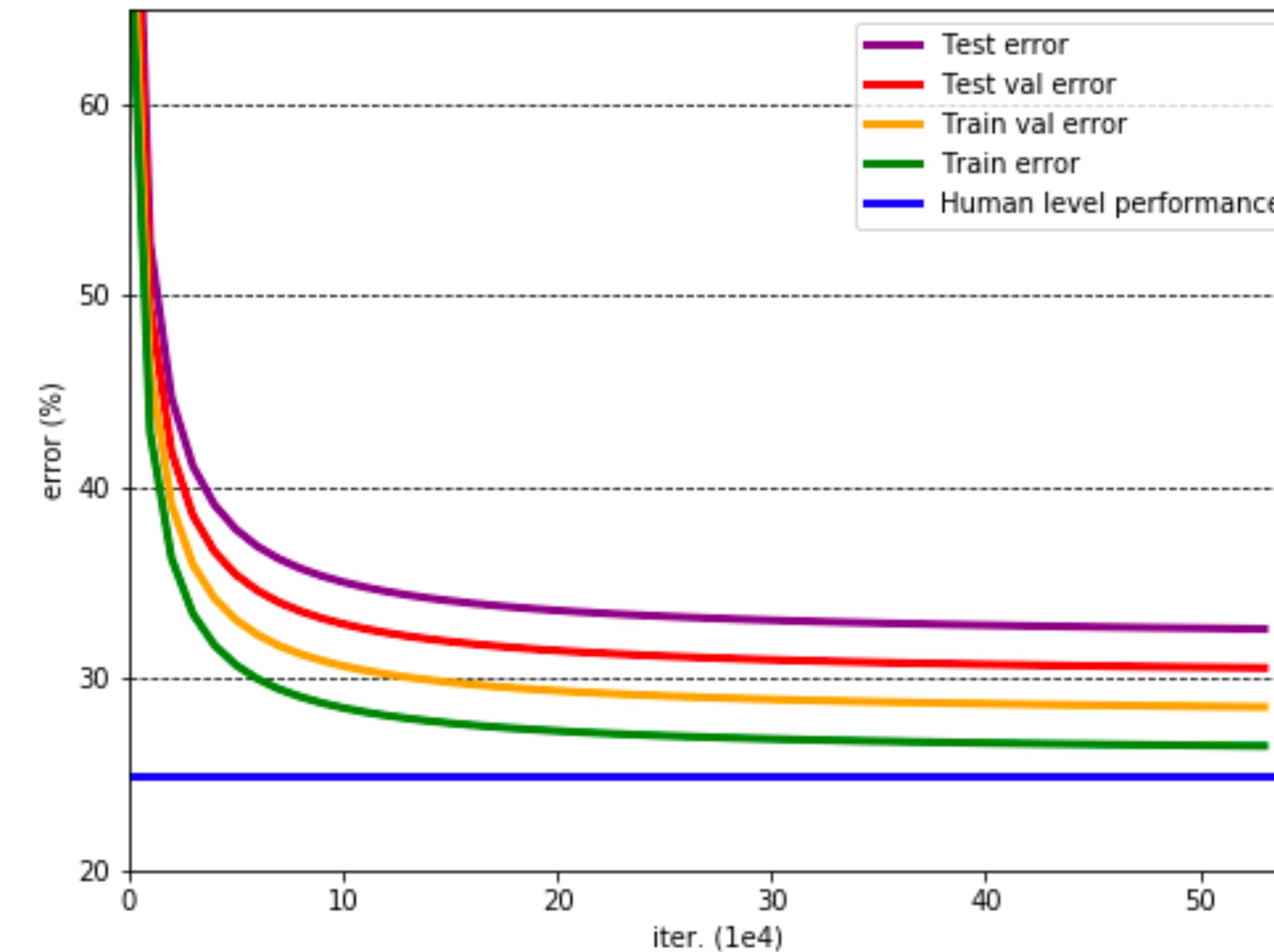


Use two val sets: one sampled from training distribution and one from test distribution

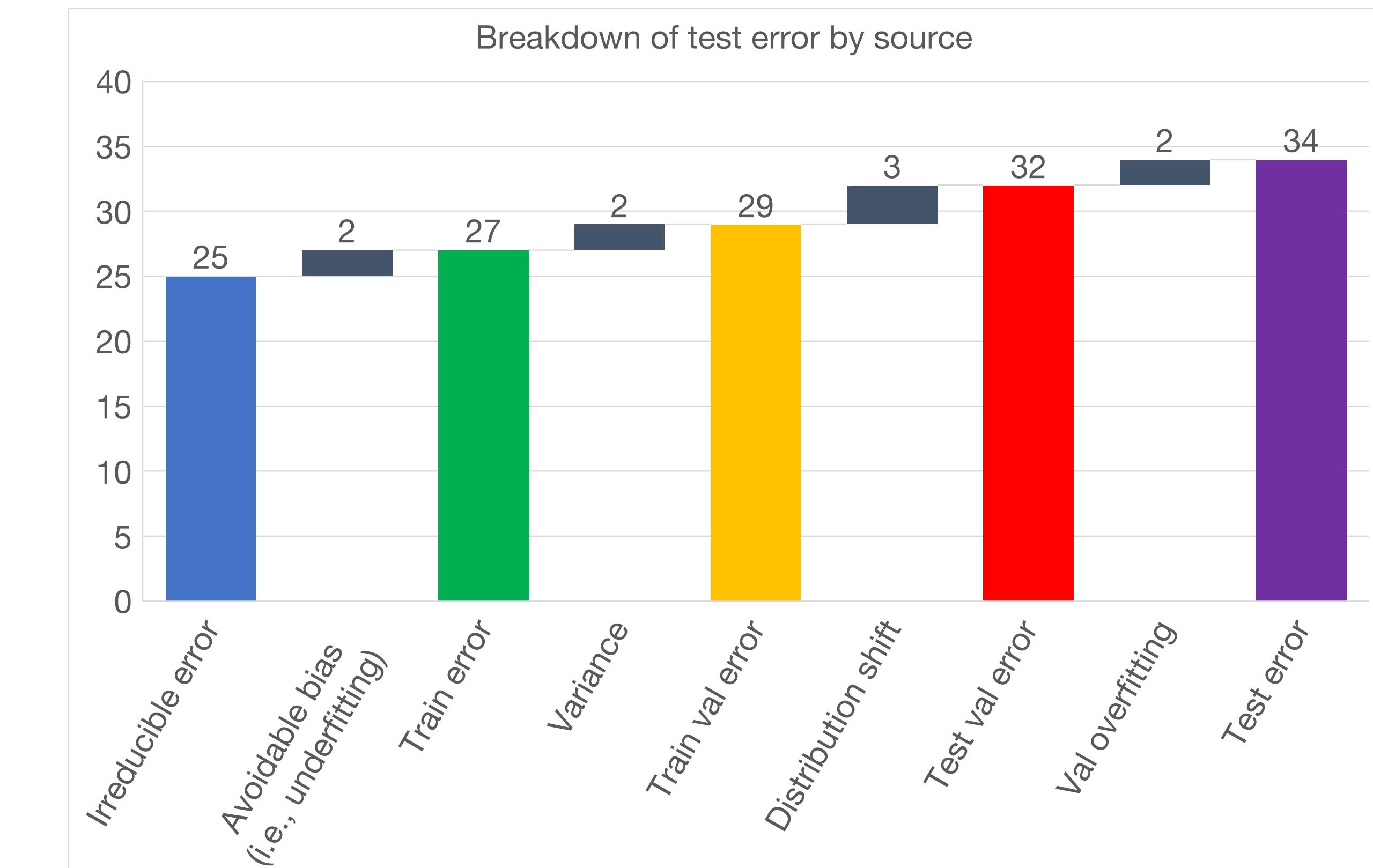
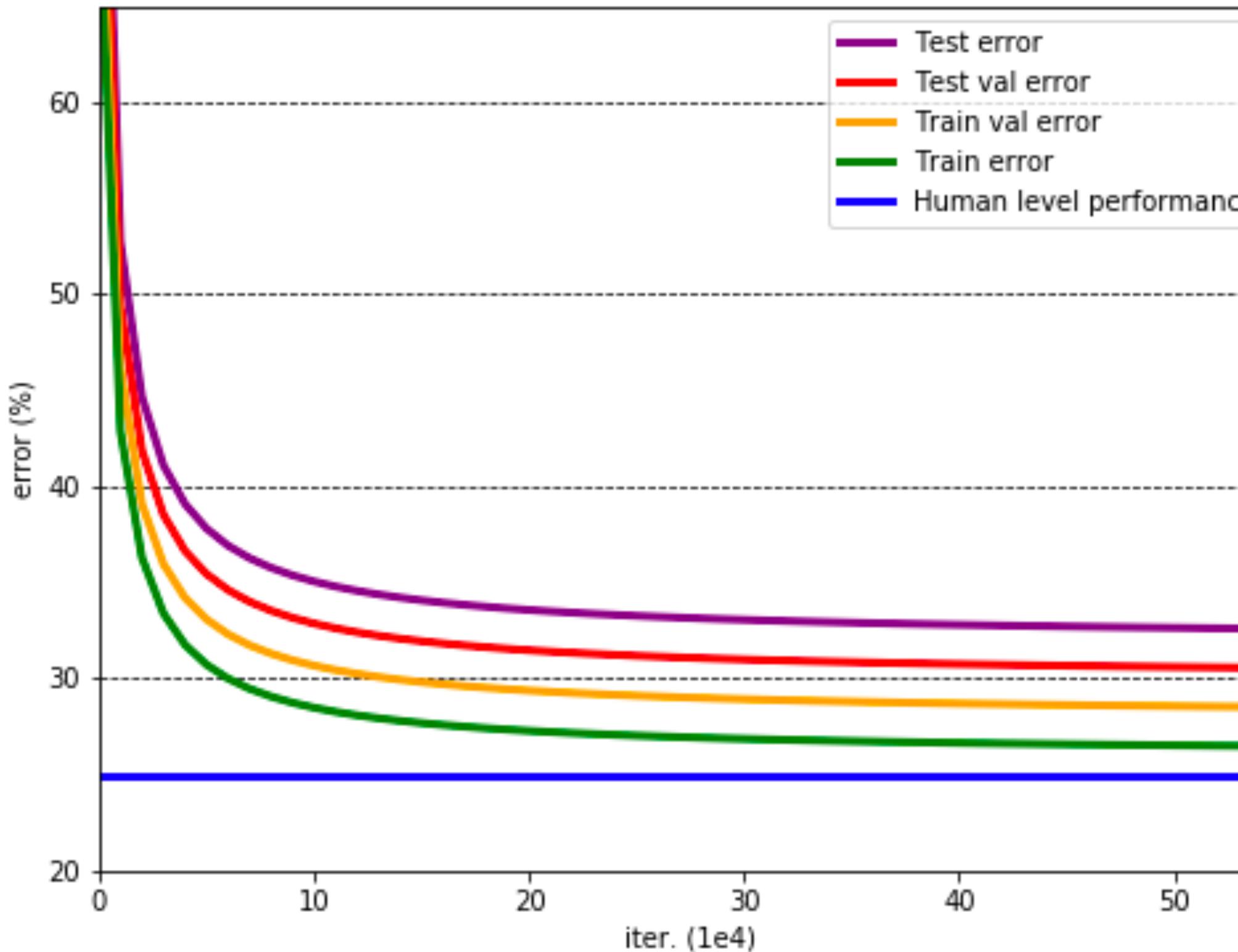
The bias-variance tradeoff



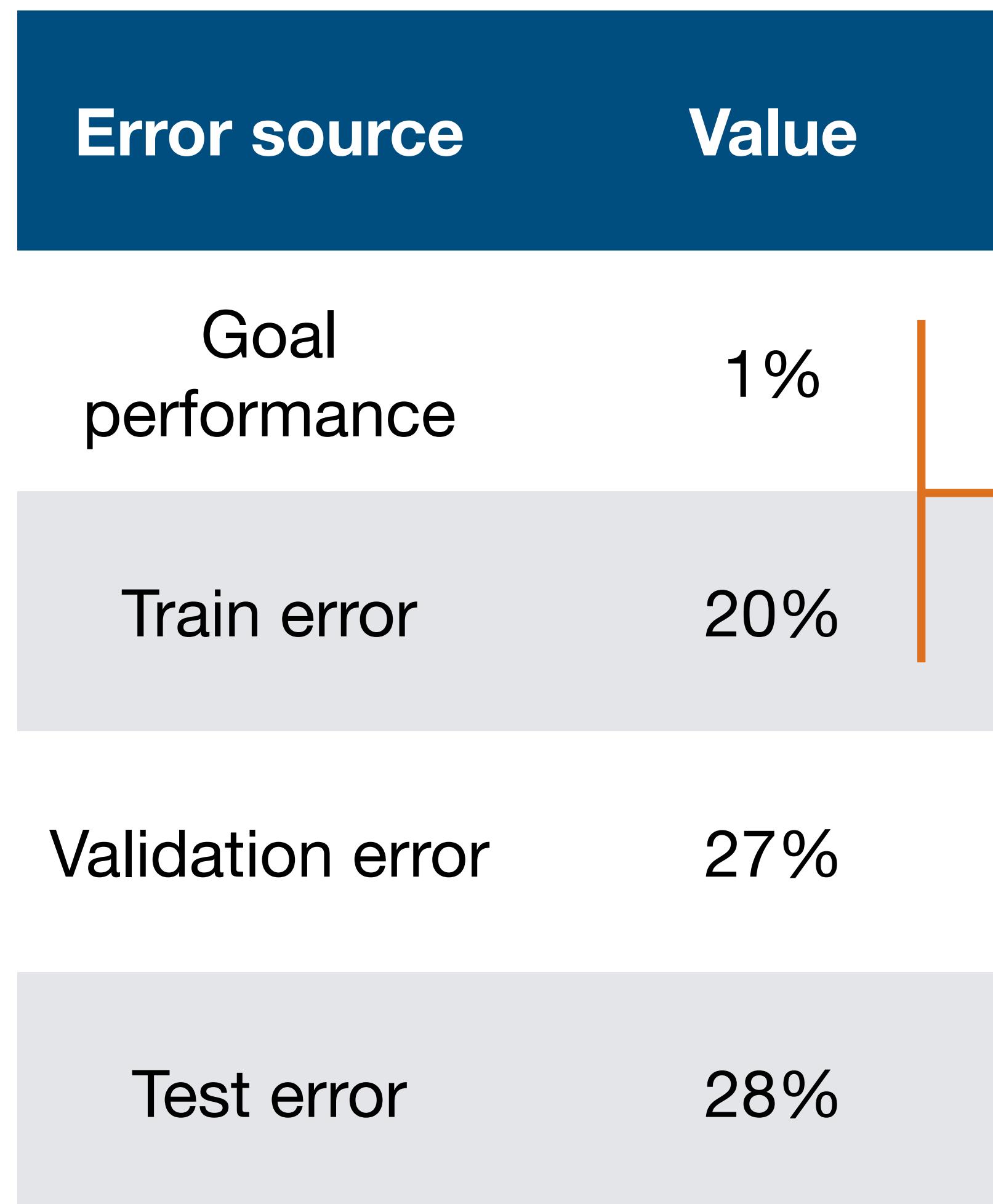
Bias-variance with distribution shift



Bias-variance with distribution shift



Train, val, and test error for pedestrian detection

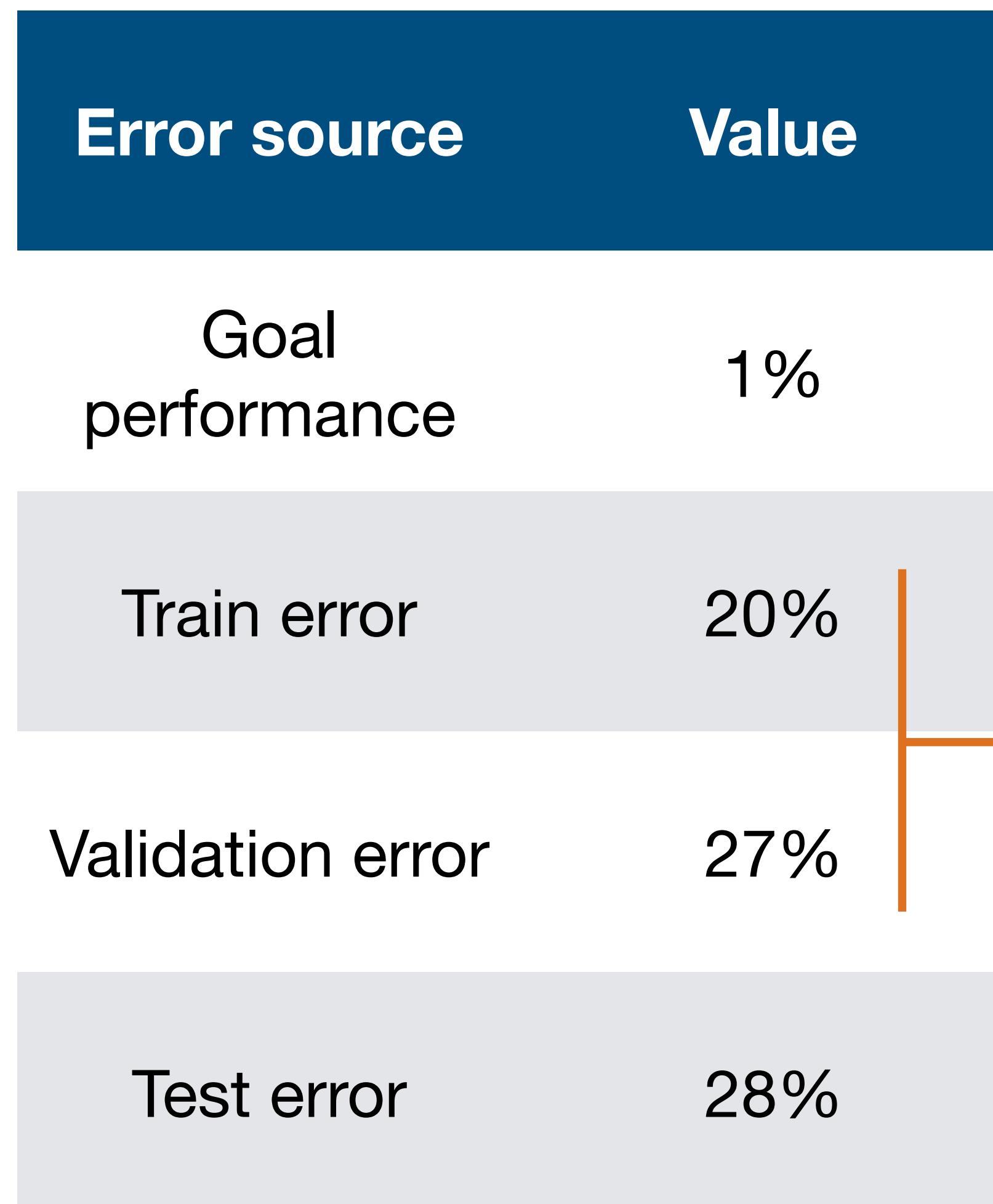


Running example



0 (no pedestrian) 1 (yes pedestrian)
Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

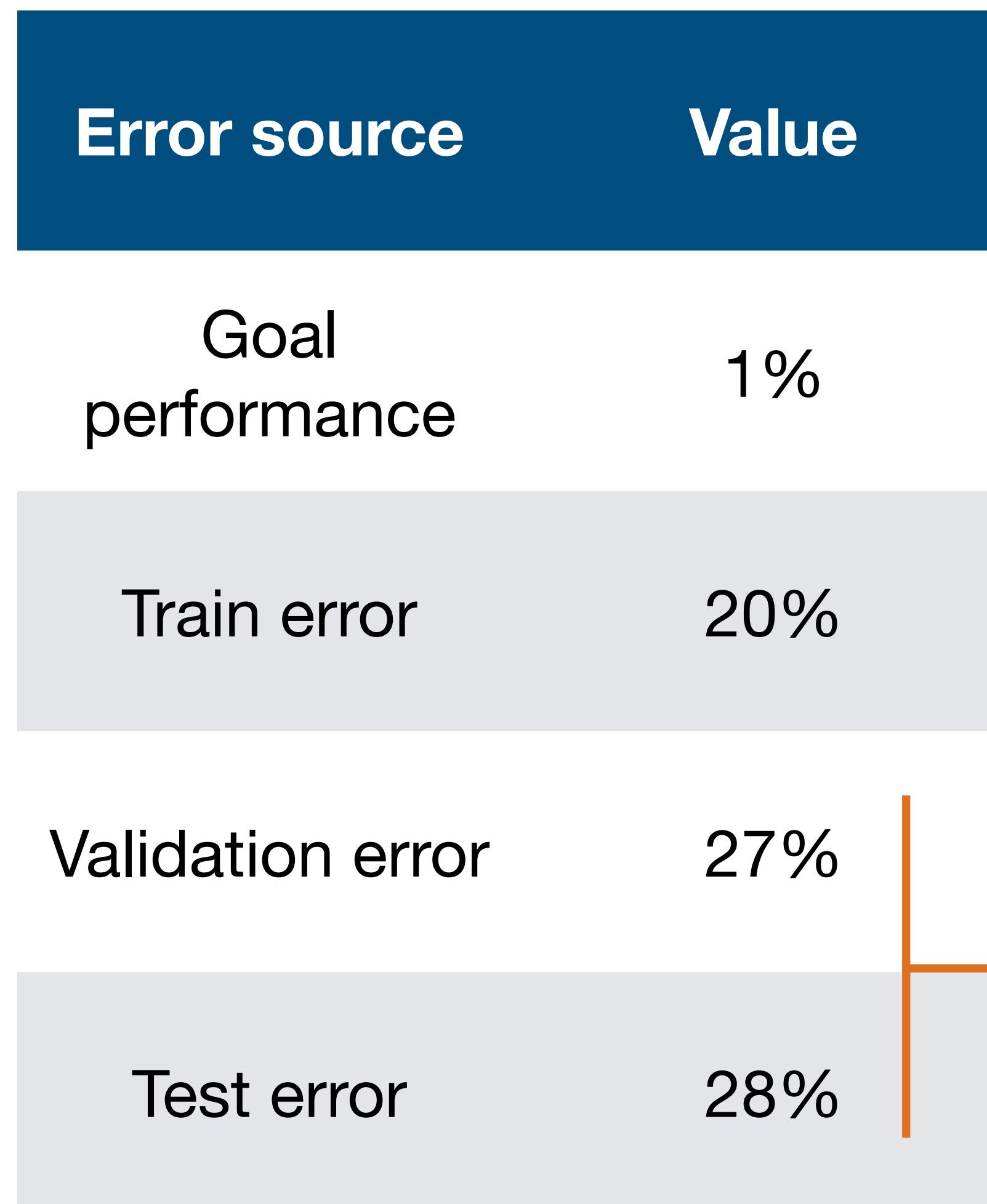


Running example



0 (no pedestrian) 1 (yes pedestrian)
Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection



Running example



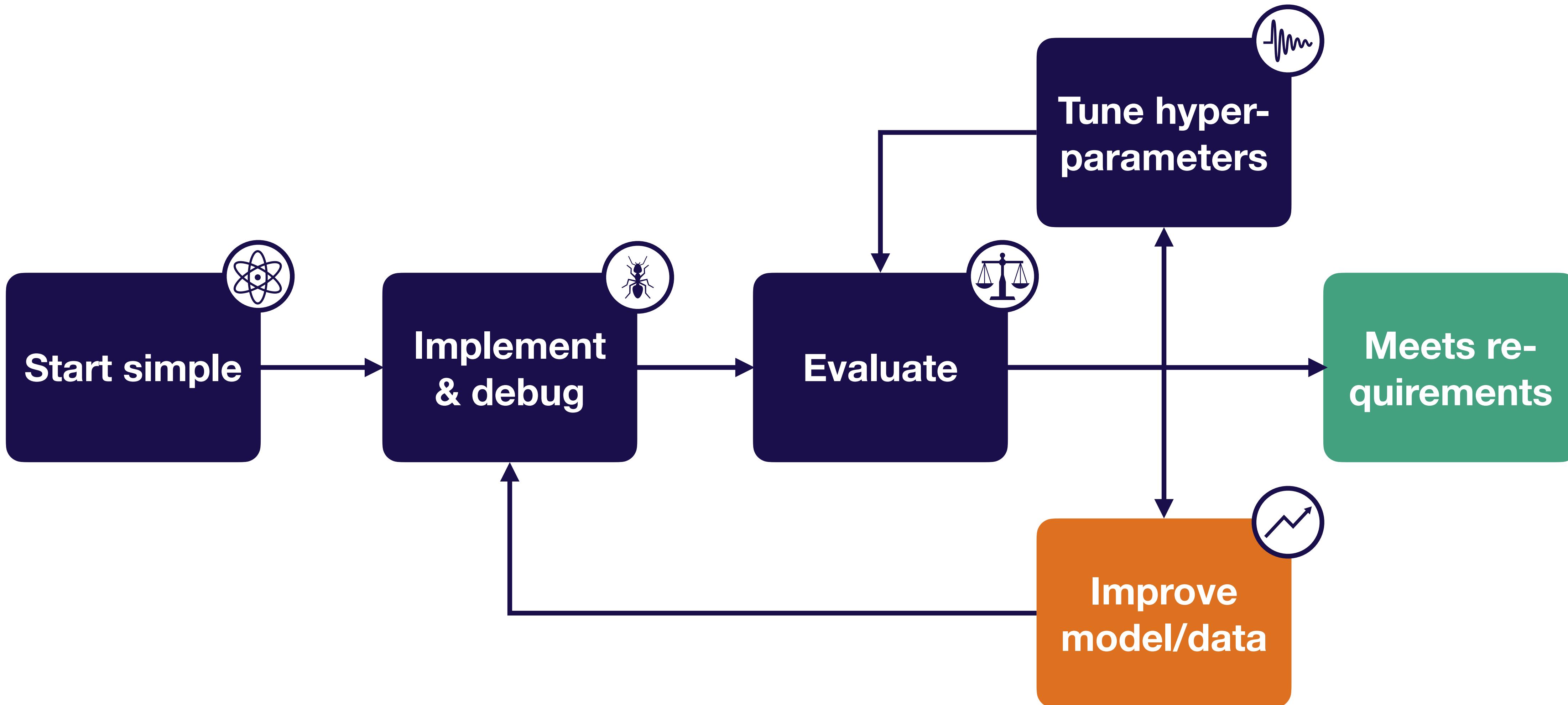
0 (no pedestrian) 1 (yes pedestrian)
Goal: 99% classification accuracy

Summary: evaluating model performance

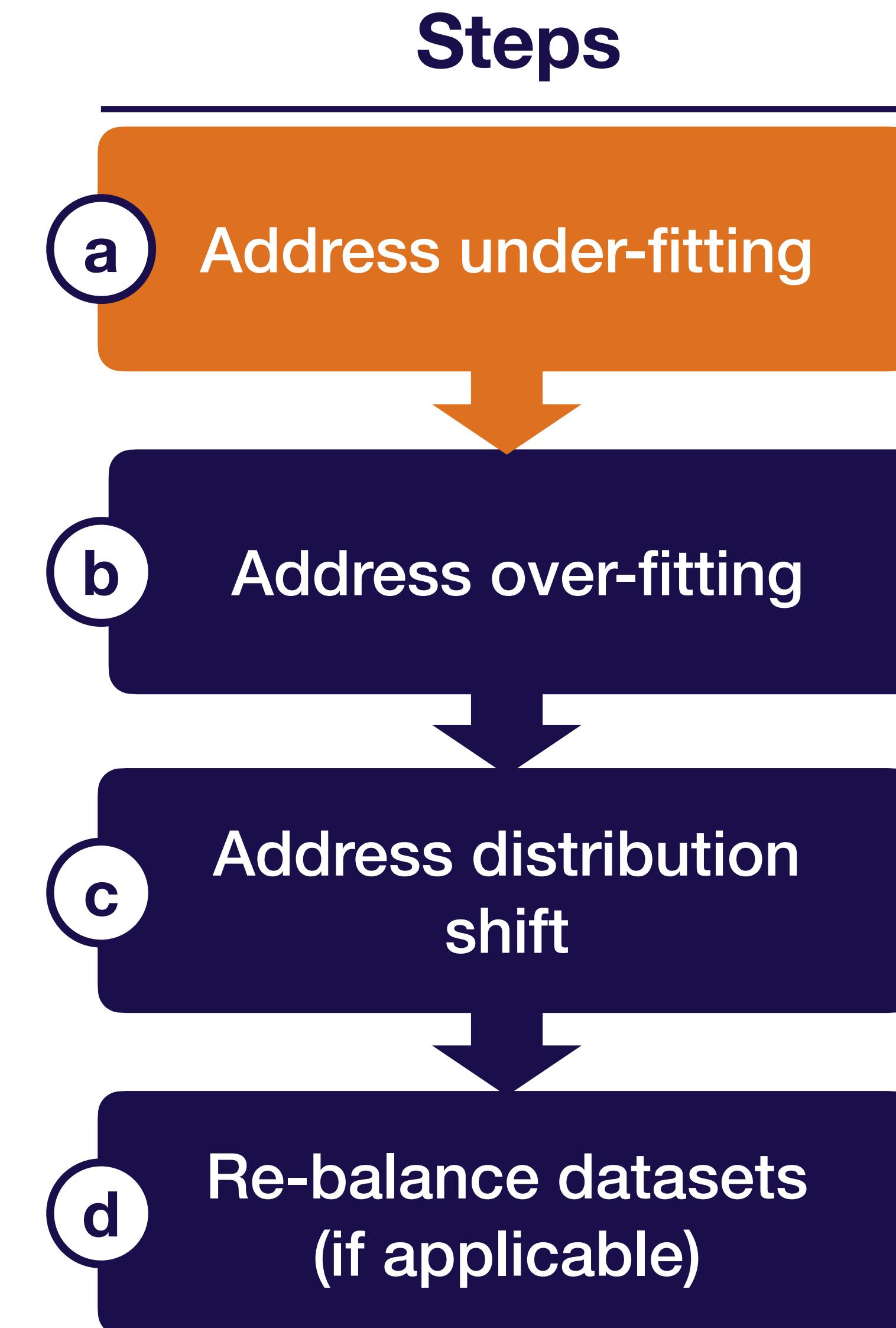
**Test error = irreducible error + bias + variance
+ distribution shift + val overfitting**

Questions?

Strategy for DL troubleshooting



Prioritizing improvements (i.e., applied b-v)



Addressing under-fitting (i.e., reducing bias)

Try first

- A. Make your model bigger (i.e., add layers or use more units per layer)
- B. Reduce regularization
- C. Error analysis
- D. Choose a different (closer to state-of-the art) model architecture (e.g., move from LeNet to ResNet)
- E. Tune hyper-parameters (e.g., learning rate)
- F. Add features

Try later

Train, val, and test error for pedestrian detection

Error source	Value	Value
Goal performance	1%	1%
Train error	20%	7%
Validation error	27%	19%
Test error	28%	20%

Add more layers
to the ConvNet



0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy
(i.e., 1% error)

Train, val, and test error for pedestrian detection

Error source	Value	Value	Value
Goal performance	1%	1%	1%
Train error	20%	7%	3%
Validation error	27%	19%	10%
Test error	28%	20%	10%

Switch to
ResNet-101



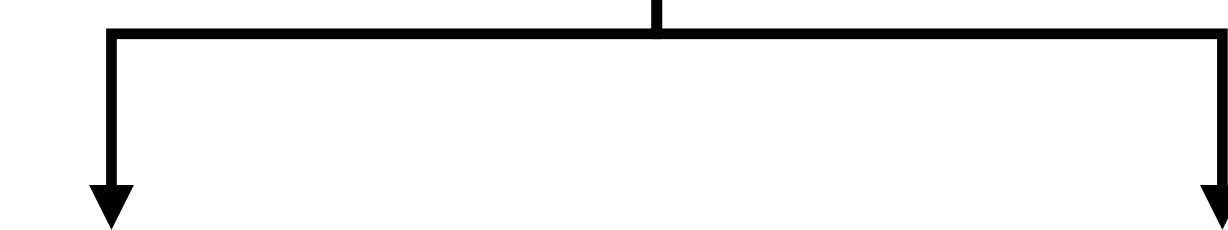
0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy
(i.e., 1% error)

Train, val, and test error for pedestrian detection

Error source	Value	Value	Value	Value
Goal performance	1%	1%	1%	1%
Train error	20%	7%	3%	0.8%
Validation error	27%	19%	10%	12%
Test error	28%	20%	10%	12%

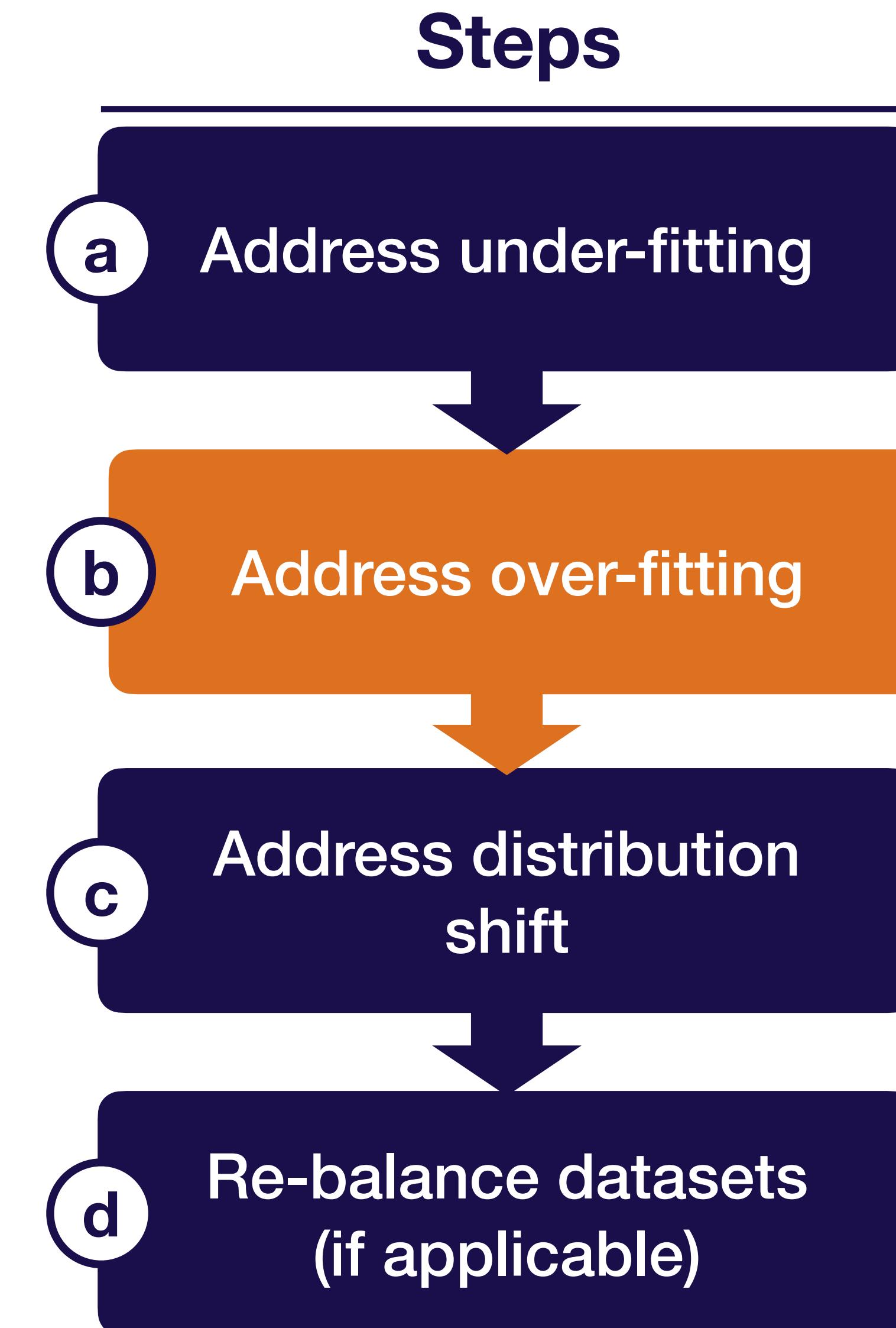
Tune learning
rate



0 (no pedestrian) 1 (yes pedestrian)

Goal: 99% classification accuracy
(i.e., 1% error)

Prioritizing improvements (i.e., applied b-v)



Addressing over-fitting (i.e., reducing variance)

Try first

- A. Add more training data (if possible!)
- B. Add normalization (e.g., batch norm, layer norm)
- C. Add data augmentation
- D. Increase regularization (e.g., dropout, L2, weight decay)
- E. Error analysis
- F. Choose a different (closer to state-of-the-art) model architecture
- G. Tune hyperparameters
- H. Early stopping
- I. Remove features
- J. Reduce model size

Try later

Addressing over-fitting (i.e., reducing variance)

Try first

- A. Add more training data (if possible!)
- B. Add normalization (e.g., batch norm, layer norm)
- C. Add data augmentation
- D. Increase regularization (e.g., dropout, L2, weight decay)
- E. Error analysis
- F. Choose a different (closer to state-of-the-art) model architecture
- G. Tune hyperparameters
- H. Early stopping
- I. Remove features
- J. Reduce model size

Not recommended!

Try later

Train, val, and test error for pedestrian detection

Error source	Value
Goal performance	1%
Train error	0.8%
Validation error	12%
Test error	12%

Running example



0 (no pedestrian) 1 (yes pedestrian)

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Increase dataset size to 250,000		
Error source	Value	Value
Goal performance	1%	1%
Train error	0.8%	1.5%
Validation error	12%	5%
Test error	12%	6%



Running example

0 (no pedestrian) 1 (yes pedestrian)
Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Error source	Value	Value	Value
Goal performance	1%	1%	1%
Train error	0.8%	1.5%	1.7%
Validation error	12%	5%	4%
Test error	12%	6%	4%

Add weight
decay



Running example



0 (no pedestrian)

1 (yes pedestrian)

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Error source	Value	Value	Value	Value
Goal performance	1%	1%	1%	1%
Train error	0.8%	1.5%	1.7%	2%
Validation error	12%	5%	4%	2.5%
Test error	12%	6%	4%	2.6%

Add data augmentation



Running example



0 (no pedestrian)

1 (yes pedestrian)

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Tune num layers, optimizer params, weight initialization, kernel size, weight decay

Error source	Value	Value	Value	Value	Value
Goal performance	1%	1%	1%	1%	1%
Train error	0.8%	1.5%	1.7%	2%	0.6%
Validation error	12%	5%	4%	2.5%	0.9%
Test error	12%	6%	4%	2.6%	1.0%

Running example

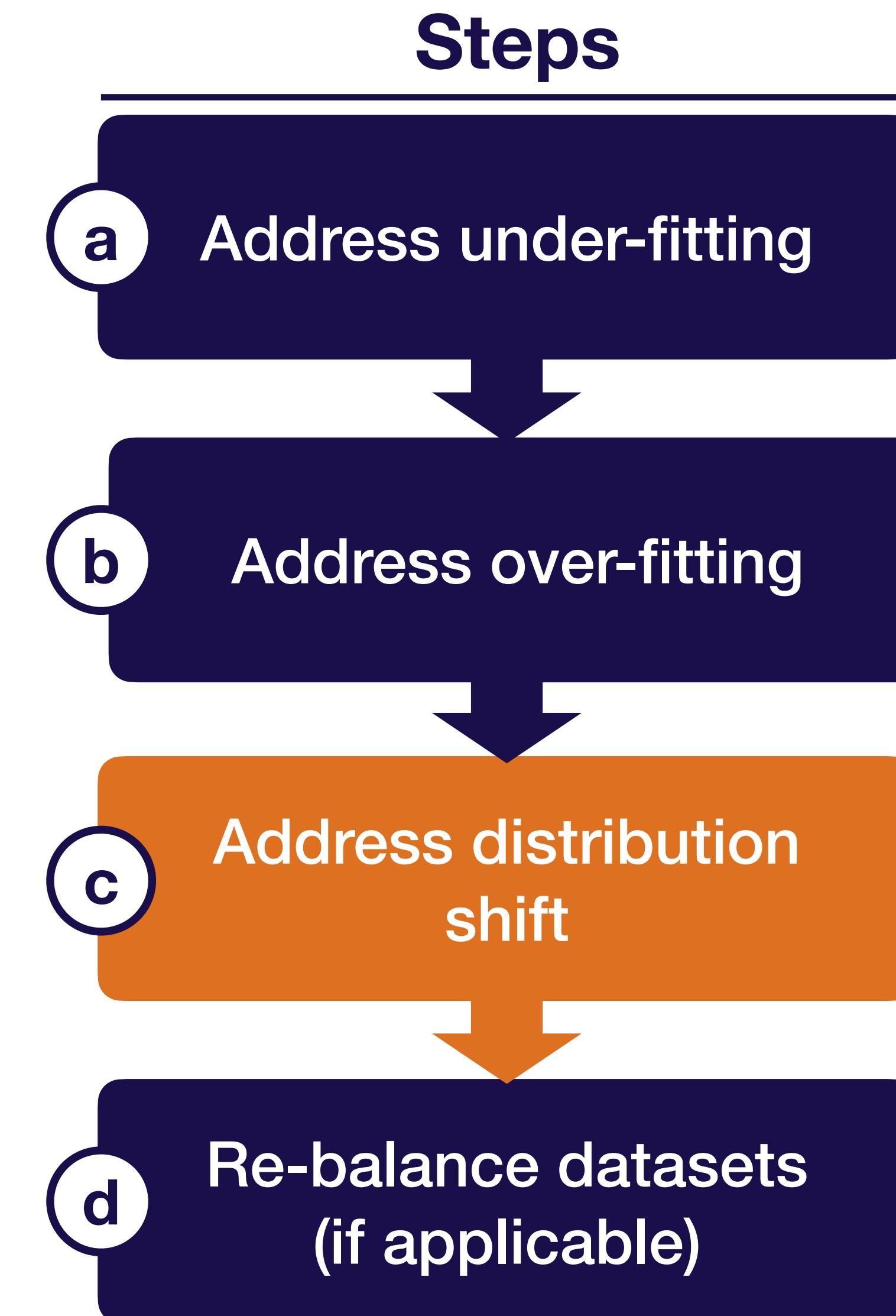


0 (no pedestrian)

1 (yes pedestrian)

Goal: 99% classification accuracy

Prioritizing improvements (i.e., applied b-v)



Addressing distribution shift

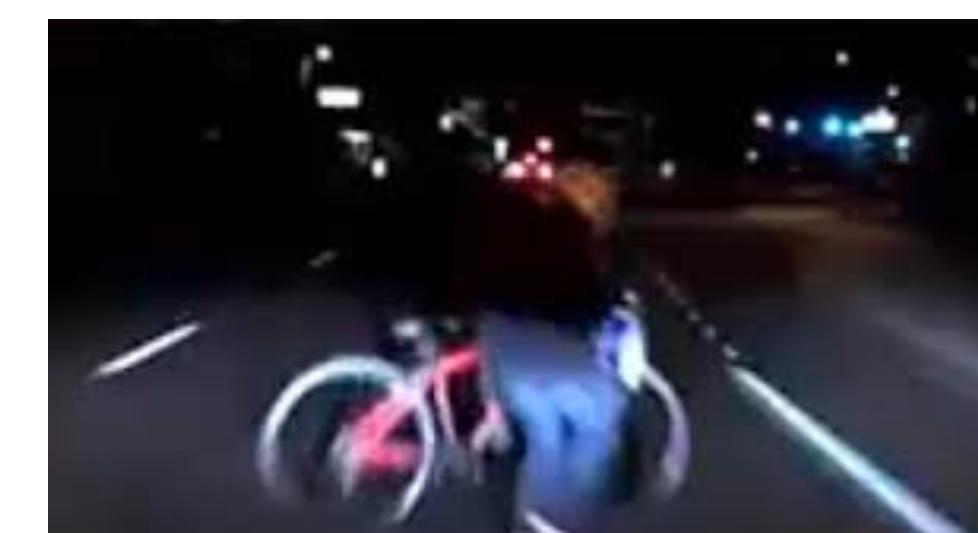
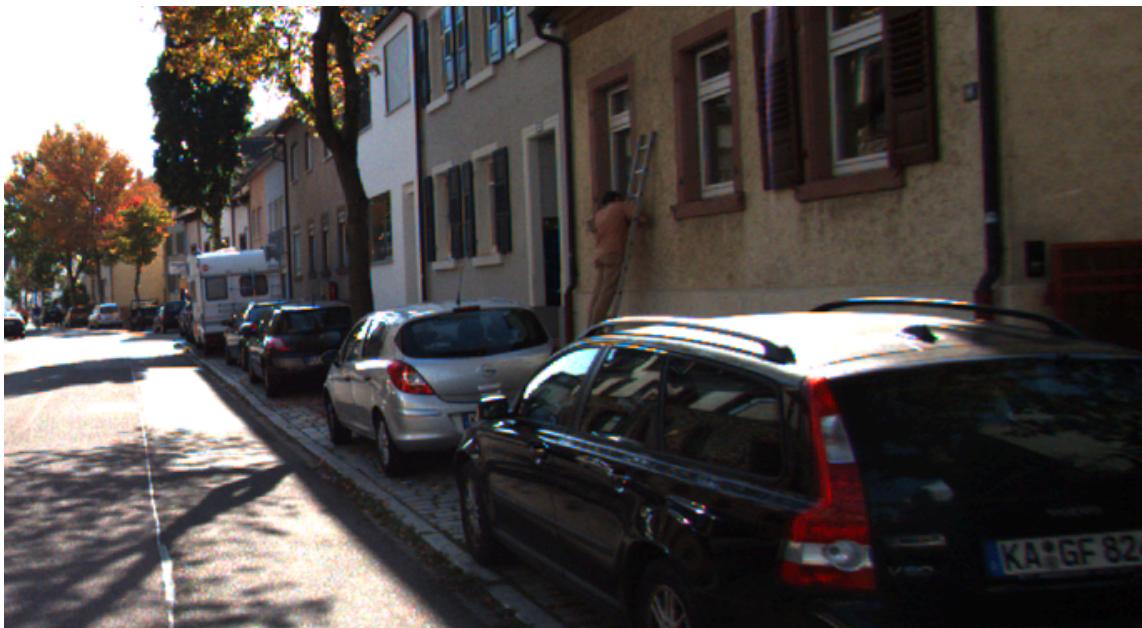
Try first

- A. Analyze test-val set errors & collect more training data to compensate
- B. Analyze test-val set errors & synthesize more training data to compensate
- C. Apply domain adaptation techniques to training & test distributions

Try later

Error analysis

Test-val set errors (no pedestrian detected)

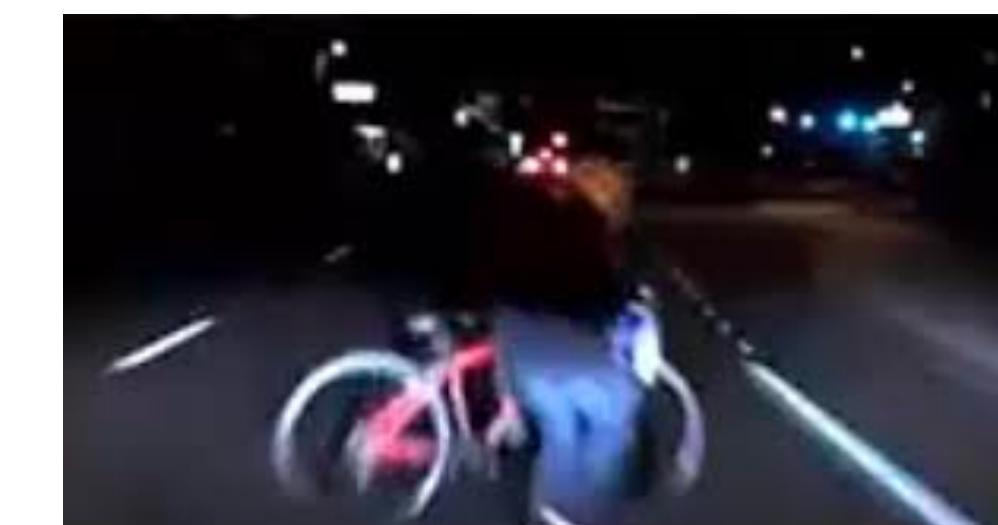


Train-val set errors (no pedestrian detected)



Error analysis

Test-val set errors (no pedestrian detected)



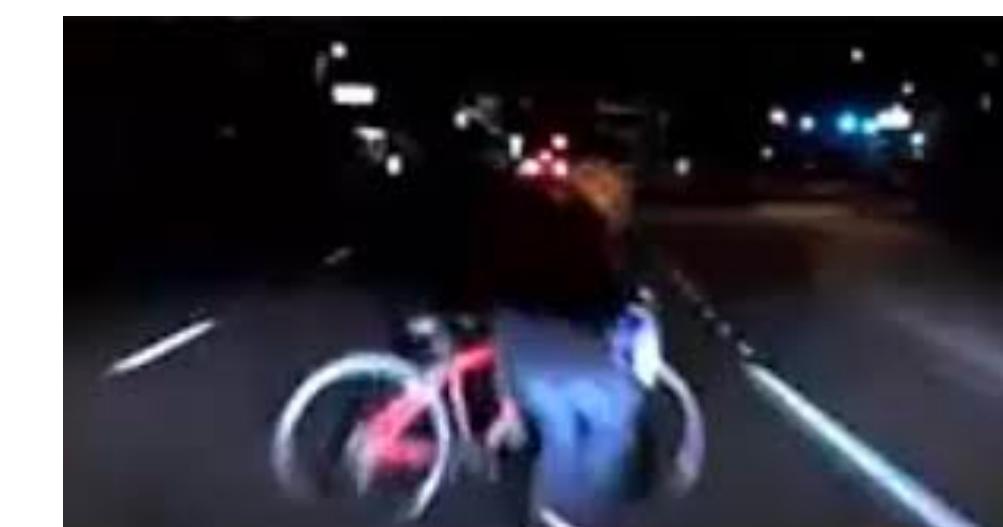
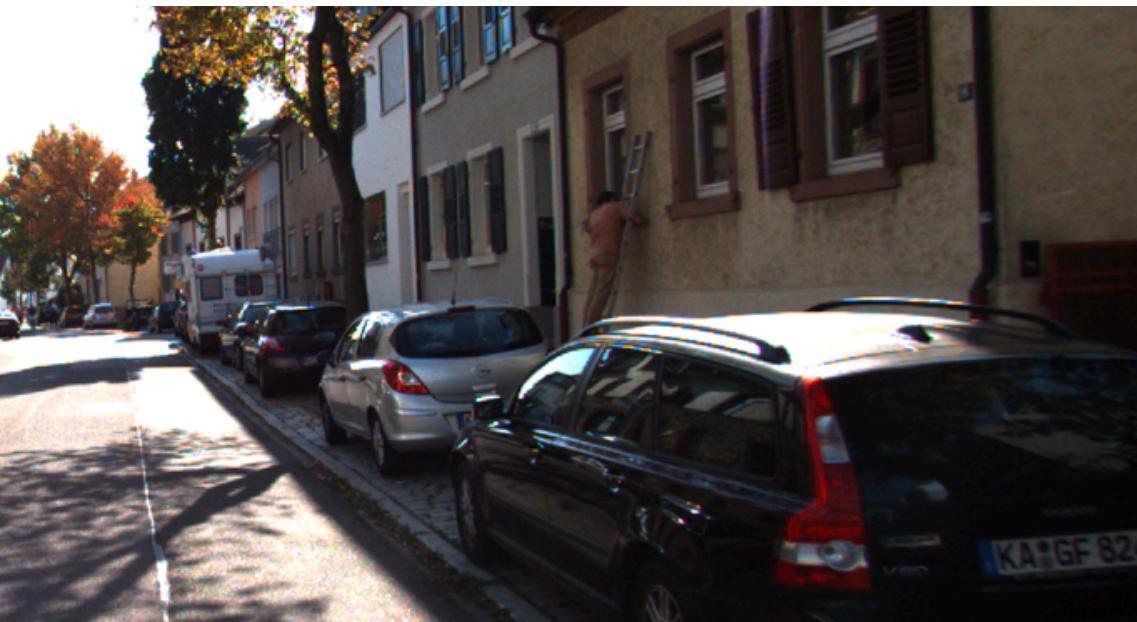
**Error type 1: hard-to-see
pedestrians**

Train-val set errors (no pedestrian detected)



Error analysis

Test-val set errors (no pedestrian detected)



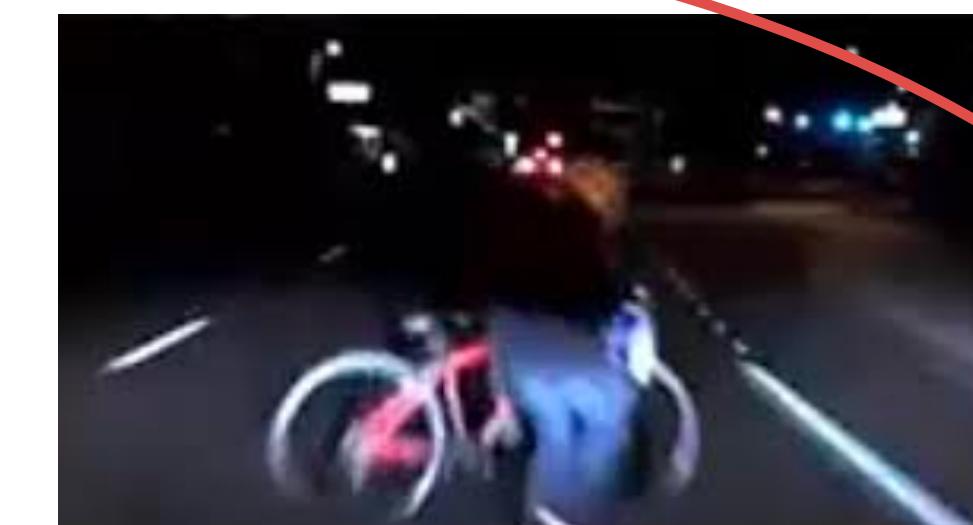
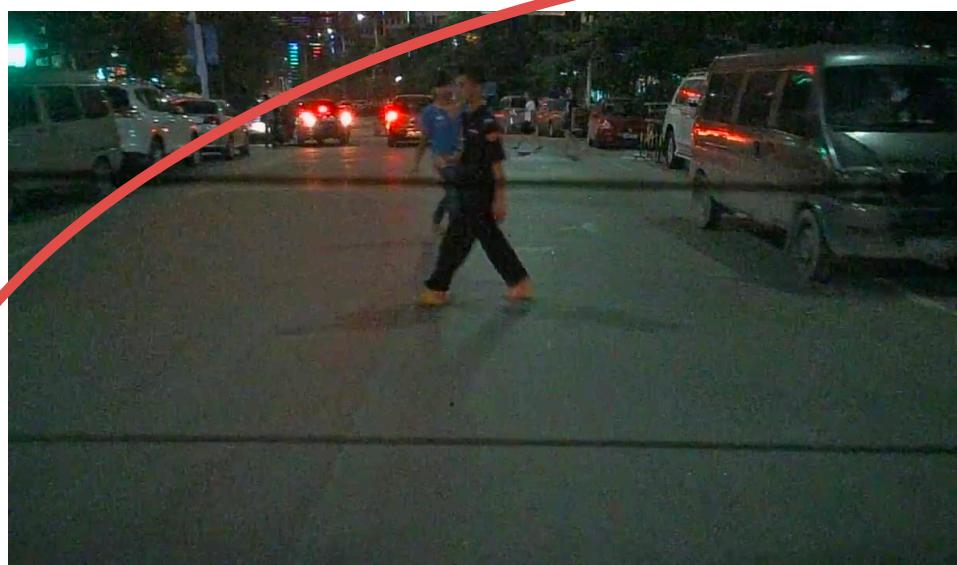
Train-val set errors (no pedestrian detected)



Error type 2: reflections

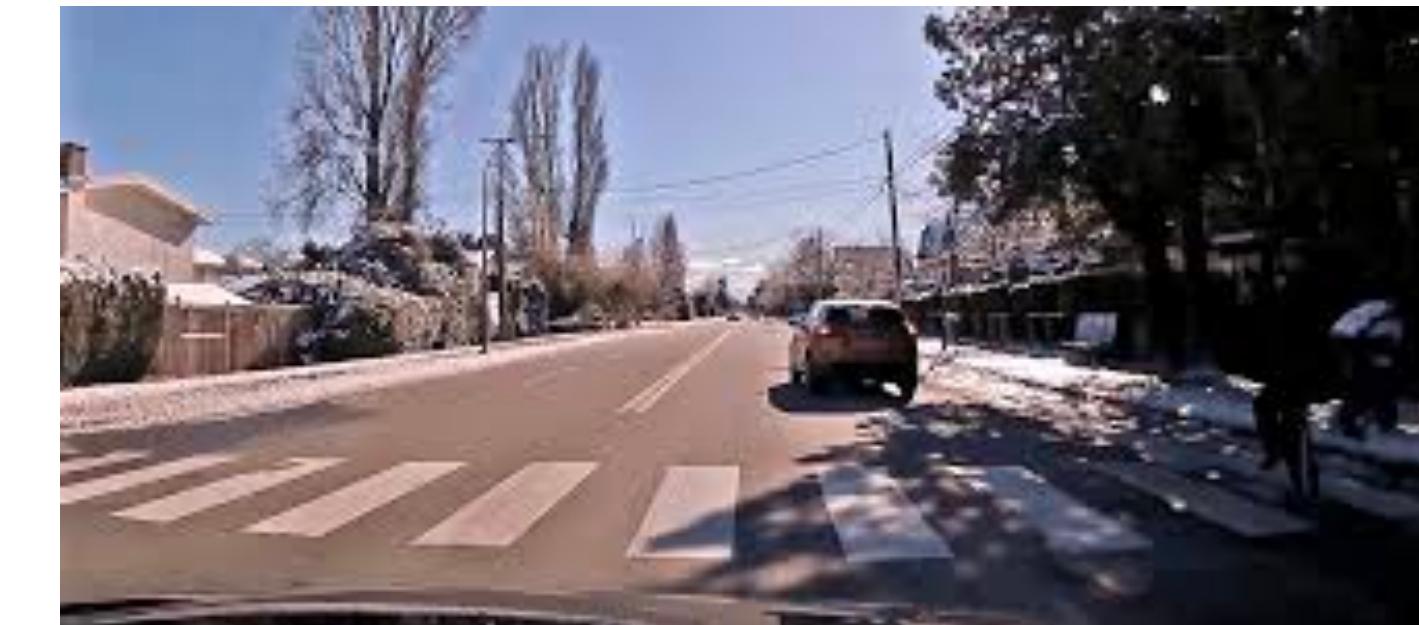
Error analysis

Test-val set errors (no pedestrian detected)



**Error type 3 (test-val only):
night scenes**

Train-val set errors (no pedestrian detected)



Error analysis

Error type	Error % (train-val)	Error % (test-val)	Potential solutions	Priority
1. Hard-to-see pedestrians	0.1%	0.1%	<ul style="list-style-type: none">• Better sensors	Low
2. Reflections	0.3%	0.3%	<ul style="list-style-type: none">• Collect more data with reflections• Add synthetic reflections to train set• Try to remove with pre-processing• Better sensors	Medium
3. Nighttime scenes	0.1%	1%	<ul style="list-style-type: none">• Collect more data at night• Synthetically darken training images• Simulate night-time data• Use domain adaptation	High

Domain adaptation

What is it?

Techniques to train on “source” distribution and generalize to another “target” using only unlabeled data or limited labeled data

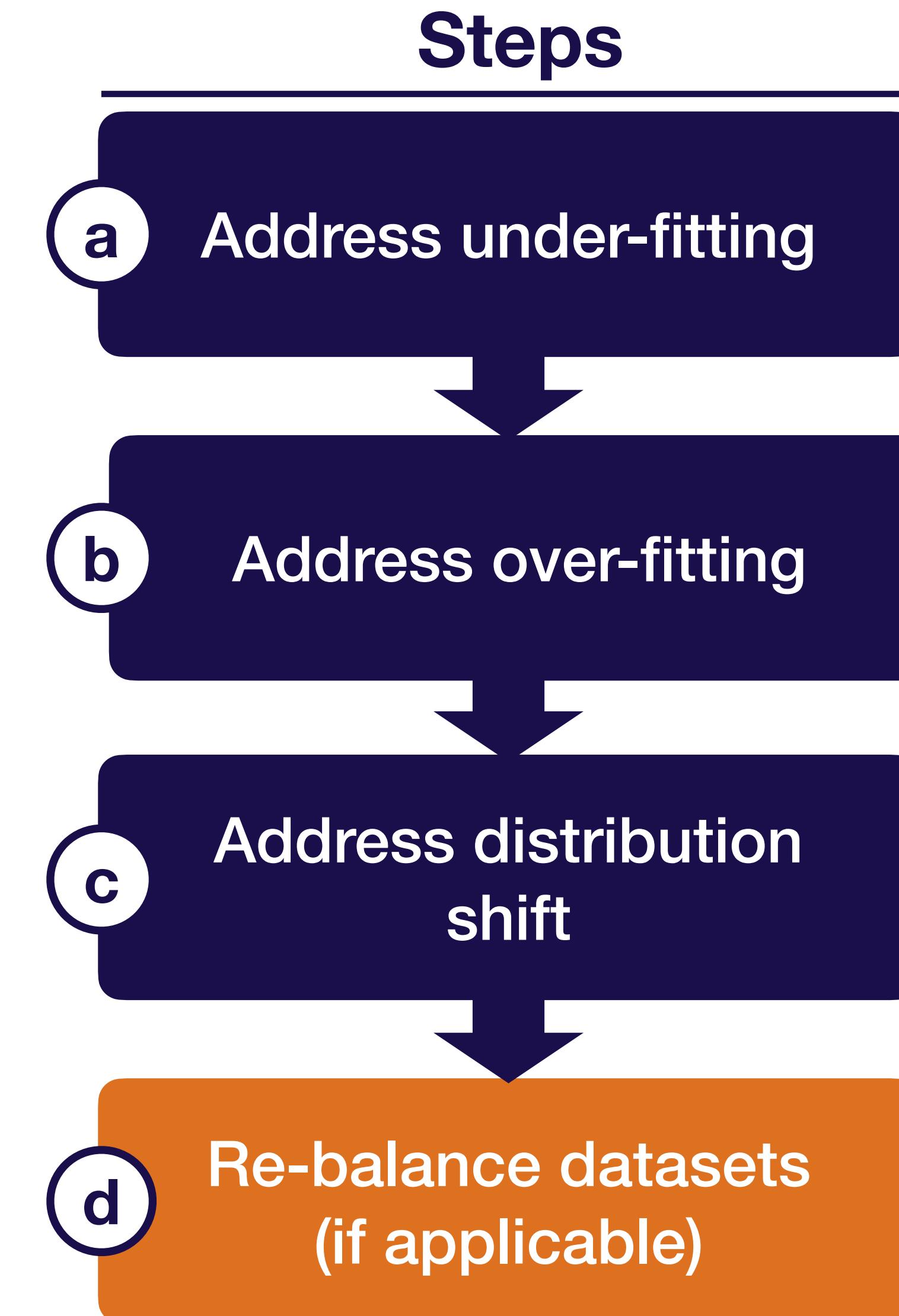
When should you consider using it?

- Access to labeled data from test distribution is limited
- Access to relatively similar data is plentiful

Types of domain adaptation

Type	Use case	Example techniques
Supervised	You have limited data from target domain	<ul style="list-style-type: none">• Fine-tuning a pre-trained model• Adding target data to train set
Un-supervised	You have lots of unlabeled data from target domain	<ul style="list-style-type: none">• Correlation Alignment (CORAL)• Domain confusion• CycleGAN

Prioritizing improvements (i.e., applied b-v)

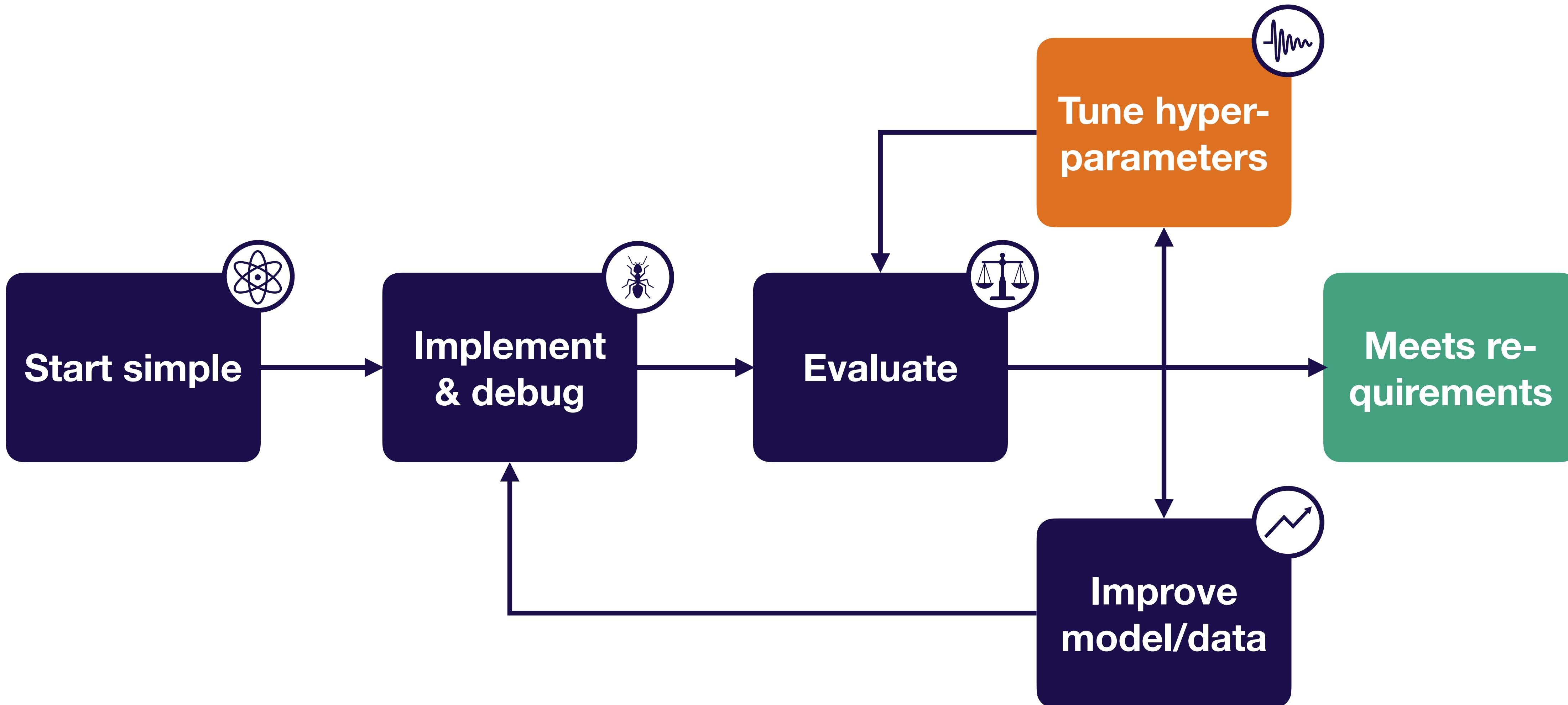


Rebalancing datasets

- If (test)-val looks significantly better than test, you overfit to the val set
- This happens with small val sets or lots of hyper parameter tuning
- When it does, recollect val data

Questions?

Strategy for DL troubleshooting



Hyperparameter optimization

Model & optimizer choices?

Network: ResNet

- How many layers?
- Weight initialization?
- Kernel size?
- Etc

Optimizer: Adam

- Batch size?
- Learning rate?
- beta1, beta2, epsilon?

Regularization

-

Running example



0 (no pedestrian)

1 (yes pedestrian)

Goal: 99% classification accuracy

Which hyper-parameters to tune?

Choosing hyper-parameters

- More sensitive to some than others
- Depends on choice of model
- Rules of thumb (only) to the right
- Sensitivity is relative to default values!
(e.g., if you are using all-zeros weight initialization or vanilla SGD, changing to the defaults will make a big difference)

Hyperparameter	Approximate sensitivity
Learning rate	High
Learning rate schedule	High
Optimizer choice	Low
Other optimizer params (e.g., Adam beta1)	Low
Batch size	Low
Weight initialization	Medium
Loss function	High
Model depth	Medium
Layer size	High
Layer params (e.g., kernel size)	Medium
Weight of regularization	Medium
Nonlinearity	Low

Method 1: manual hyperparam optimization

How it works

- Understand the algorithm
 - E.g., higher learning rate means faster less stable training
- Train & evaluate model
- Guess a better hyperparam value & re-evaluate
- Can be combined with other methods (e.g., manually select parameter ranges to optimizer over)

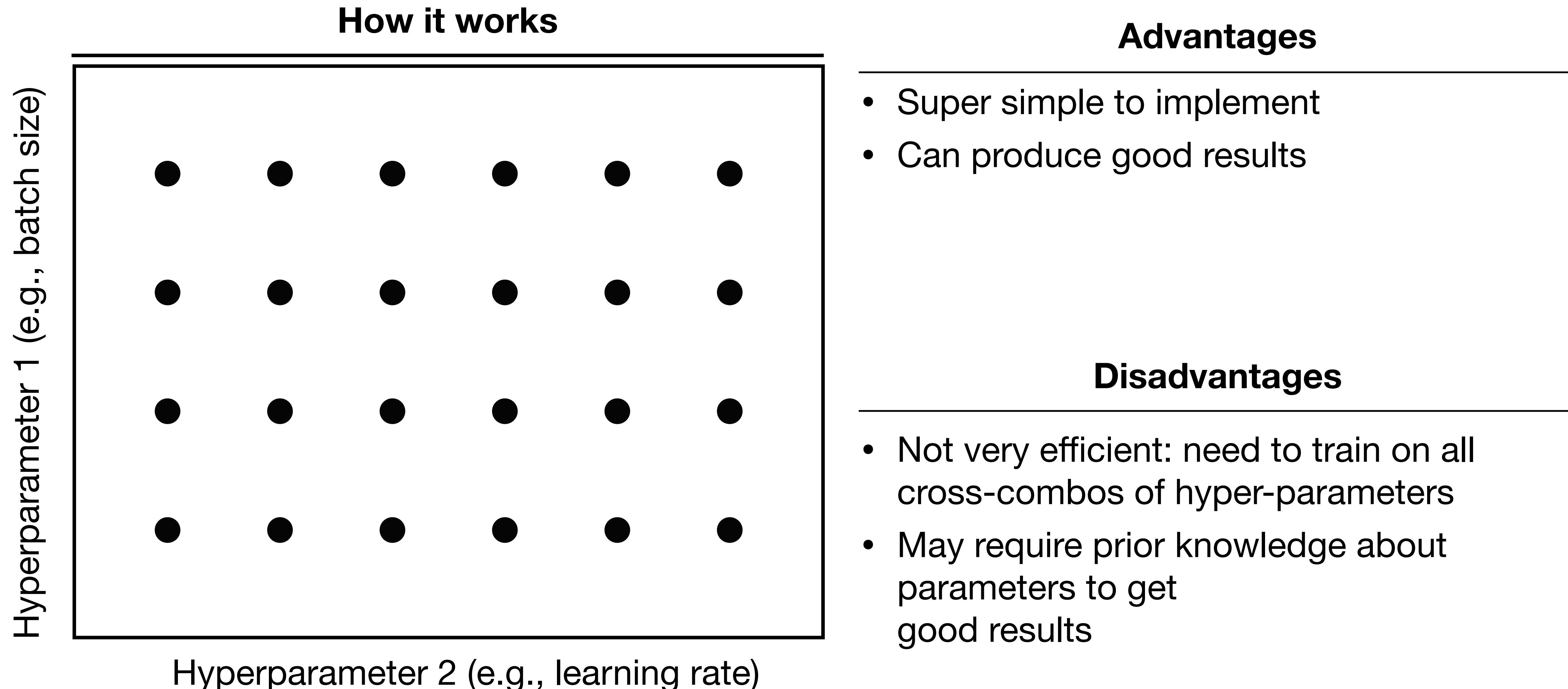
Advantages

- For a skilled practitioner, may require least computation to get good result

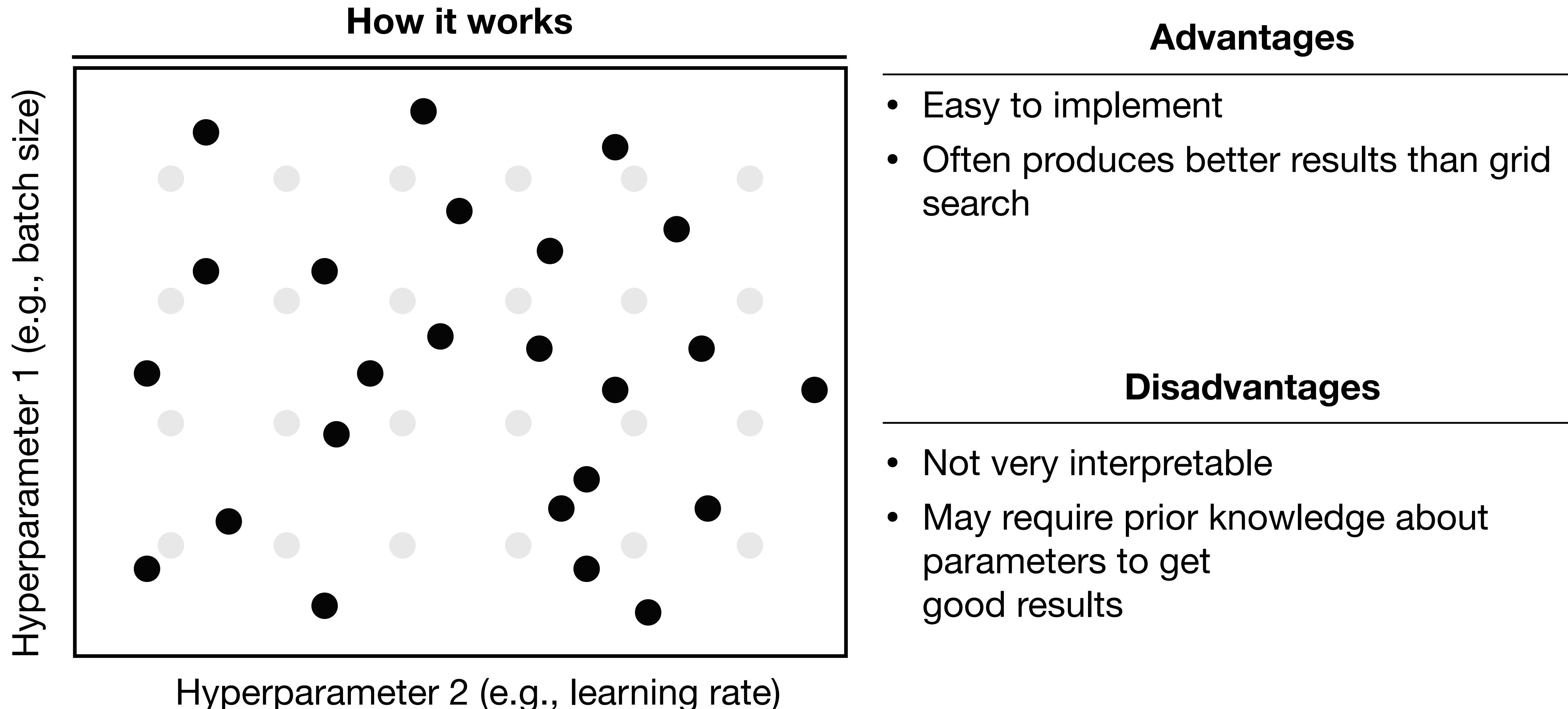
Disadvantages

- Requires detailed understanding of the algorithm
- Time-consuming

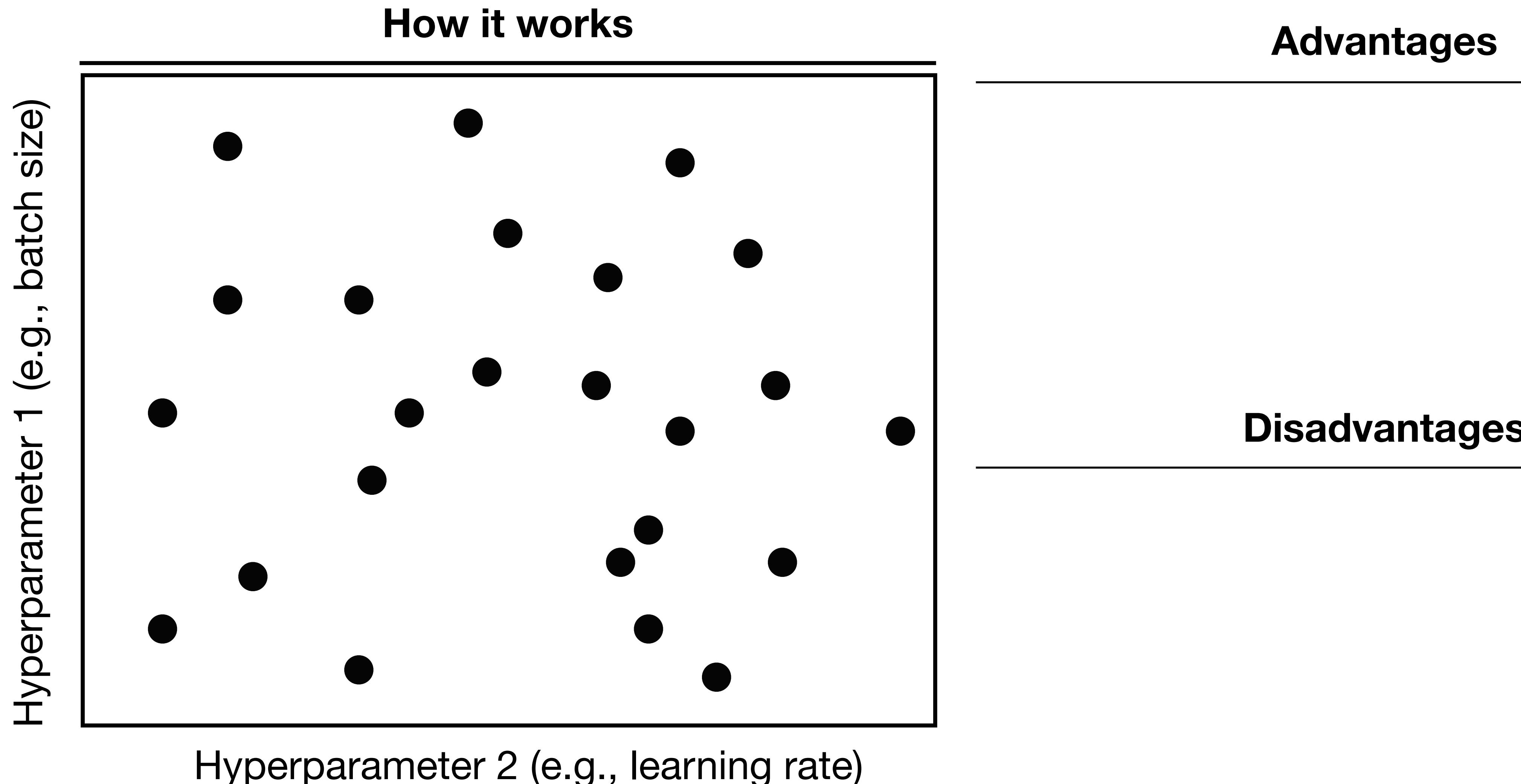
Method 2: grid search



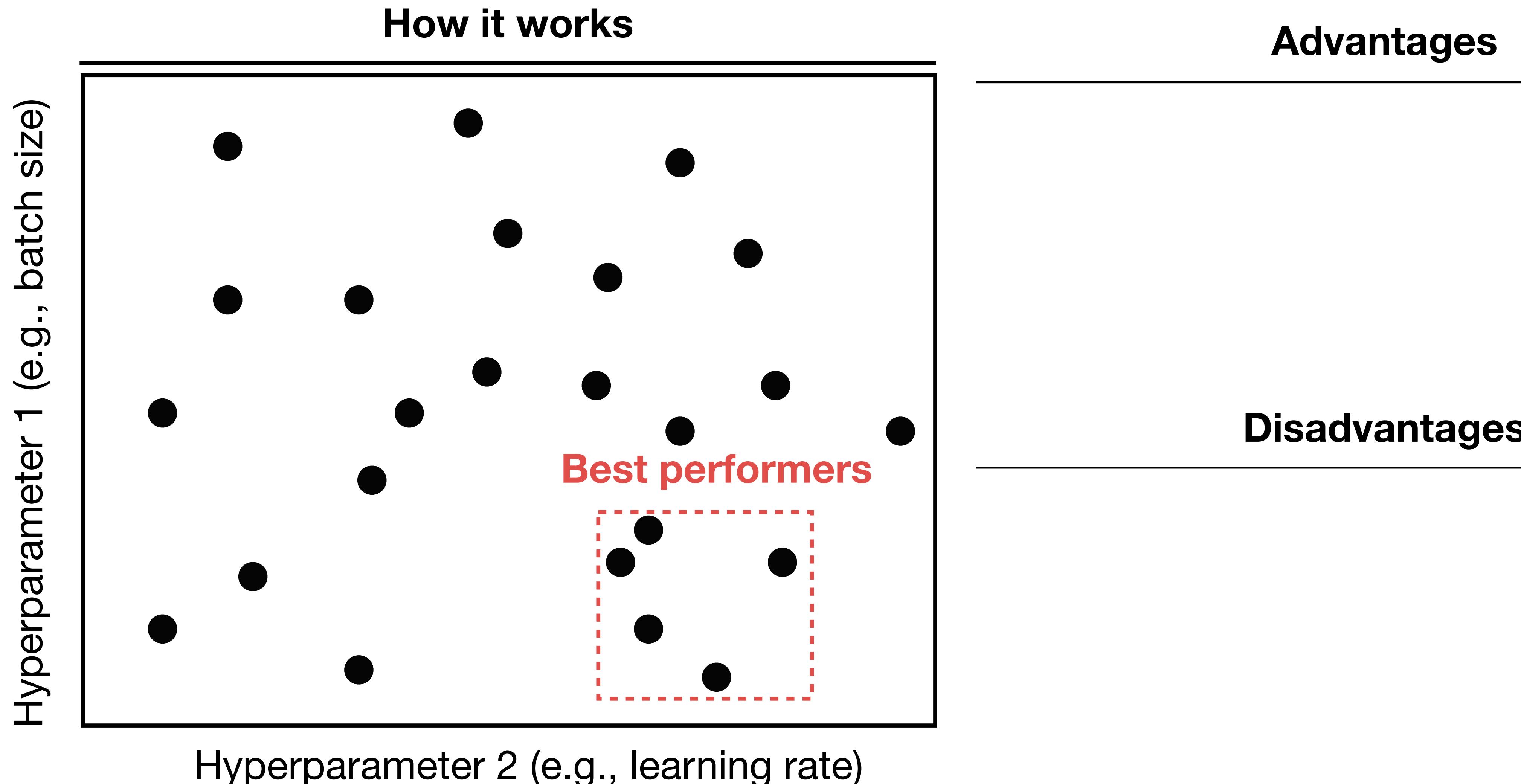
Method 3: random search



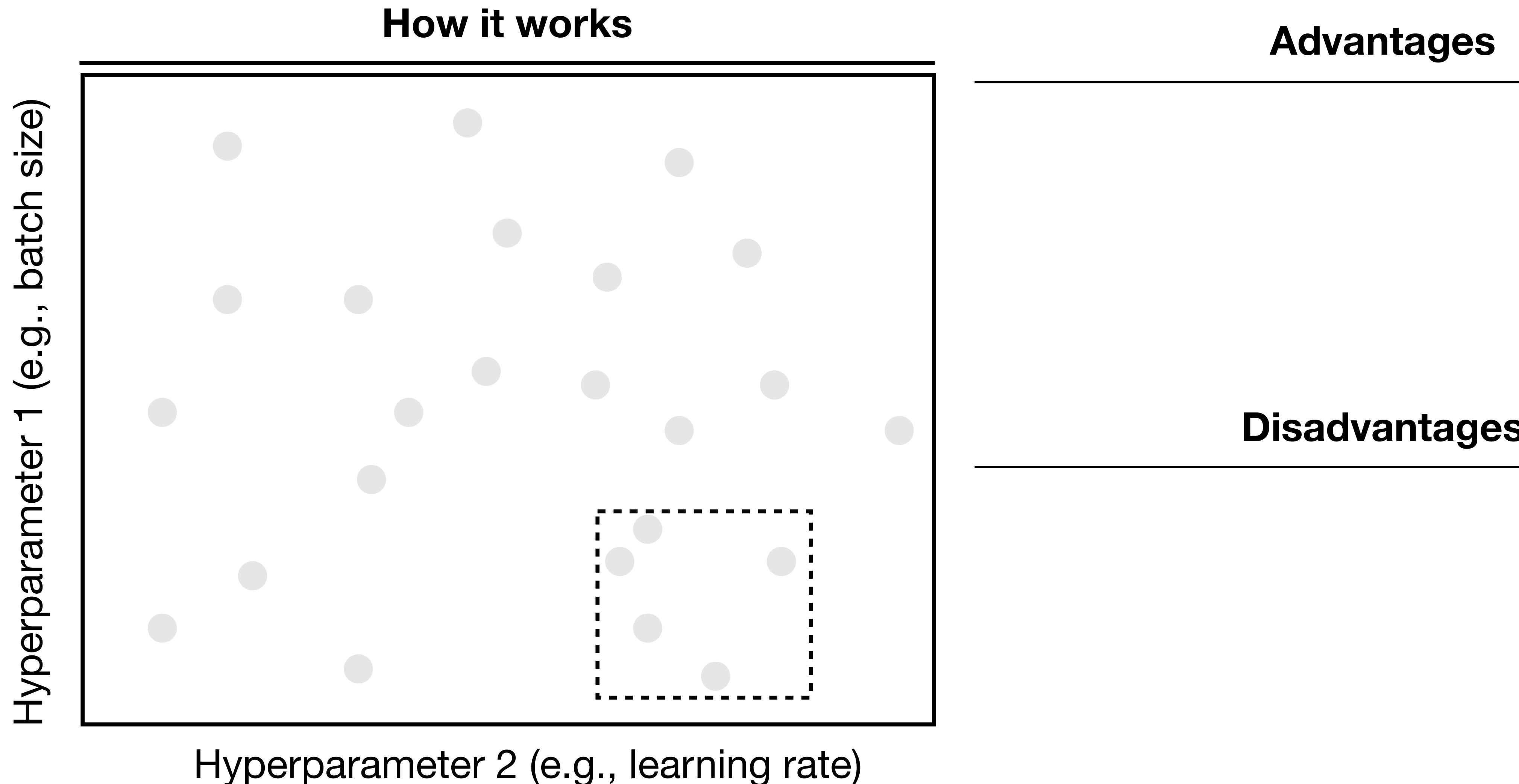
Method 4: coarse-to-fine



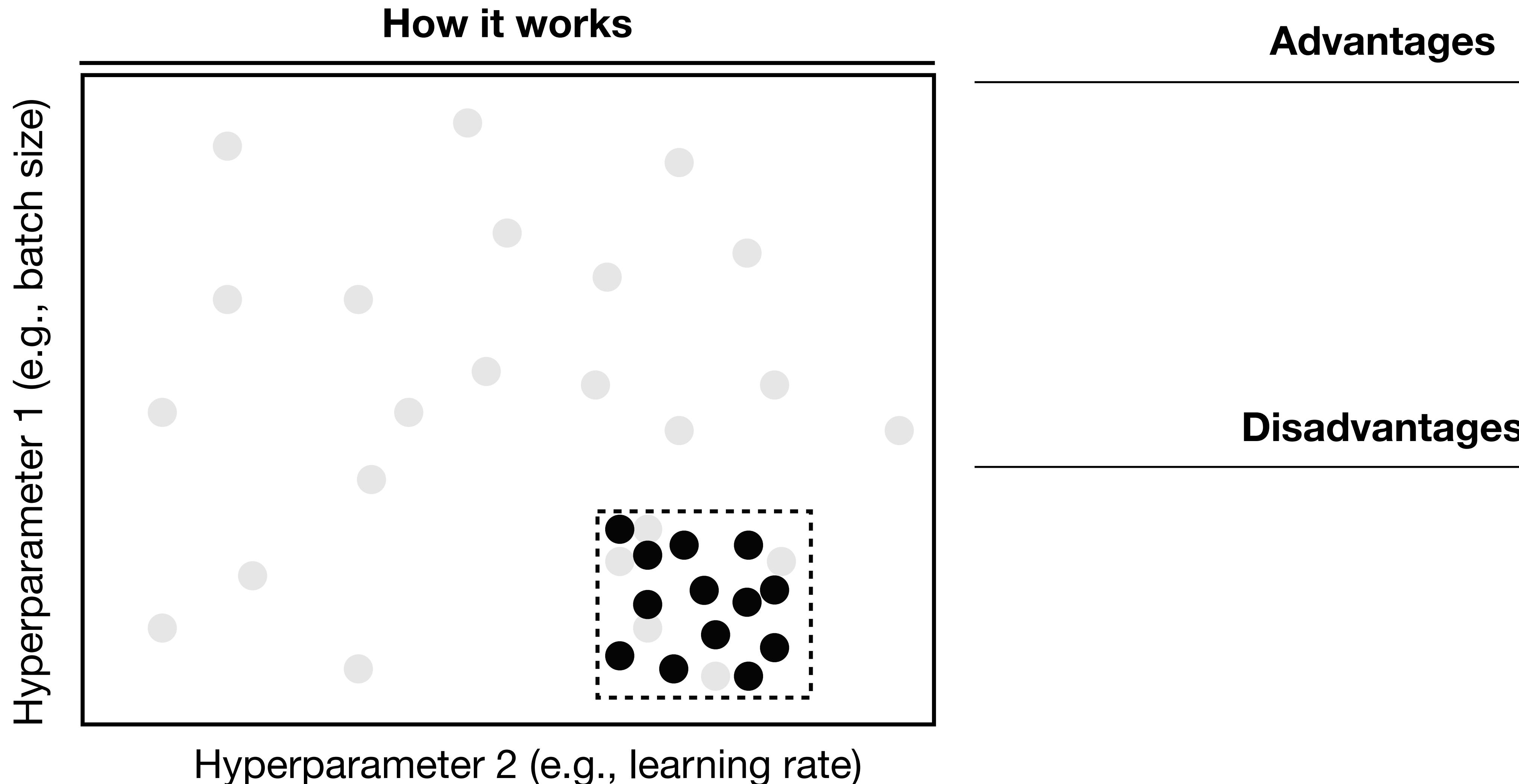
Method 4: coarse-to-fine



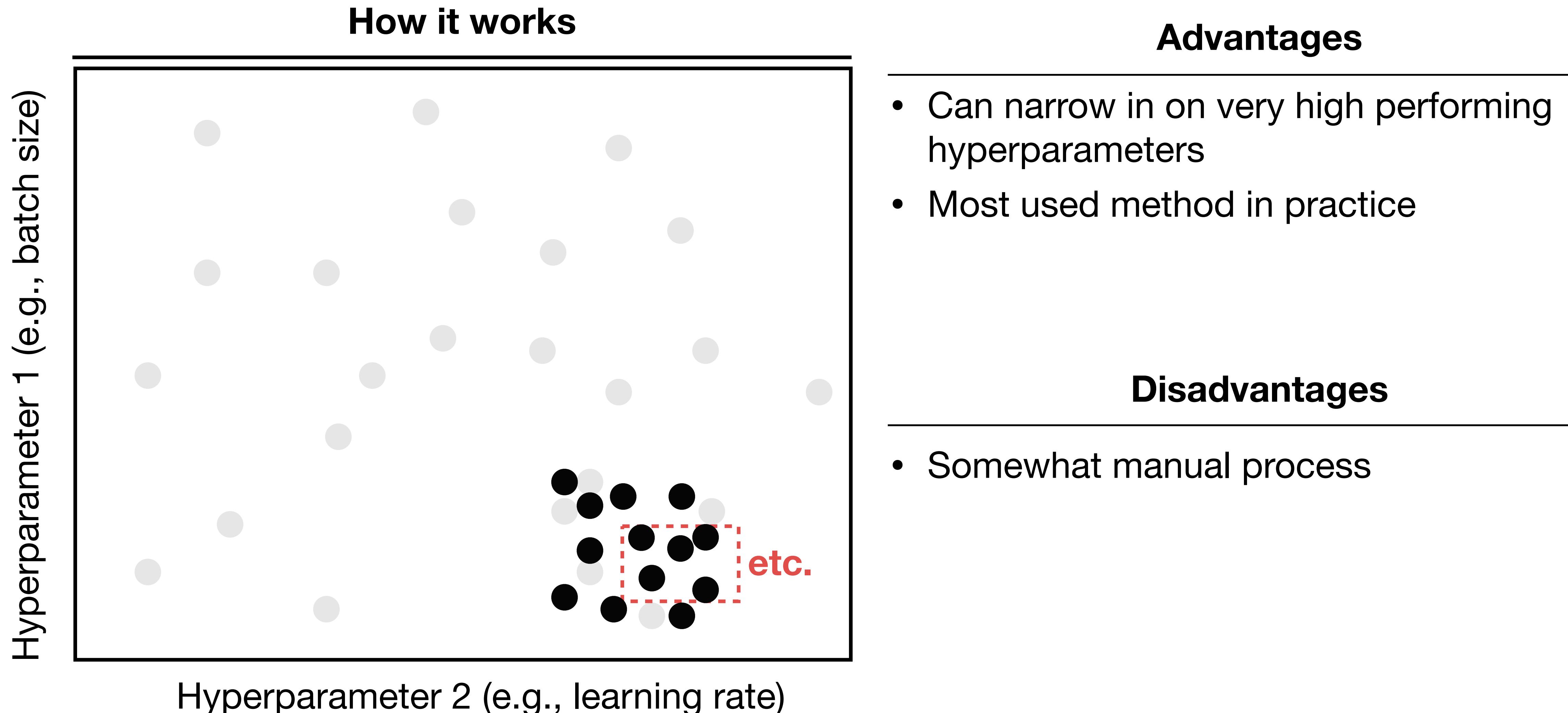
Method 4: coarse-to-fine



Method 4: coarse-to-fine



Method 4: coarse-to-fine



Method 5: Bayesian hyperparam opt

How it works (at a high level)

- Start with a prior estimate of parameter distributions
- Maintain a probabilistic model of the relationship between hyper-parameter values and model performance
- Alternate between:
 - Training with the hyper-parameter values that maximize the expected improvement
 - Using training results to update our probabilistic model
- To learn more, see:

<https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f>

Advantages

- Generally the most efficient hands-off way to choose hyperparameters

Disadvantages

- Difficult to implement from scratch
- Can be hard to integrate with off-the-shelf tools

Method 5: Bayesian hyperparam opt

How it works (at a high level)

- Start with a prior estimate of parameter distributions
- Maintain a probabilistic model of the relationship between hyper-parameter values and model p
- Alternate between:
 - Training with the values that maximize improvement
 - Using training results to update our probabilistic model
- To learn more, see:

More on tools to do this automatically in the infrastructure & tooling lecture!

Advantages

- Generally the most efficient hands-off way to choose hyperparameters
- Can be hard to integrate with off-the-shelf tools

<https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f>

Summary of how to optimize hyperparams

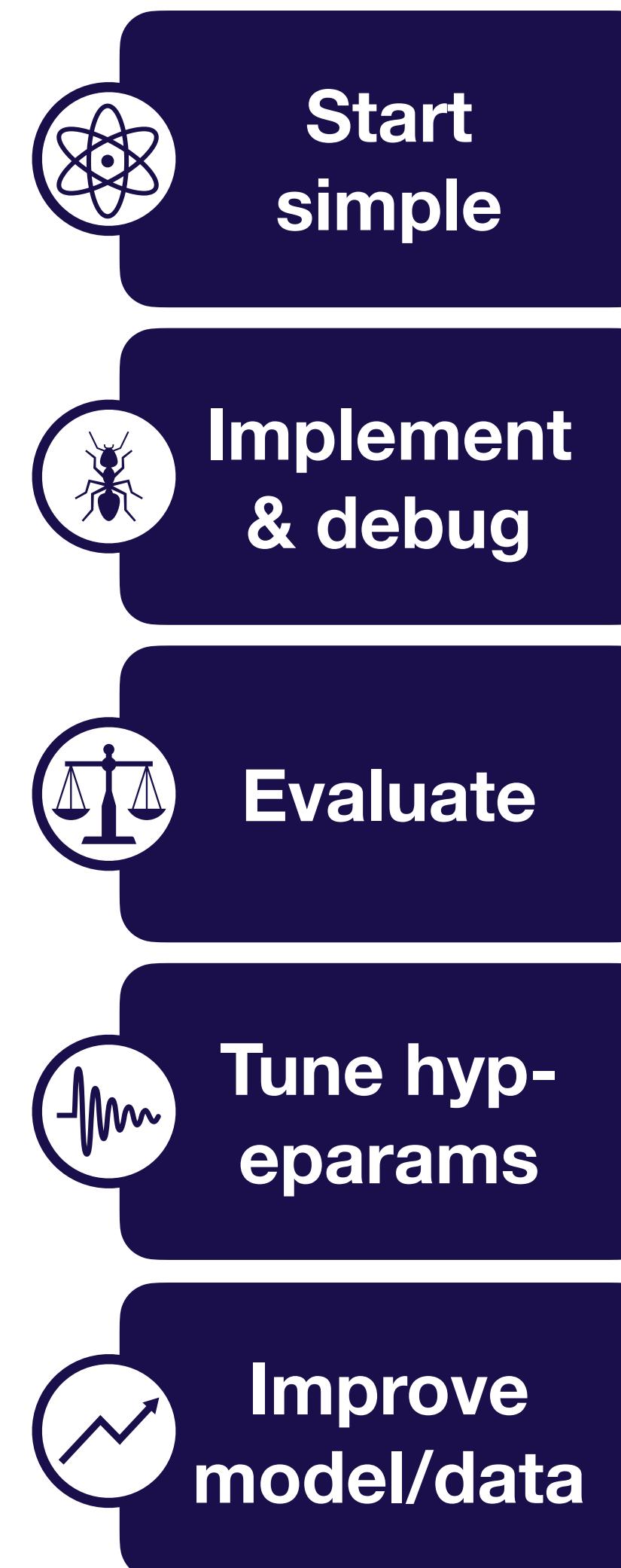
- Coarse-to-fine random searches
- Consider Bayesian hyper-parameter optimization solutions as your codebase matures

Questions?

Conclusion

- **DL debugging is hard due to many competing sources of error**
- **To train bug-free DL models, we treat building our model as an iterative process**
- **The following steps can make the process easier and catch errors as early as possible**

How to build bug-free DL models



- Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
- Once model runs, overfit a single batch & reproduce a known result
- Apply the bias-variance decomposition to decide what to do next
- Use coarse-to-fine random searches
- Make your model bigger if you underfit; add data or regularize if you overfit

Where to go to learn more

- Andrew Ng's book Machine Learning Yearning ([http://www.mlyarning.org/](http://www.mlyearning.org/))
- The following Twitter thread:
<https://twitter.com/karpathy/status/1013244313327681536>
- This blog post:
<https://pcc.cs.byu.edu/2017/10/02/practical-advice-for-building-deep-neural-networks/>

Thank you!