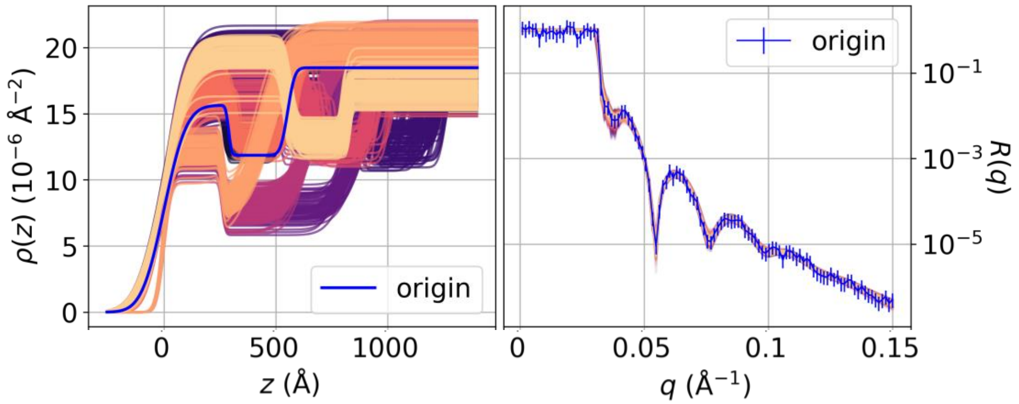
Machine learning for reflectometry: Resolving ambiguity

Vladimir Starostin and Frank Schreiber
http://www.soft-matter.uni-tuebingen.de
A. Greco, V. Munteanu, C. Völter, M. Romodin, D. Lapkin, L. Pithan, A. Gerlach, A. Hinderhofer et al.

special thanks to
S. Kowarik group
synchrotrons
neutron facilities
ML excellence cluster



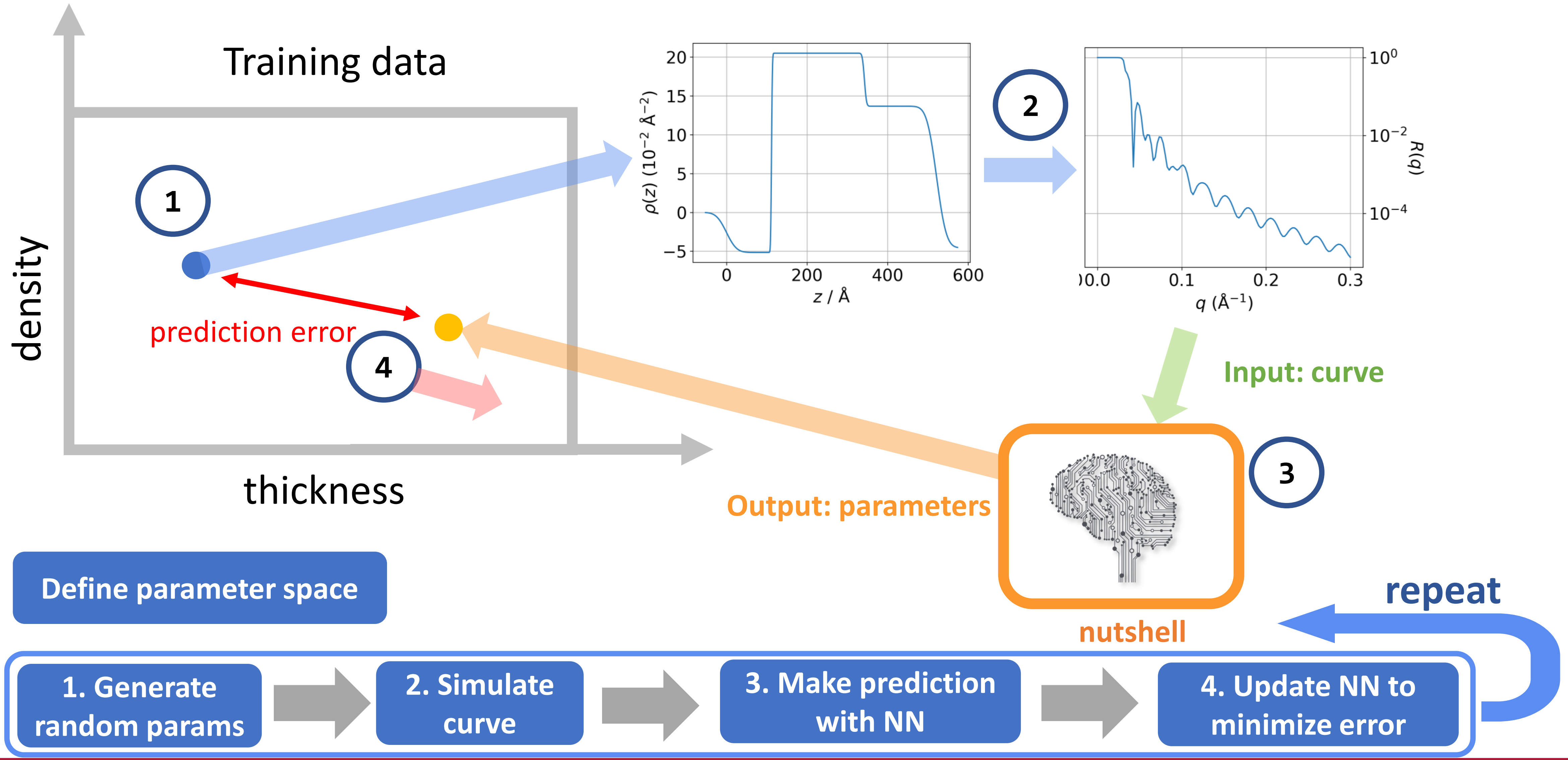
Contents

1. **`mlreflect` 1.0** - recap
2. **`mlreflect` 2.0** - ML + prior knowledge
3. **`mlreflows` (3.0)** - probabilistic ML approach

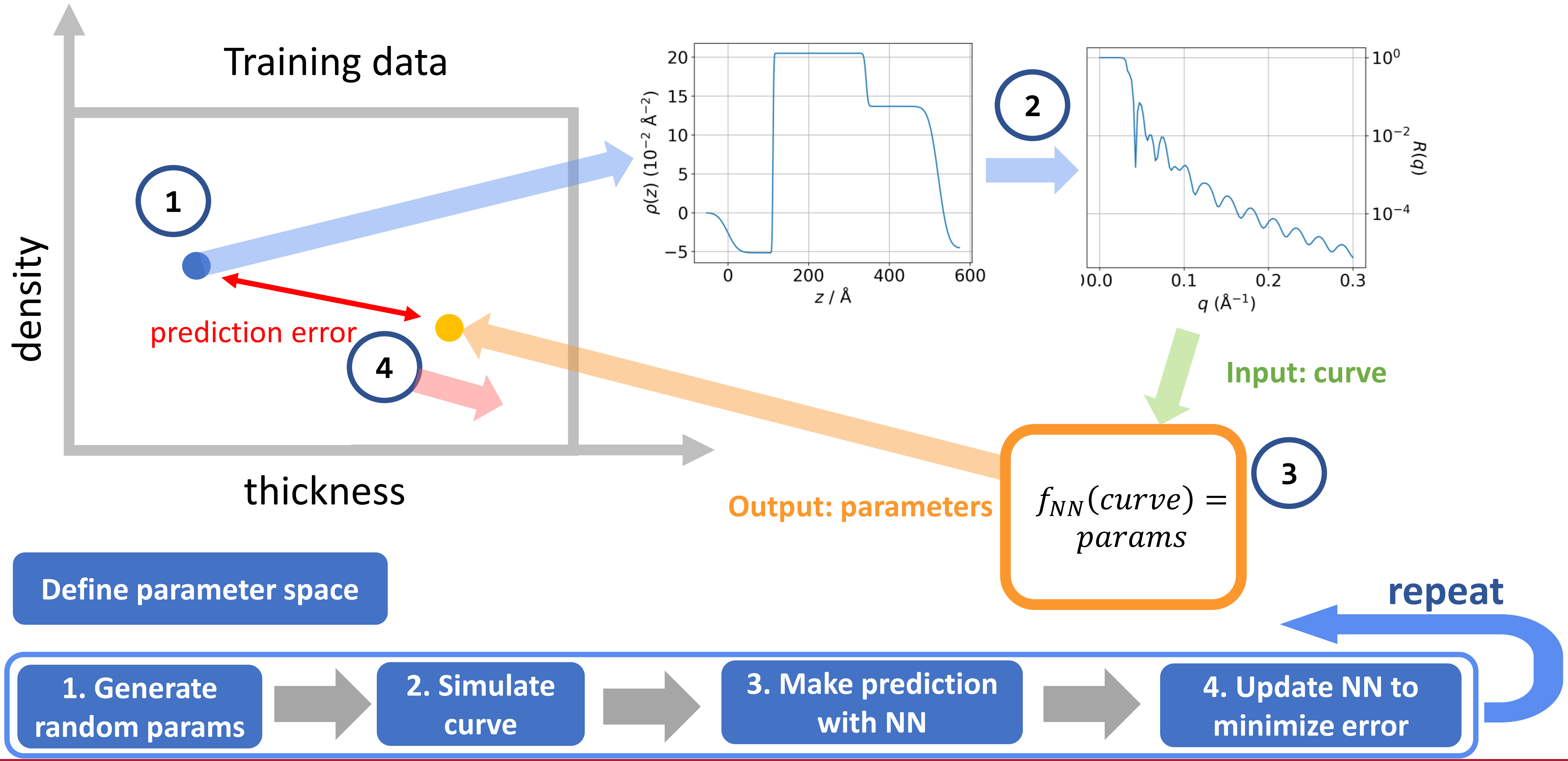
Standard Machine Learning approach (`mlreflect 1.0`)

Why `mlreflect` does not scale to higher number of parameters?

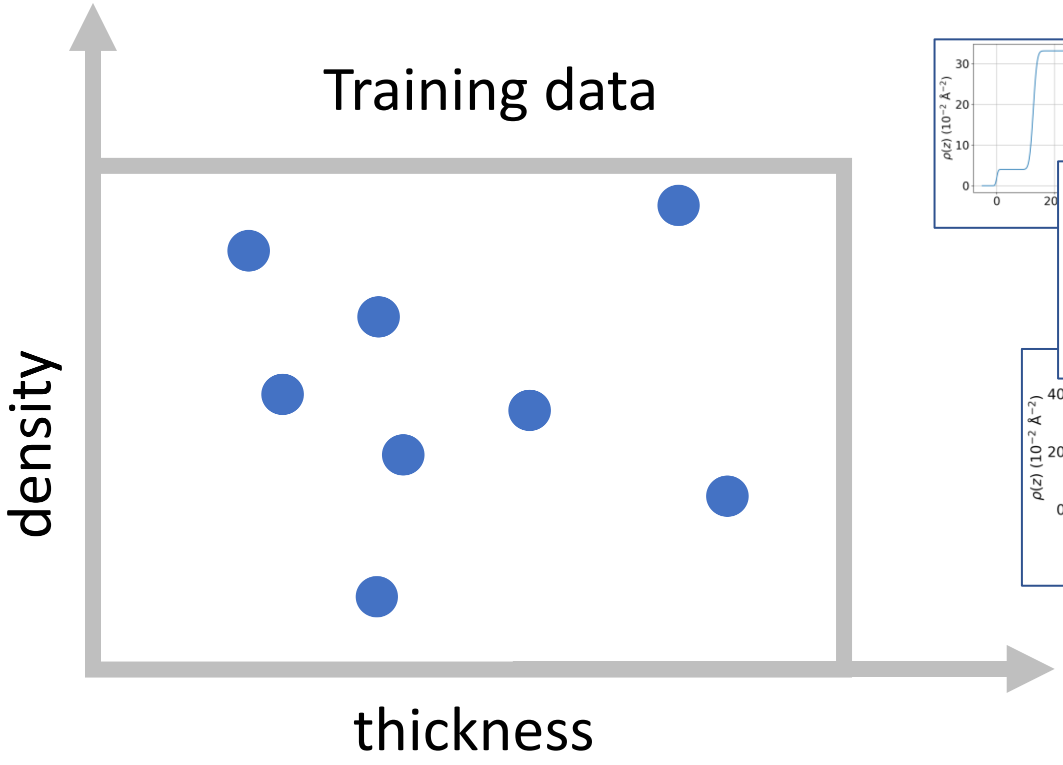
Training `mlreflect 1.0` in a nutshell

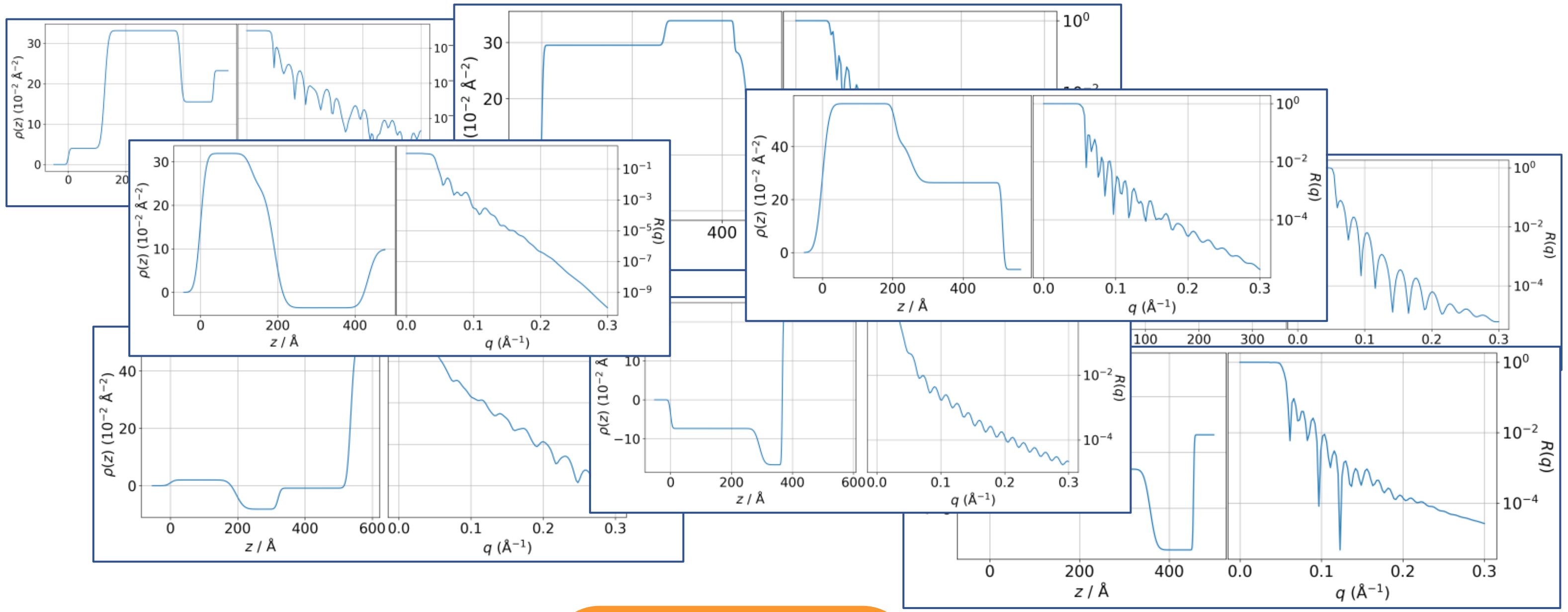


Training `mlreflect 1.0` in a nutshell



Training `mlreflect 1.0` in a nutshell





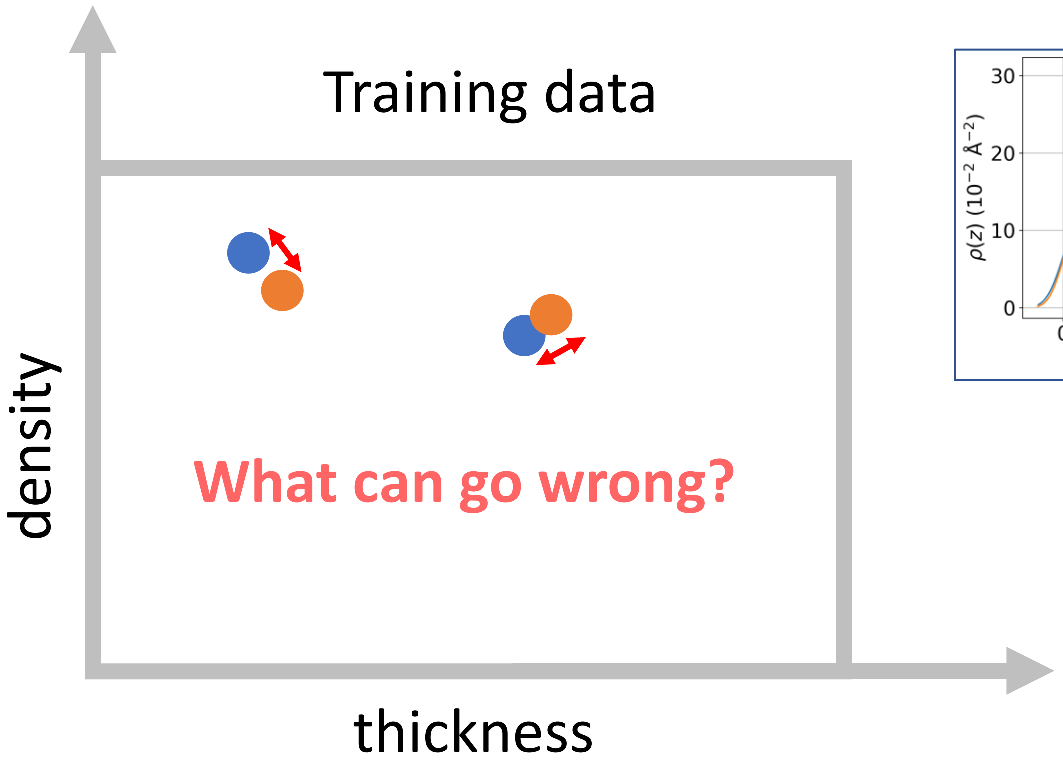
Define parameter space

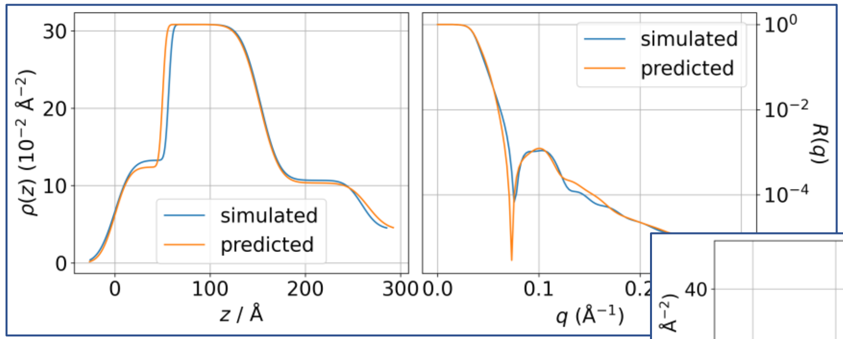
$$f_{NN}(curve) = params$$

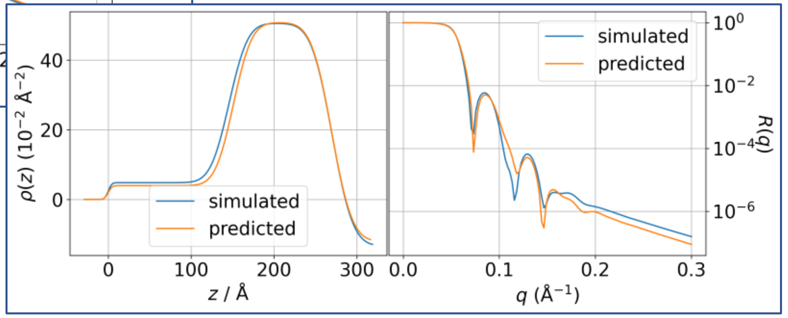
repeat

1. Generate random params → 2. Simulate curve → 3. Make prediction with NN → 4. Update NN to minimize error

Training `mlreflect 1.0` in a nutshell







$$f_{NN}(curve) = params$$

Define parameter space

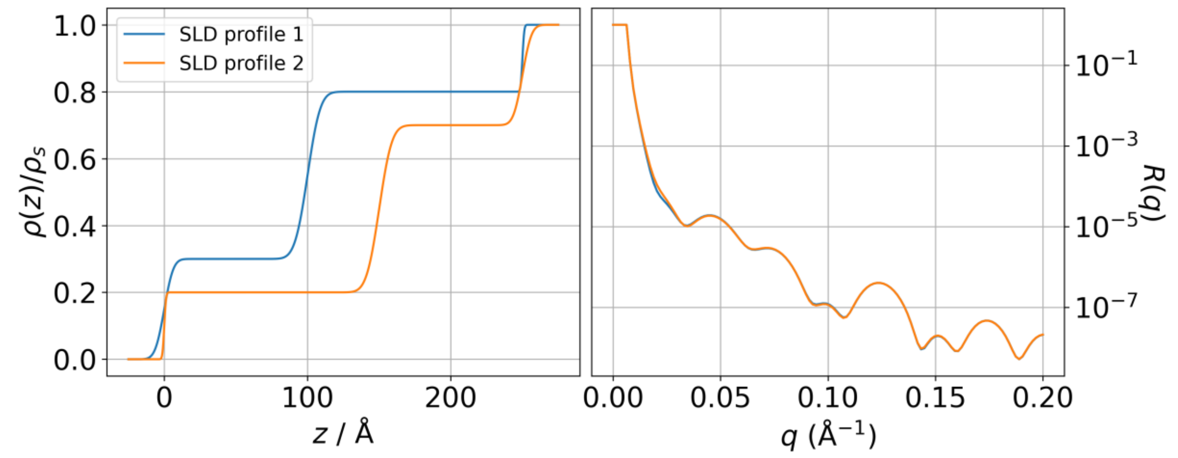
1. Generate random params → **2. Simulate curve** → **3. Make prediction with NN** → **4. Update NN to minimize error**

repeat

Ambiguity in the reflectometry data (theoretical examples)

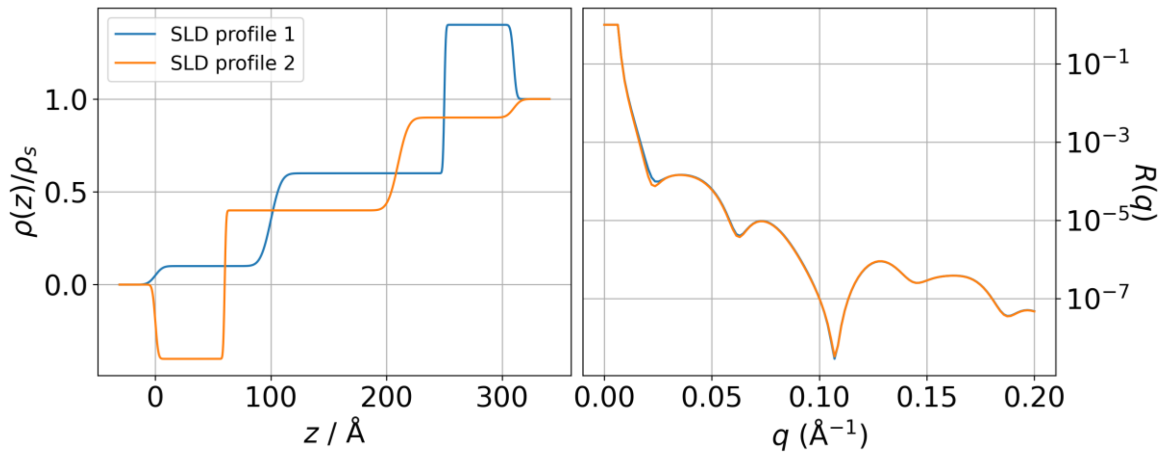
SLD profiles

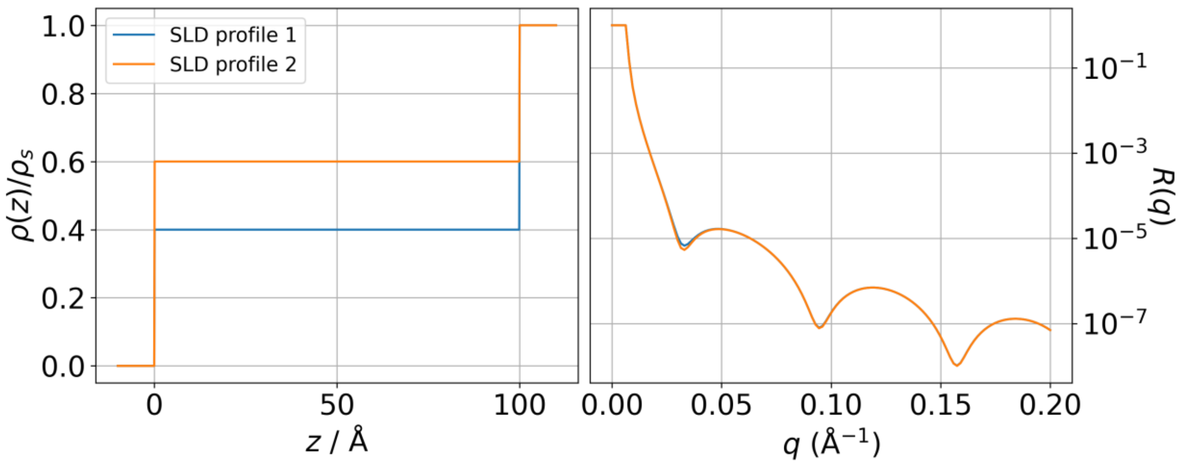
Reflectivity curves



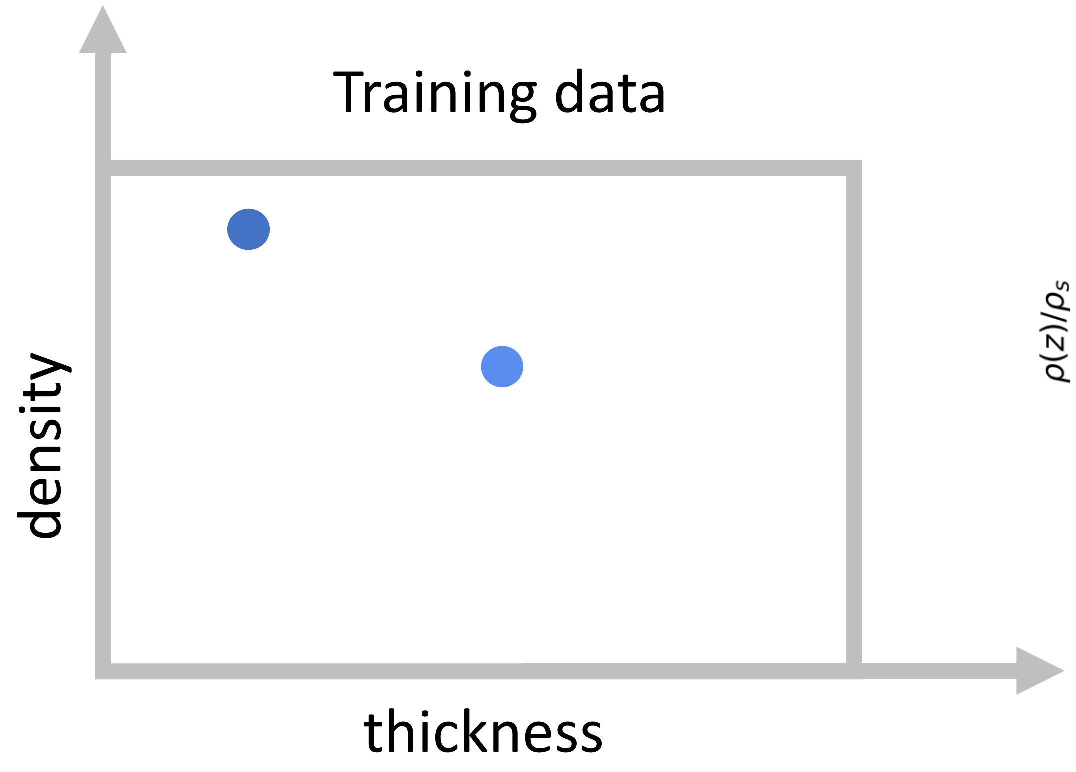
Reasons for the ambiguity:

- “Phase” problem
- Noise (counting statistics)
- Finite q range
- Finite number of q points
- …

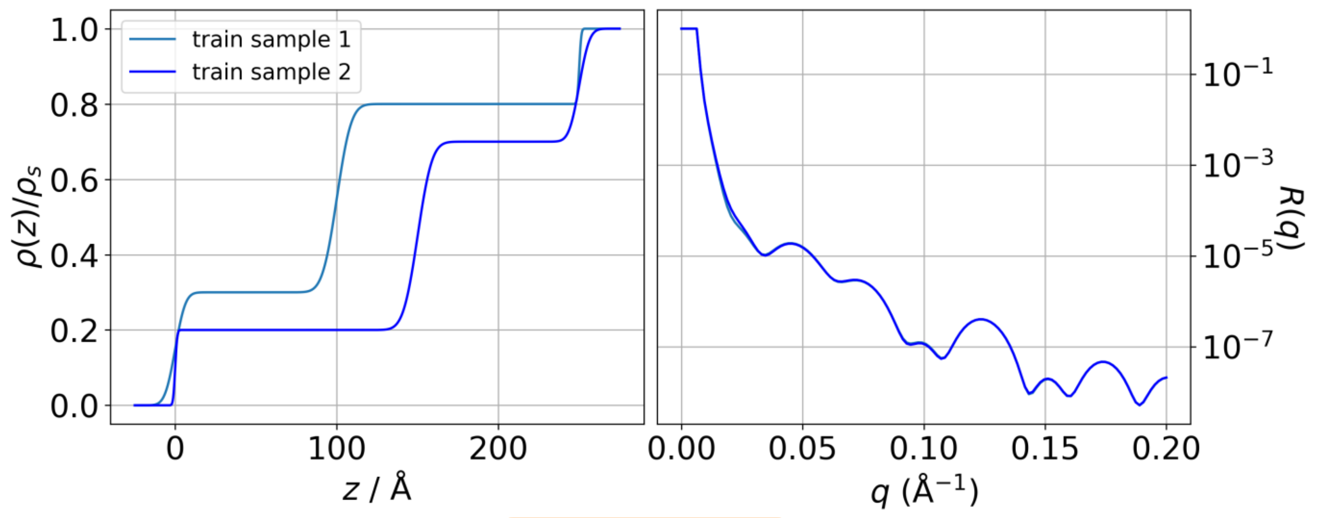




Training `mlreflect 1.0` in a nutshell

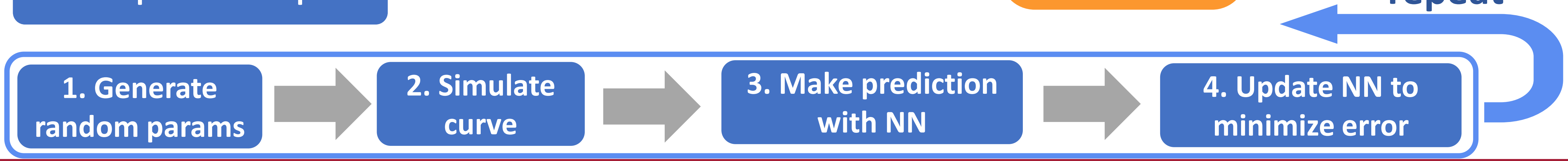


Training with ambiguity

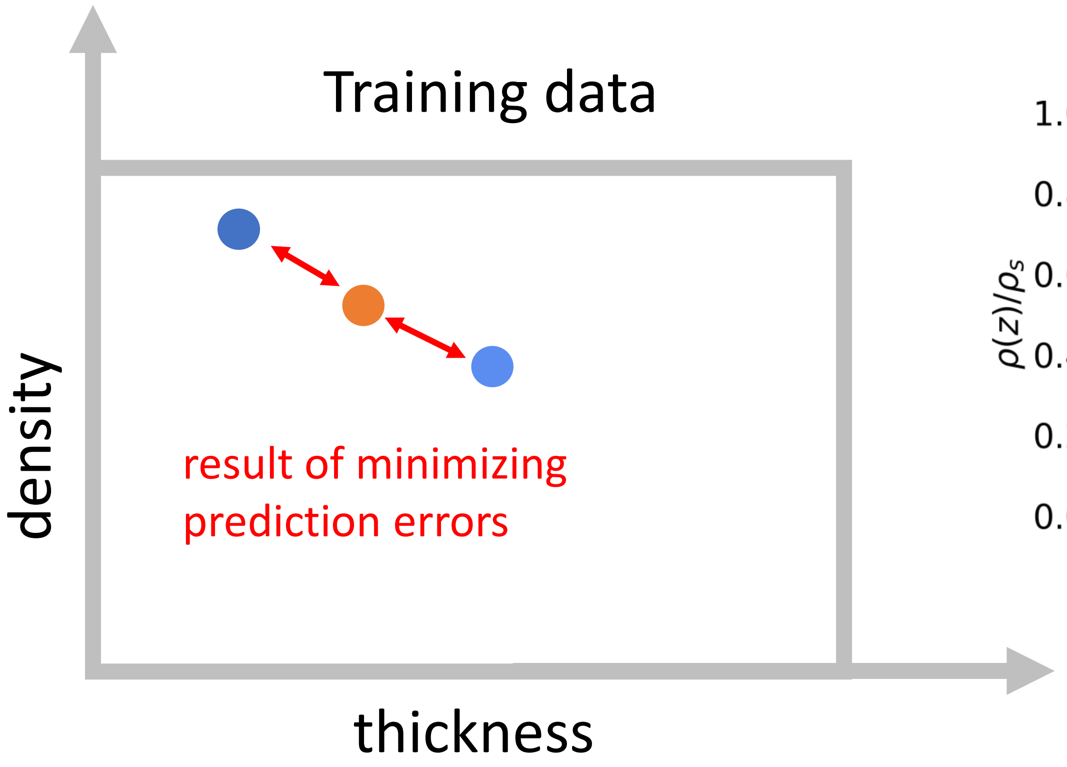


$$f_{NN}(curve) = params$$

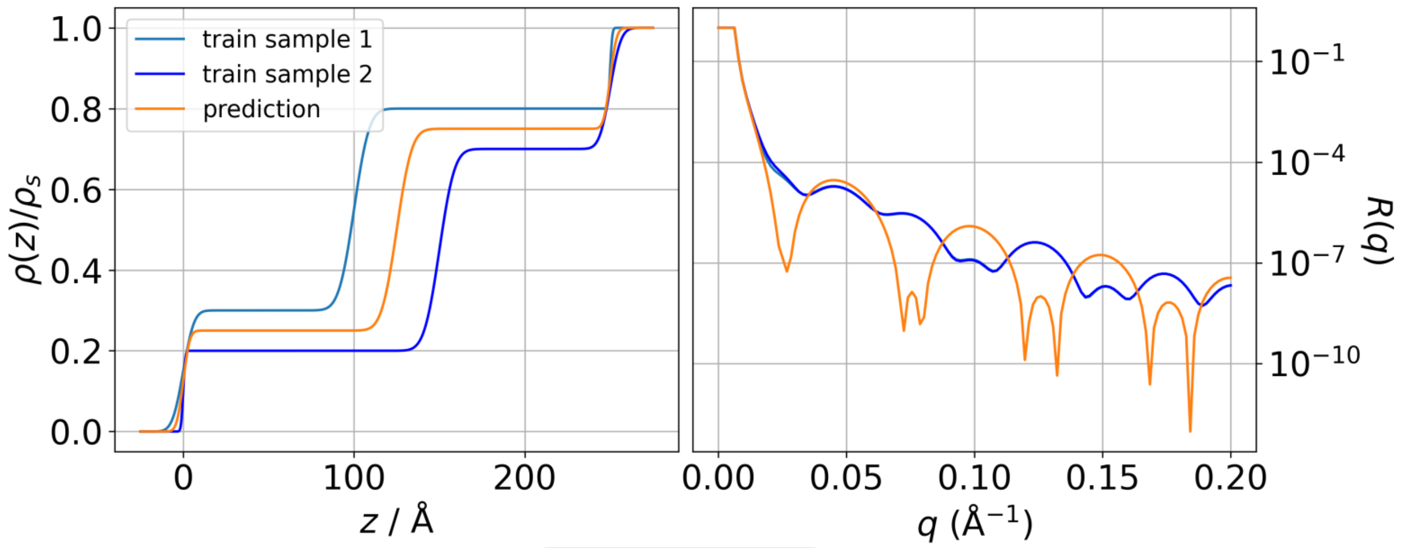
Define parameter space


1. Generate random params
2. Simulate curve
3. Make prediction with NN
4. Update NN to minimize error
repeat

Training `mlreflect 1.0` in a nutshell



Training with ambiguity



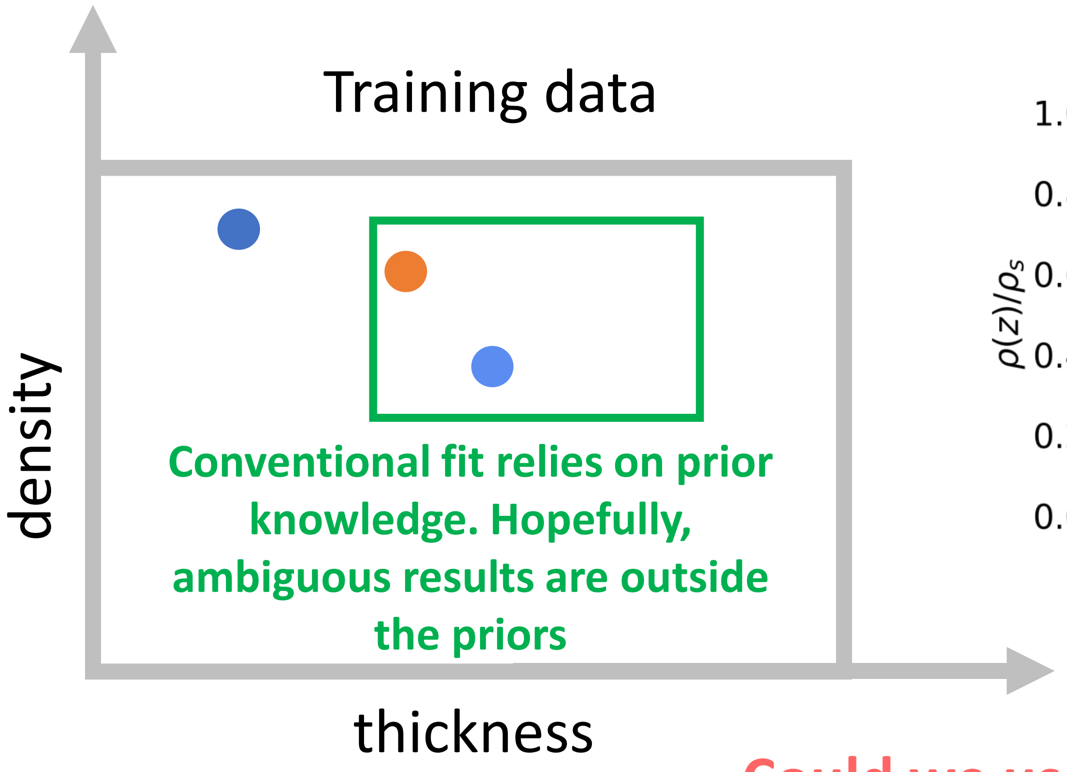
$$f_{NN}(curve) = params$$

Define parameter space

1. Generate random params → 2. Simulate curve → 3. Make prediction with NN → 4. Update NN to minimize error

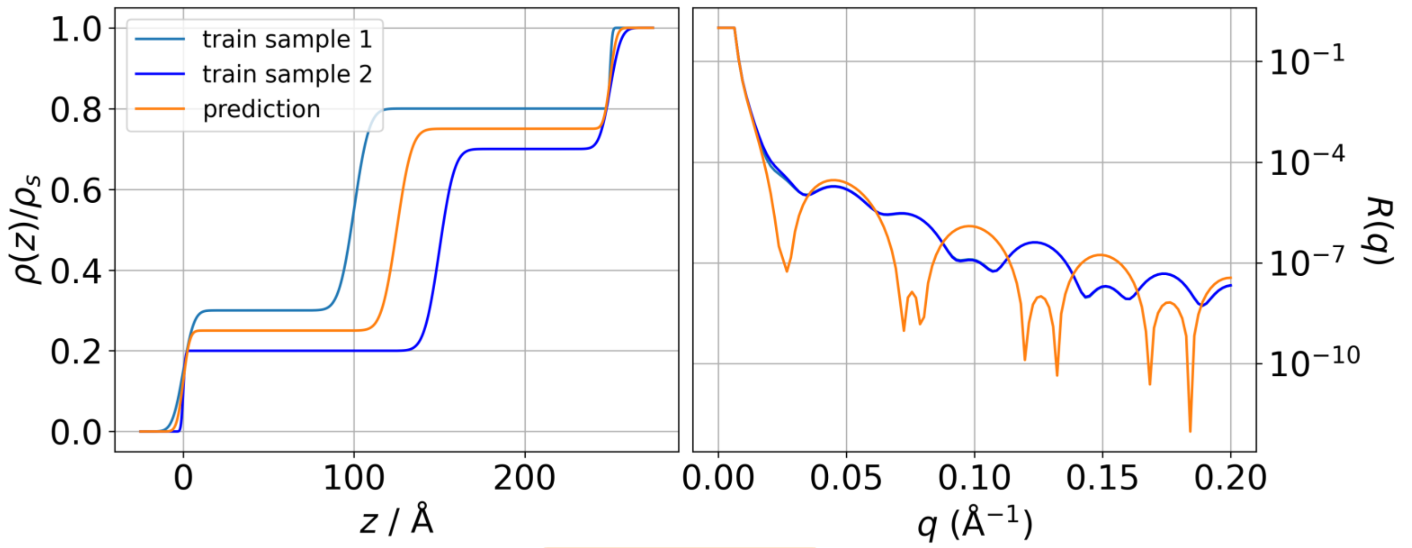
repeat

Training `mlreflect 1.0` in a nutshell



Conventional fit relies on prior knowledge. Hopefully, ambiguous results are outside the priors

Training with ambiguity



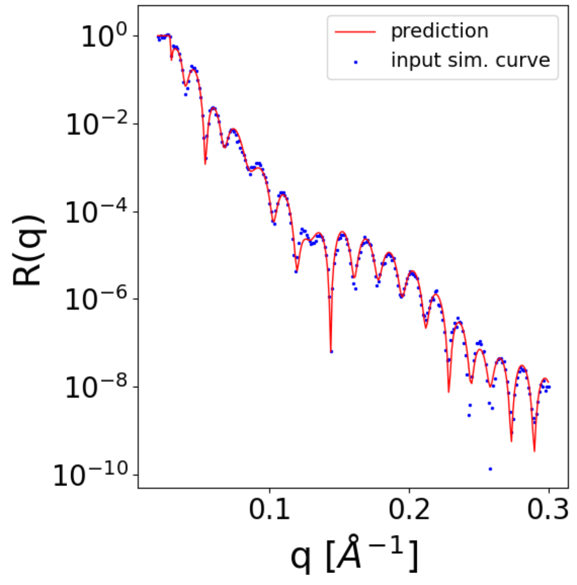
Could we use the same idea with NN?

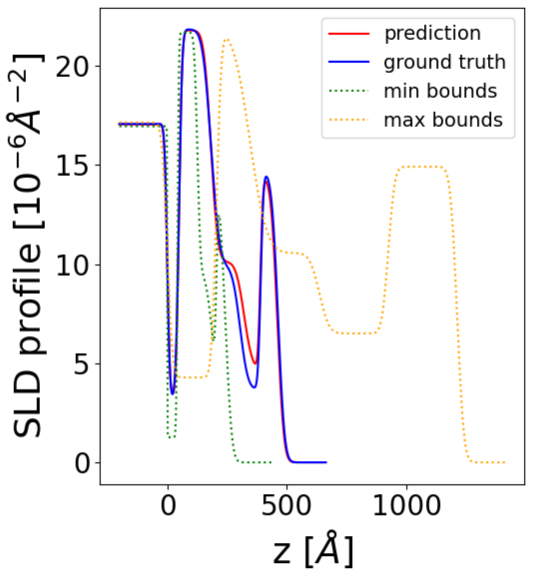
$$f_{NN}(curve) = params$$

Define parameter space

1. Generate random params → 2. Simulate curve → 3. Make prediction with NN → 4. Update NN to minimize error

repeat

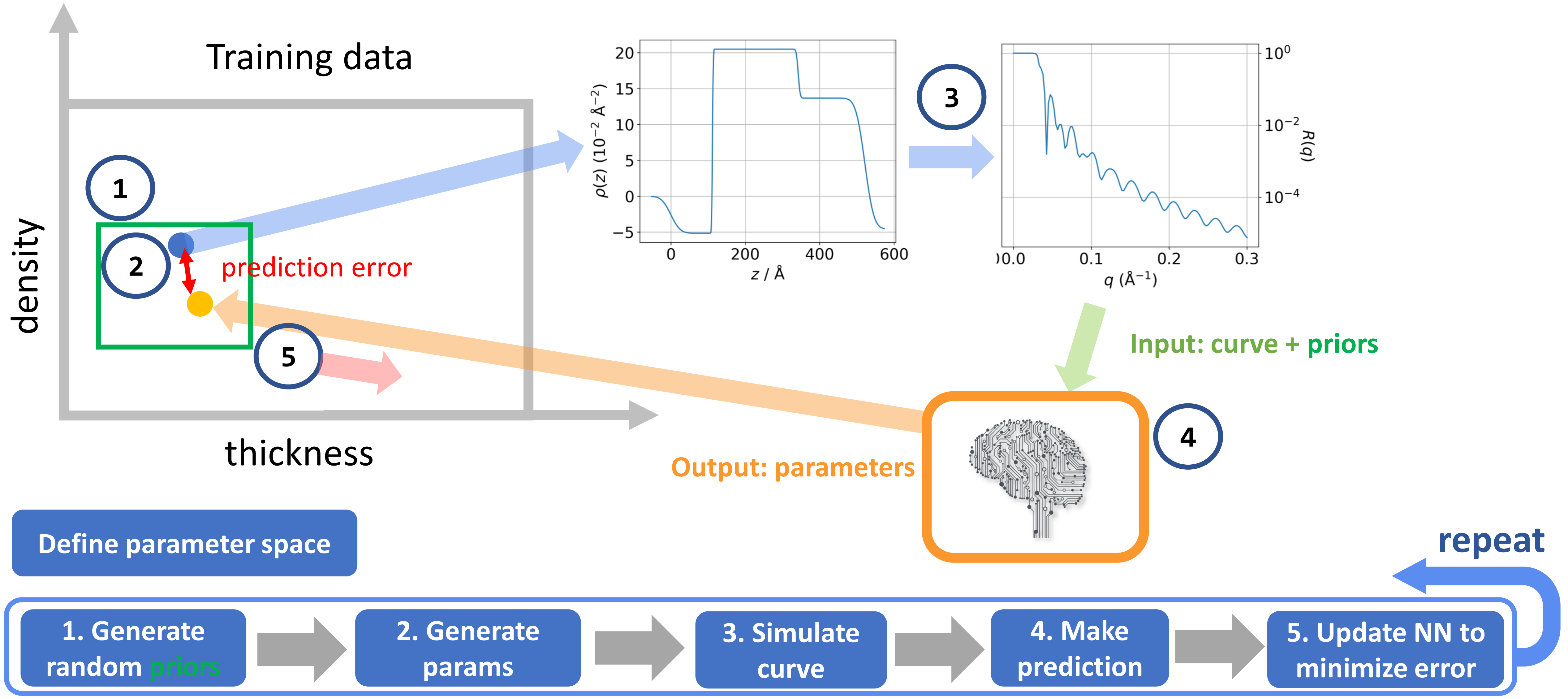




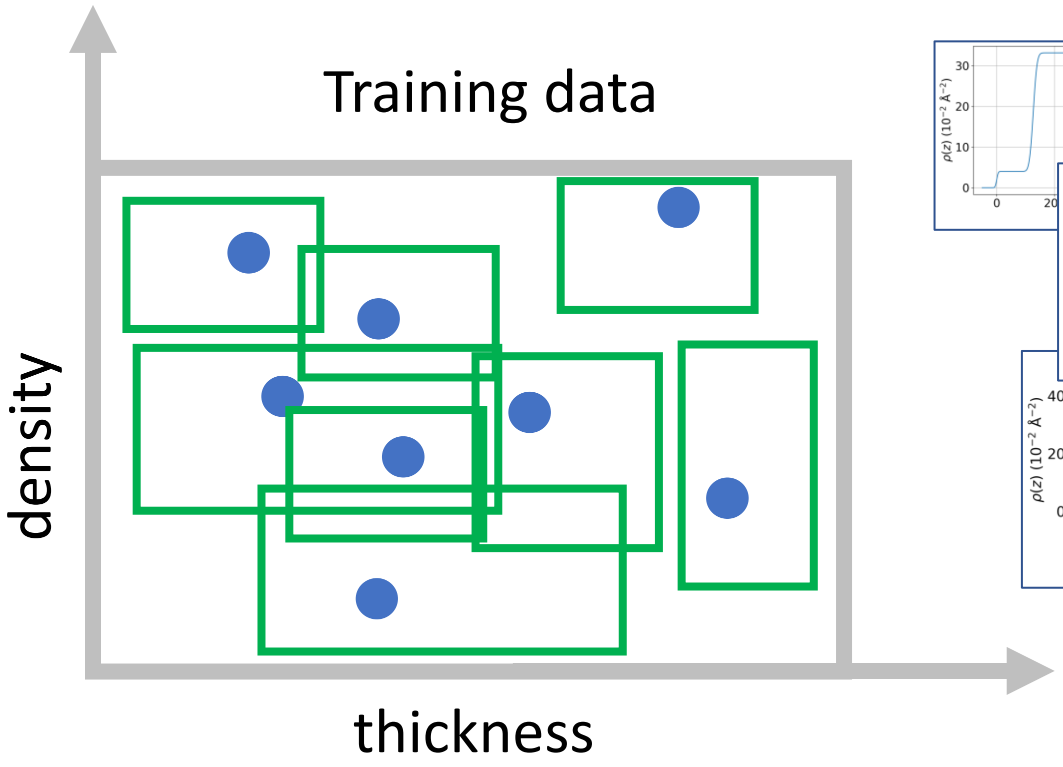
Machine Learning + Prior Knowledge (`mlreflect 2.0`)

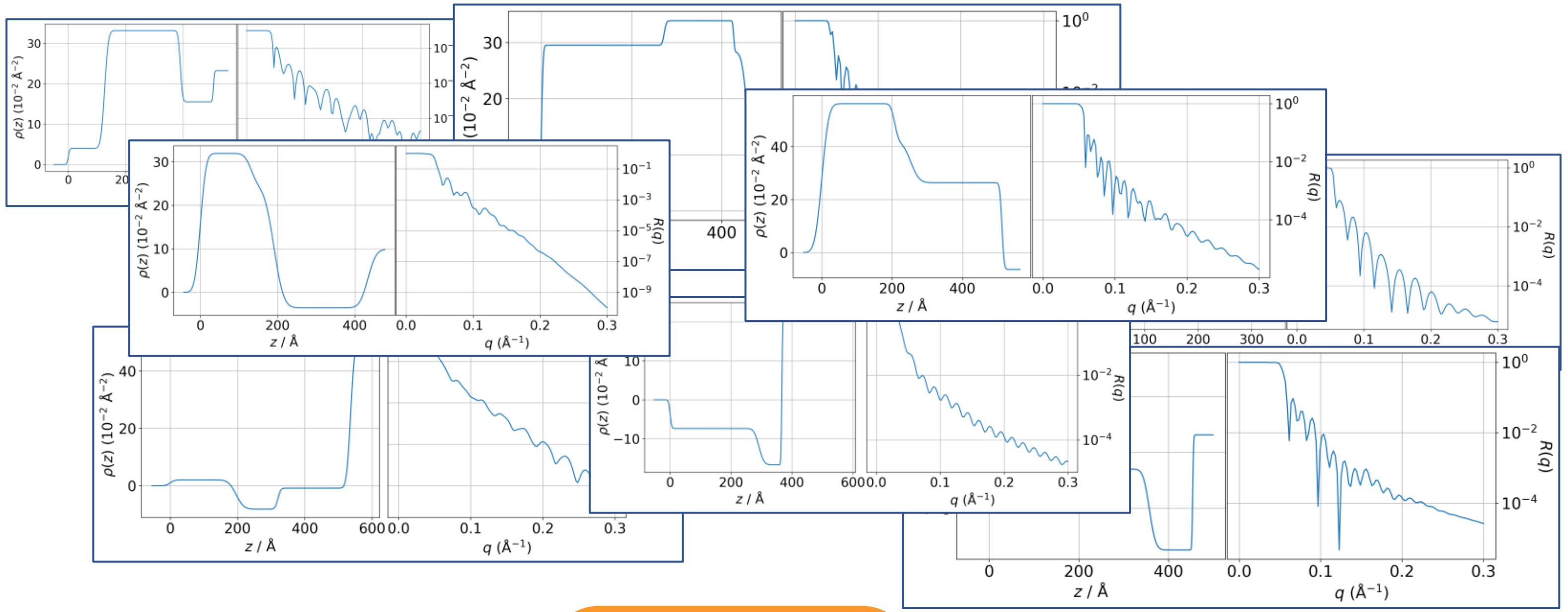
Addressing ambiguity by incorporating physical knowledge into the model

Training mlreflect 2.0



Training `mlreflect 1.0` in a nutshell





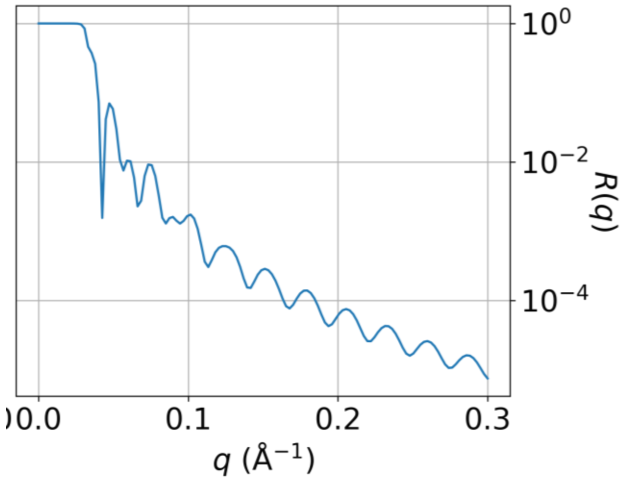
Define parameter space

repeat

1. Generate random priors → 2. Generate params → 3. Simulate curve → 4. Make prediction → 5. Update NN to minimize error

`mlreflect 2.0` from a user's perspective

Measured curve



+

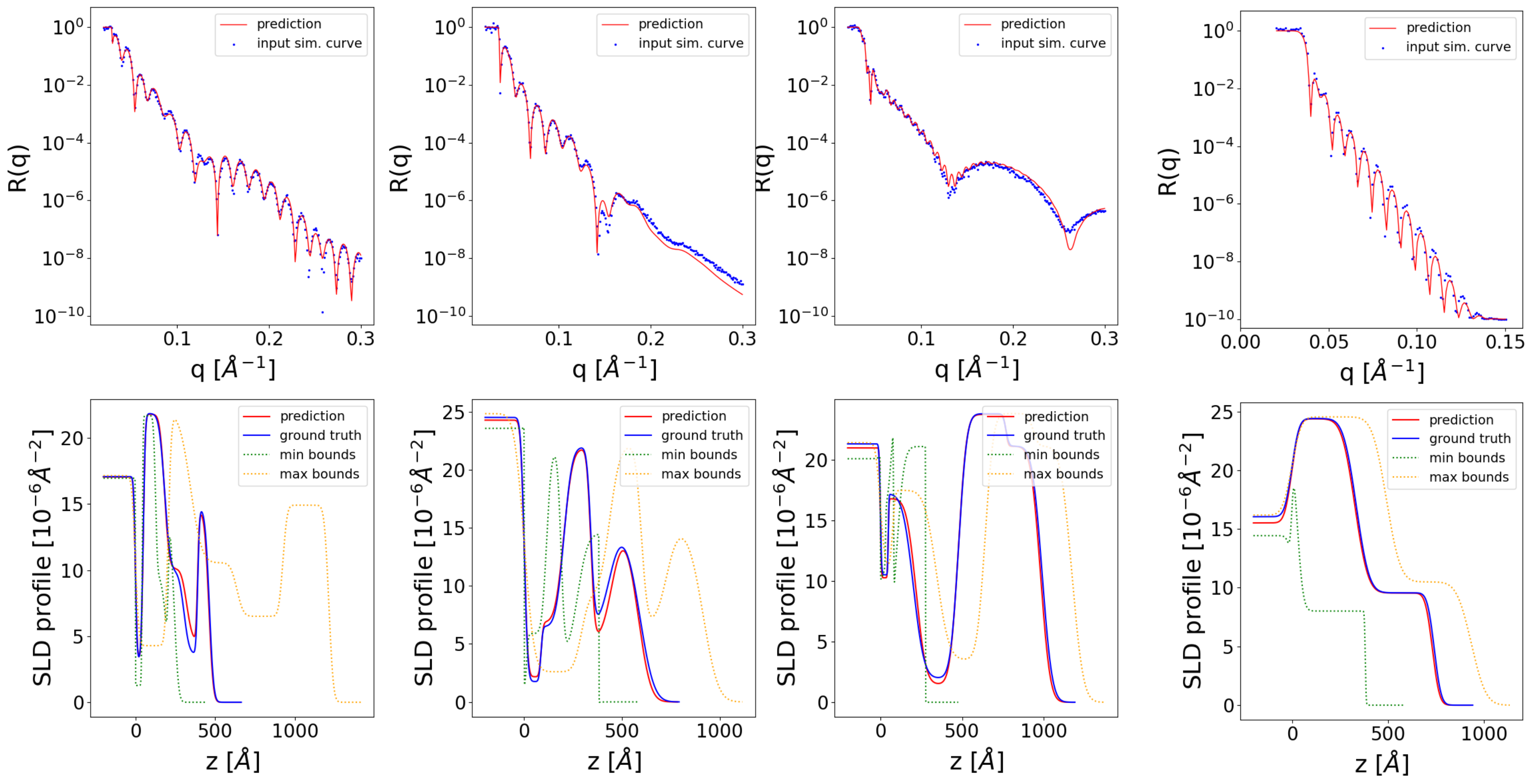
Parameter ranges - additional input

Parameter	Min value	Max value
d_1	10.	50.
ρ_1	15.1	15.3
σ_1	0.	40.
…	…	…

```
params = model.predict(input=(refl_curve, param_ranges))
```

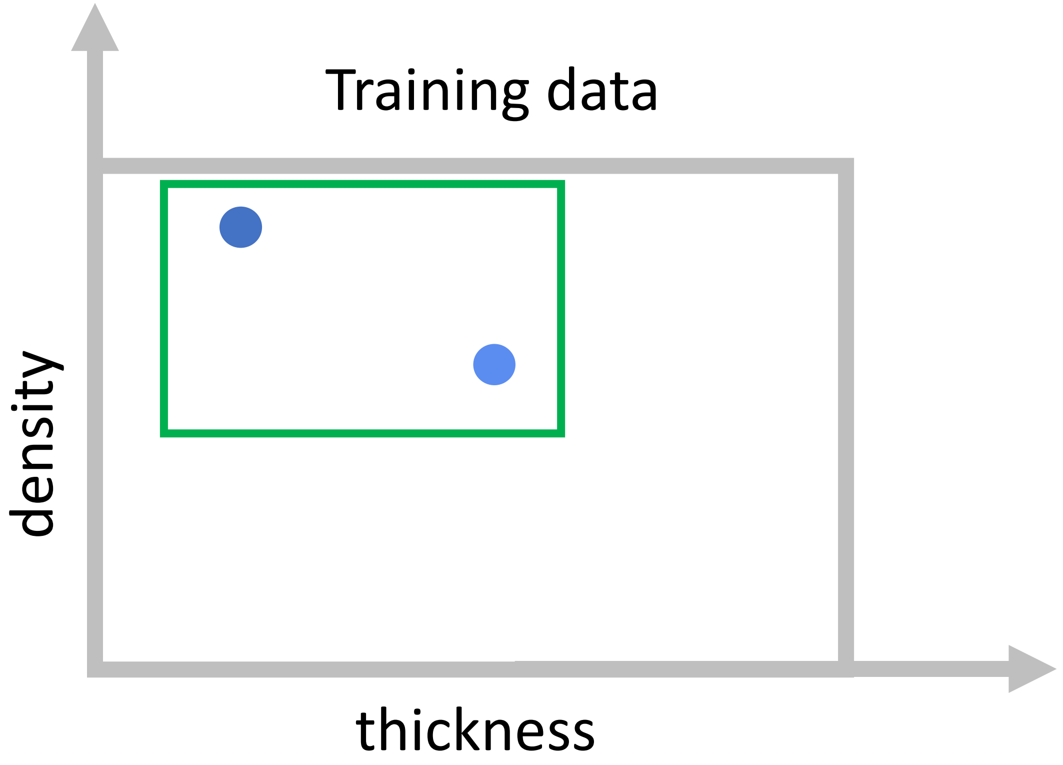
`mlreflect 2.0` - complex scenarios with multiple layers

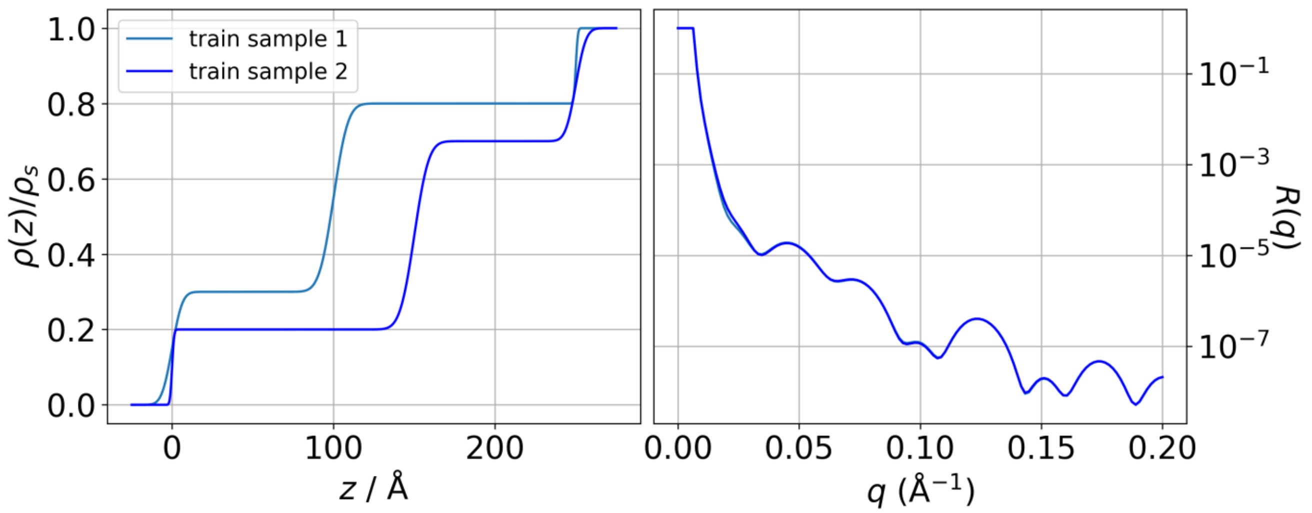
Single model, different sample-dependent parameter ranges



- Works with high number of parameters (tested up to 17)
- No postprocessing (can be improved further by gradient descent)
- Real-time (milliseconds)
- If the ranges are wide enough to include ambiguous results, both ML and conventional fit will fail.

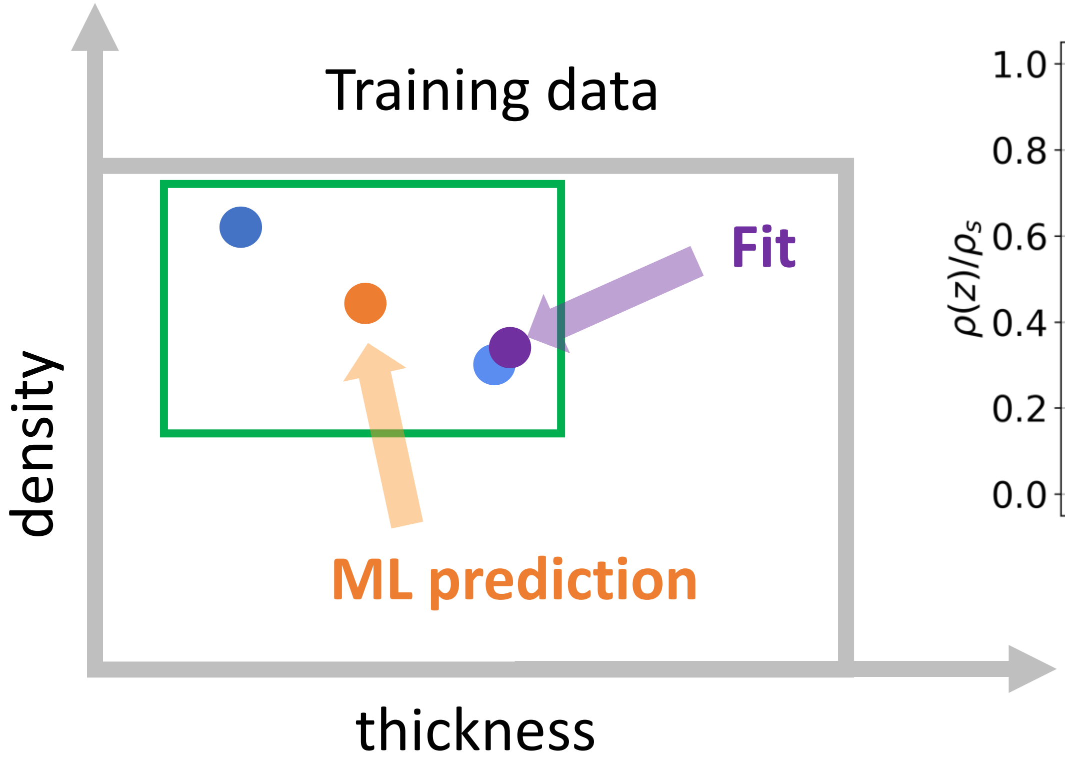
Both `mlreflect 2.0` and conventional fit fail in case of ambiguity

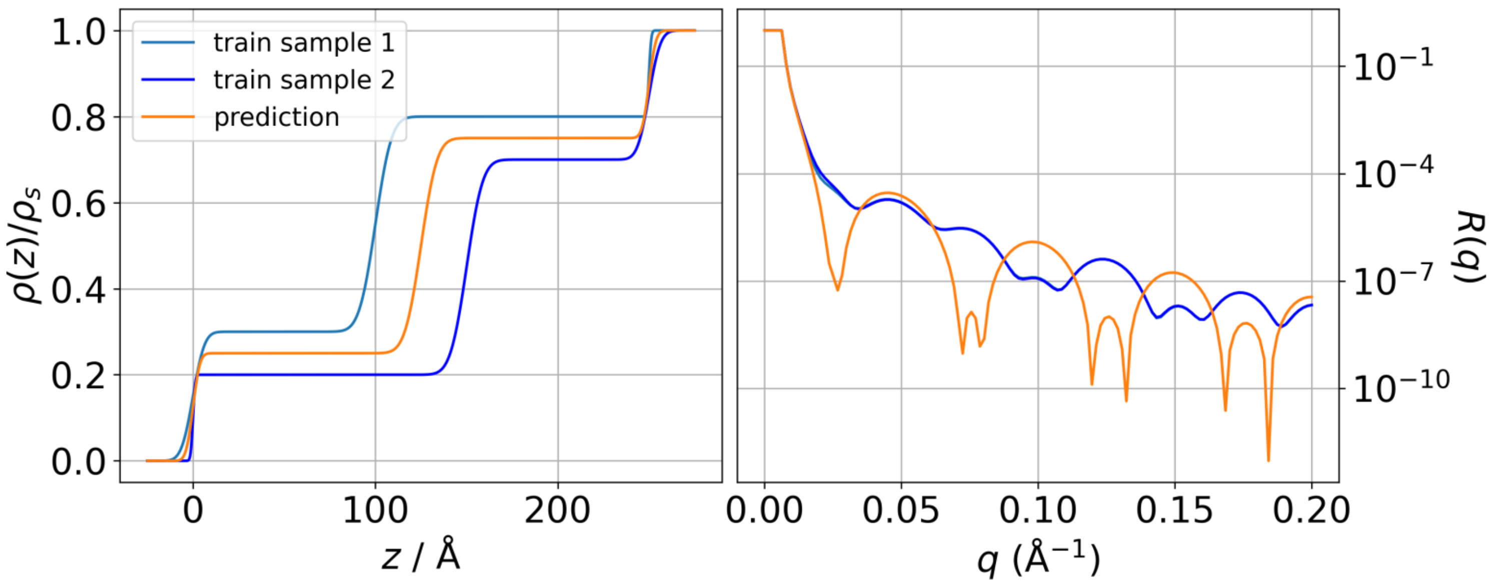




Both solutions might be physically correct, and these methods only provide one.

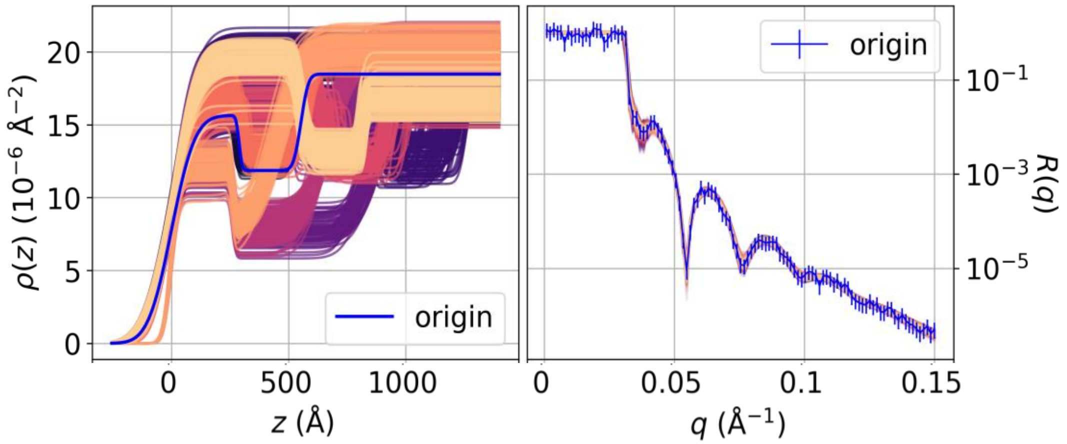
Both `mlreflect 2.0` and conventional fit fail in case of ambiguity





Both solutions might be physically correct, and these methods only provide one.

To get all the solutions, we need probabilistic methods (Bayesian analysis)

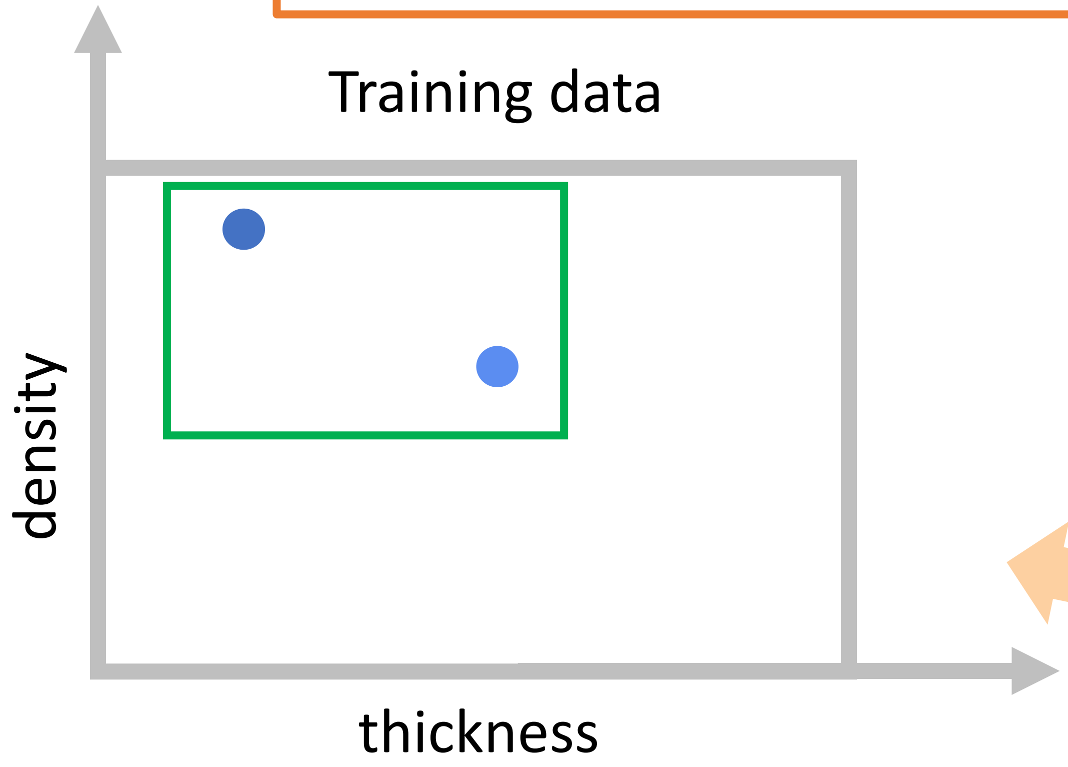


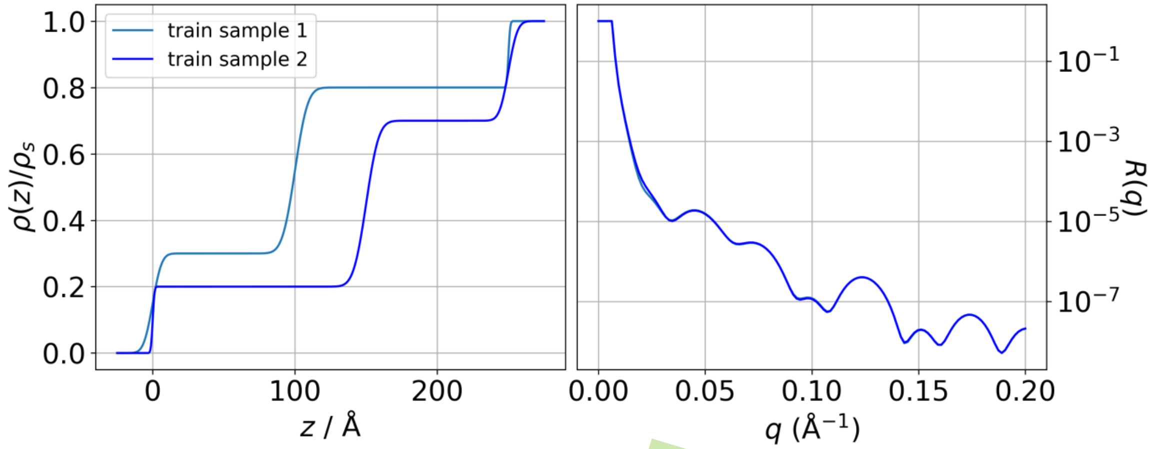
Probabilistic machine learning approach to reflectometry – `mlreflows (3.0)`

Fast, accurate, reliable Bayesian reflectometry analysis with normalizing flows

How mlreflows resolves ambiguity

Normalizing flows is trained to estimate the whole parameter distribution for every curve



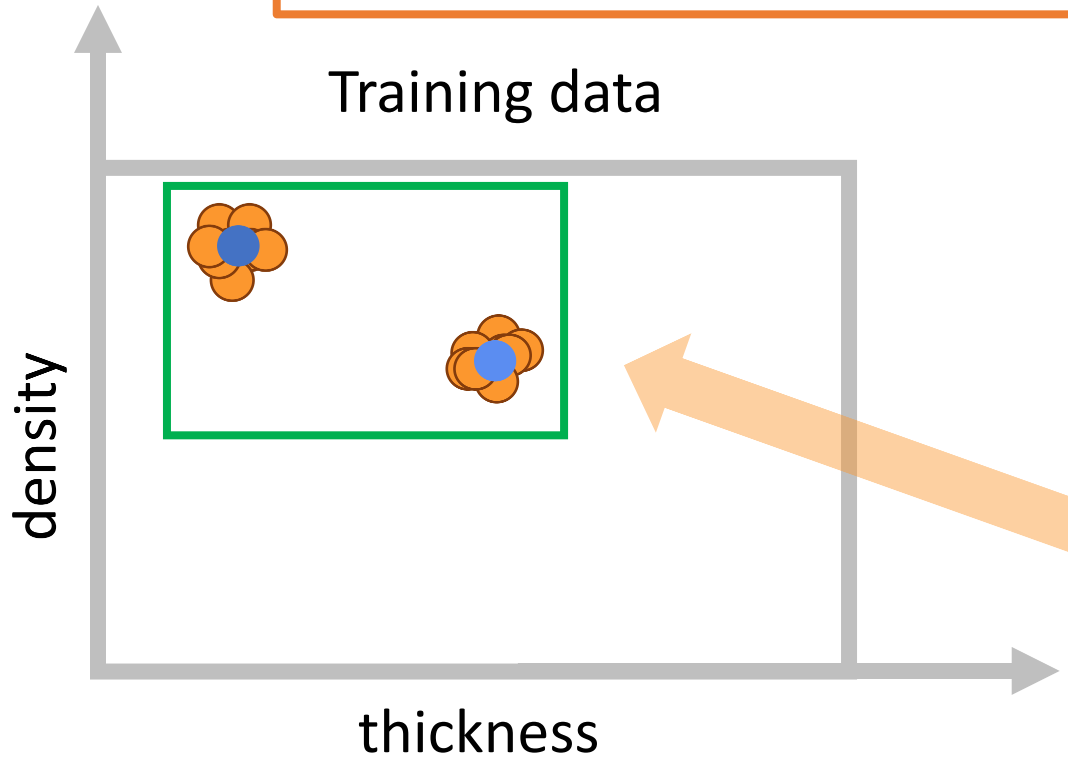


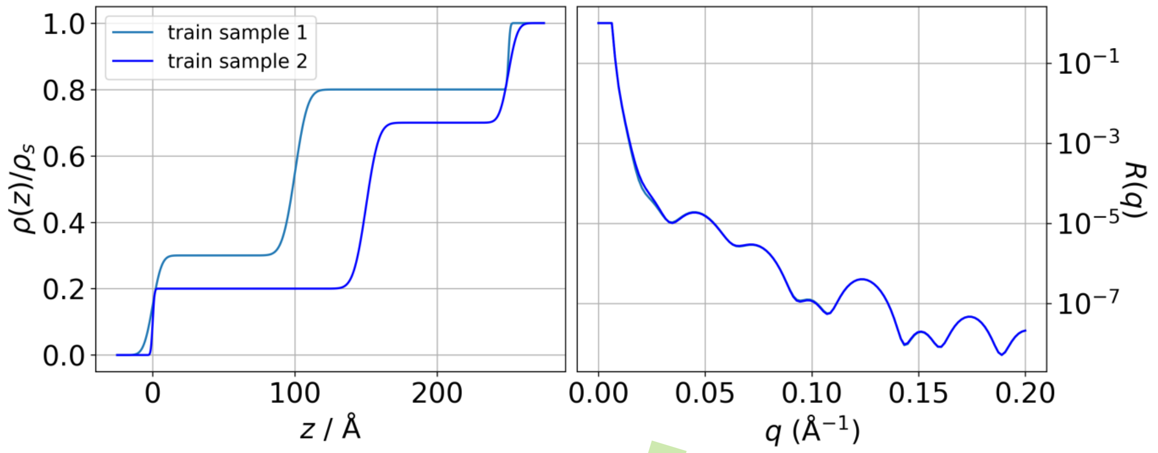
Input: curve + priors

Generate samples from estimated distribution

How `mlreflows` resolves ambiguity

Normalizing flows is trained to estimate the whole parameter distribution for every curve



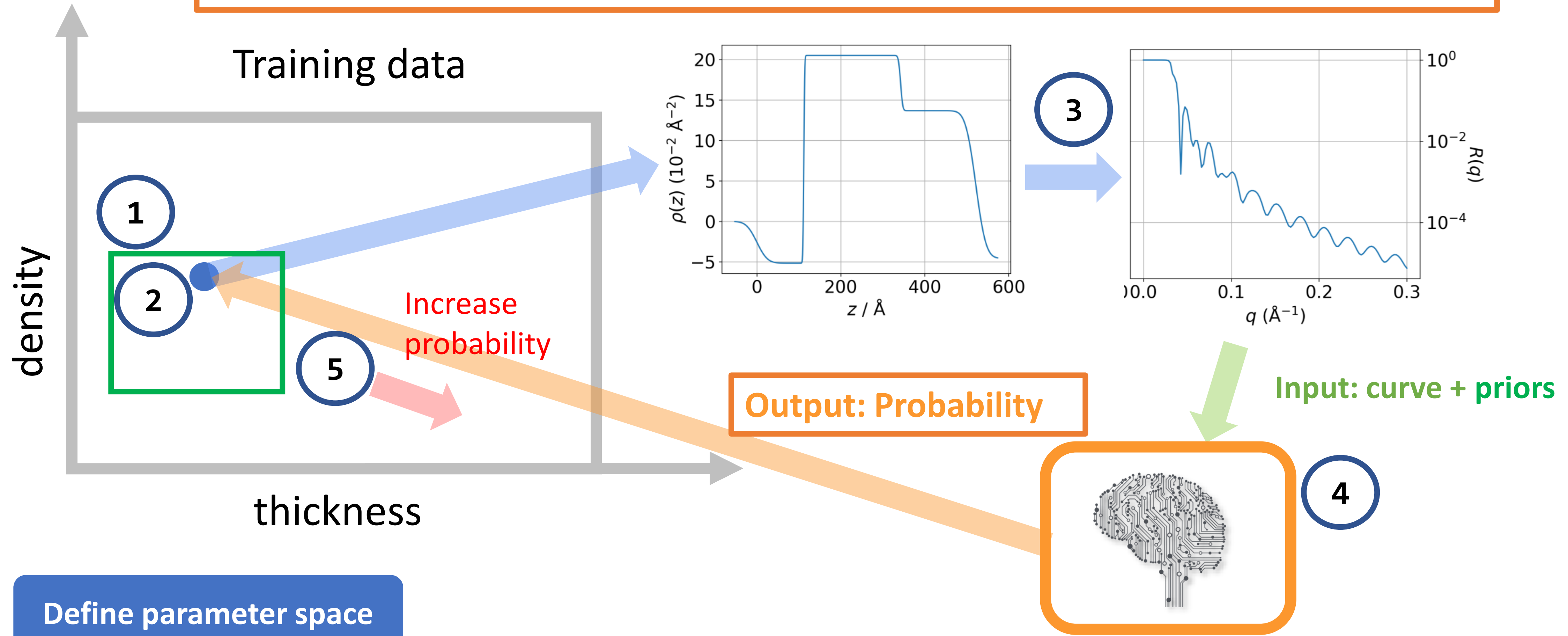


Input: curve + priors

Generate samples from estimated distribution

Training `mlreflows`

Normalizing flows is trained to estimate the whole parameter distribution for every curve



Define parameter space

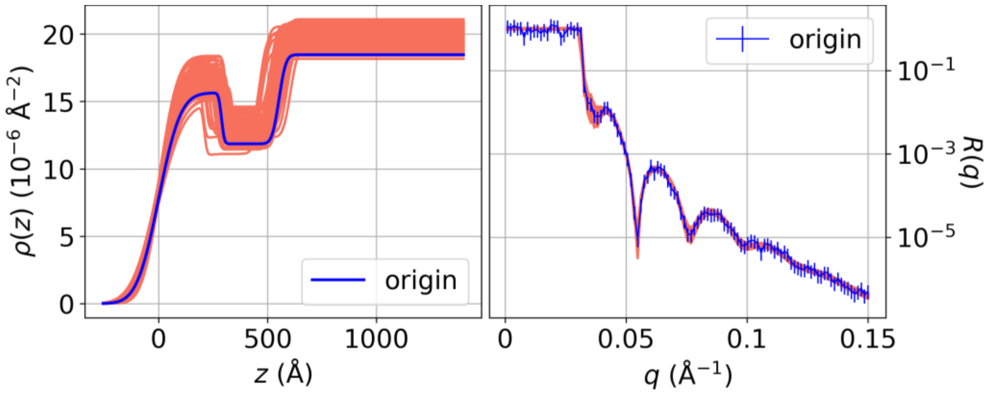
1. Generate random priors → **2. Generate params** → **3. Simulate curve** → **4. Calculate probability** → **5. Update NN to increase probability**

repeat

`mlreflows` from a user's perspective

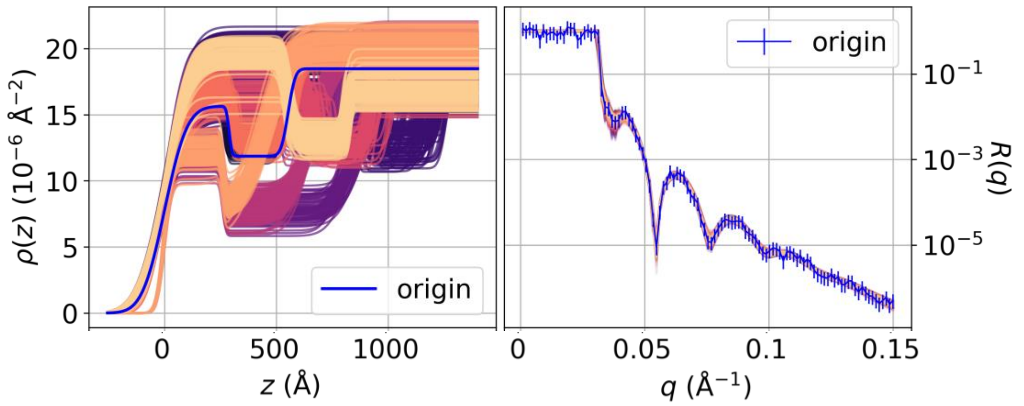
```
model.sample(num_samples, input=(refl_curve, narrow_param_ranges))
```

Narrow range



```
model.sample(num_samples, input=(refl_curve, wide_param_ranges))
```

Wide range

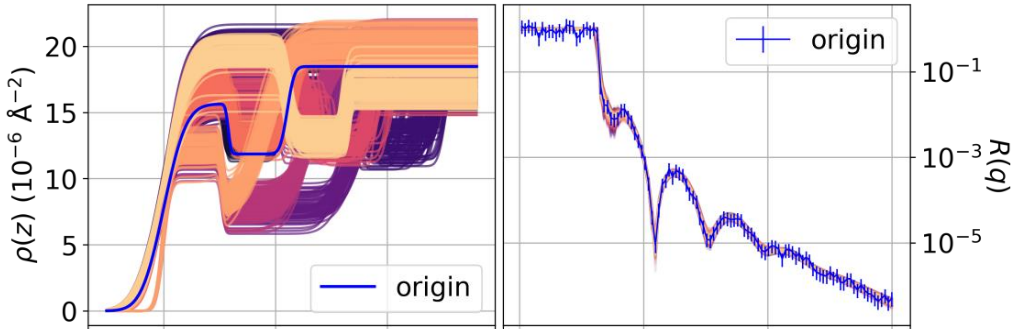


Unlike **mlreflect 2.0**, it does not fail in the case of ambiguity. It provides all the solutions!

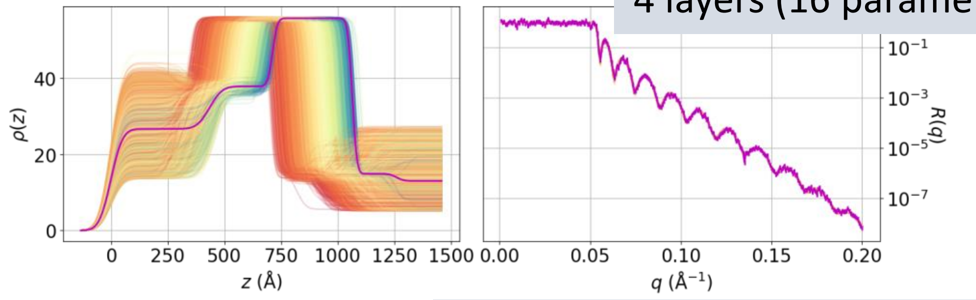
mlreflows – complex scenarios

Simulated data

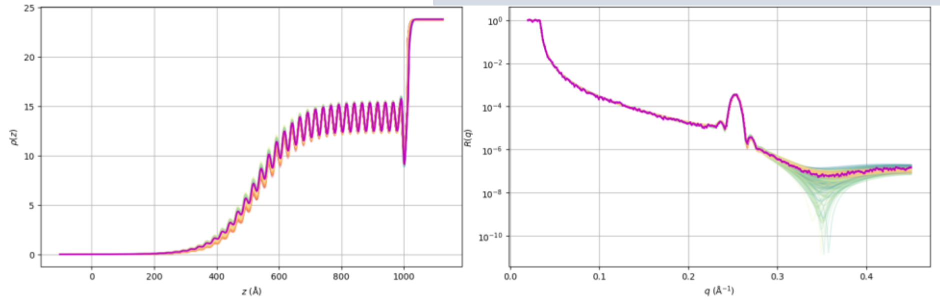
2 layers (10 parameters)



4 layers (16 parameters)

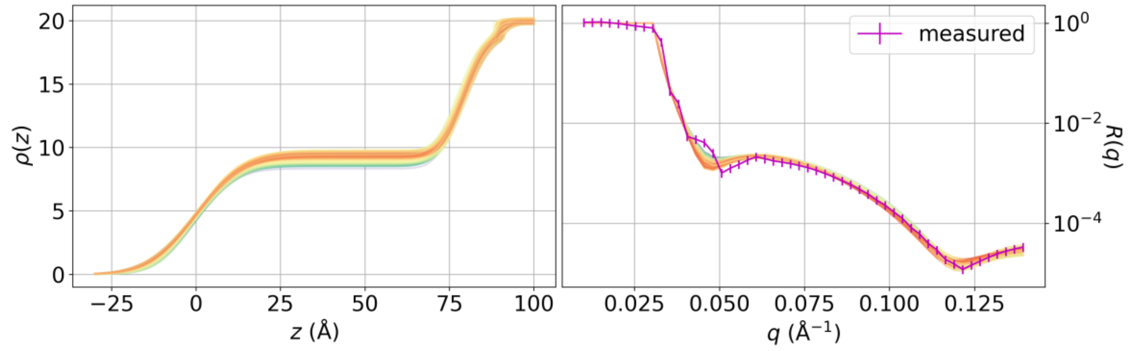


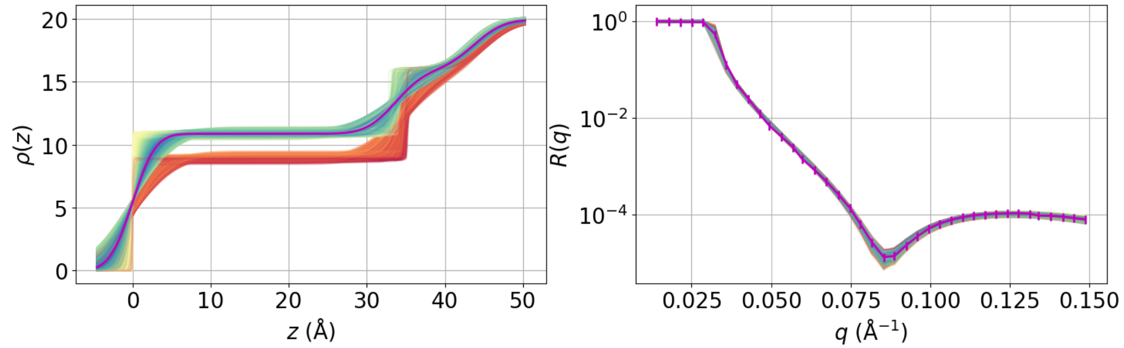
Periodic structure (19 parameters)

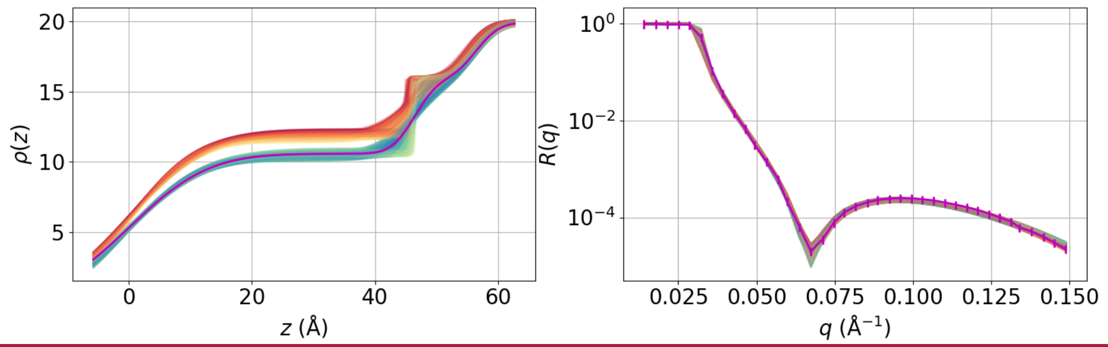


Experimental XRR data

2 layers (10 parameters)

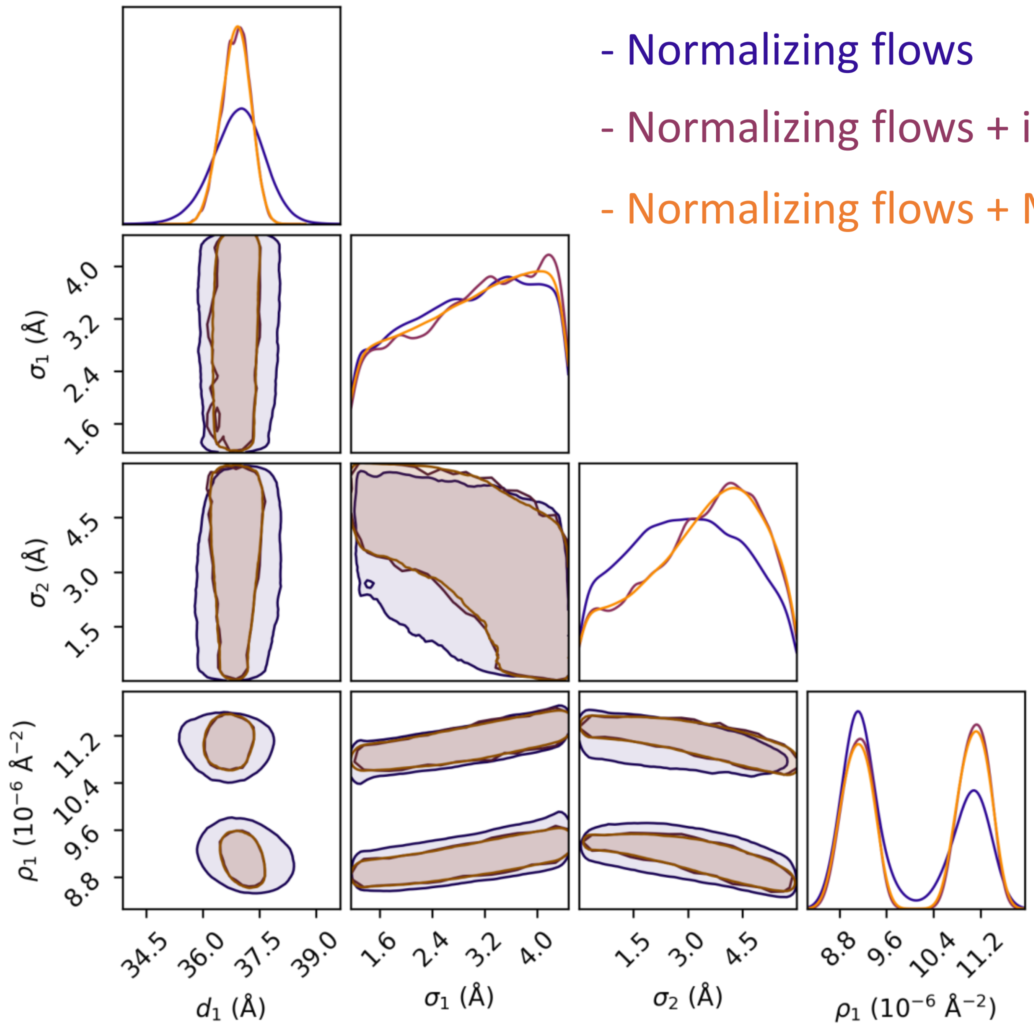






Normalizing flows coupled with MCMC / IS

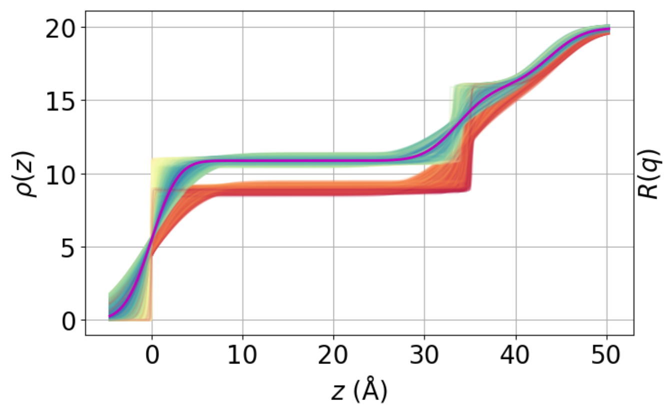
Corner plot with 4 parameters out of 10

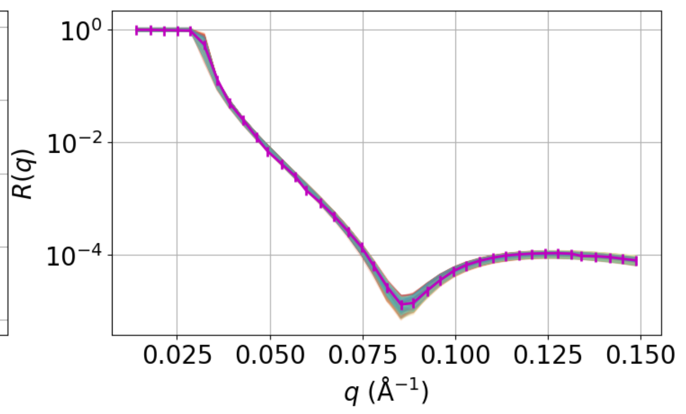


- Normalizing flows
- Normalizing flows + importance sampling
- Normalizing flows + MCMC

Can be refined to provide **accurate** distribution in mere seconds

Can it miss any modes?

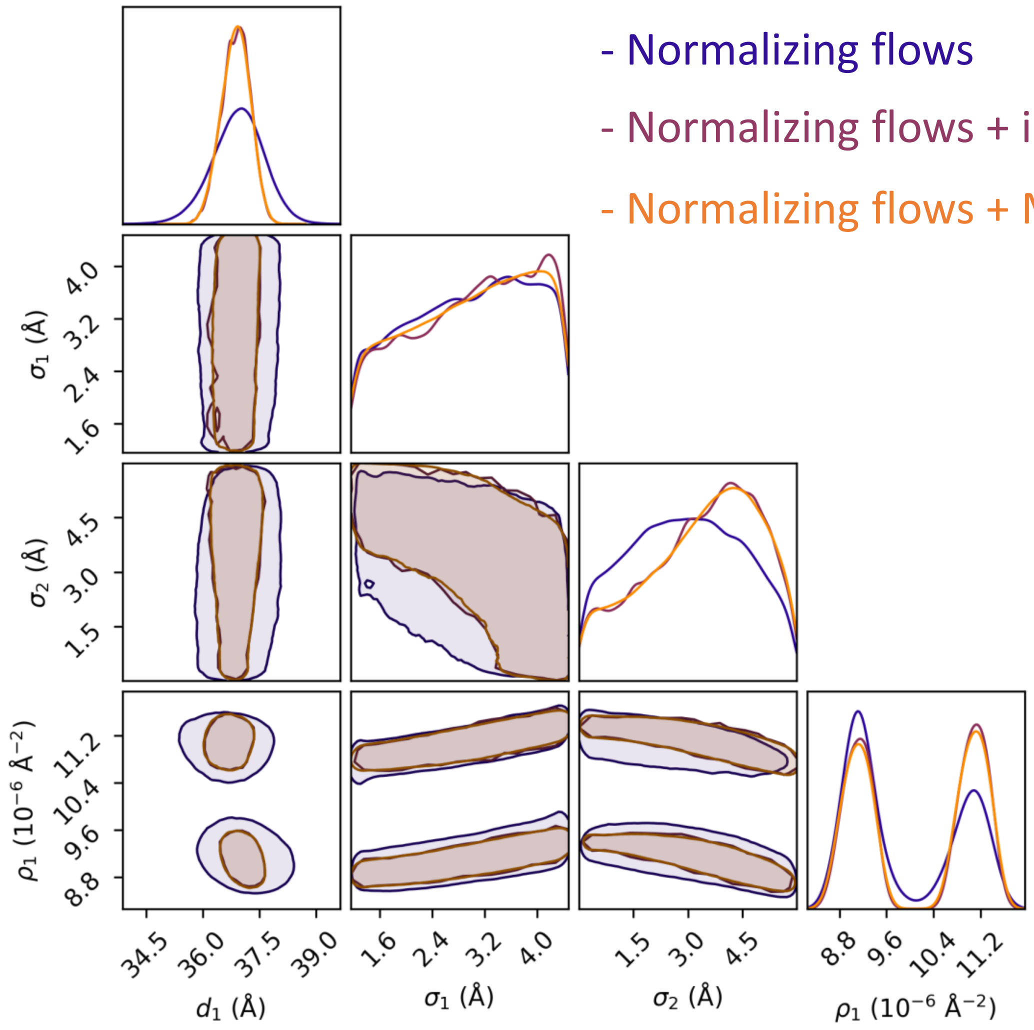




Starostin et. al (under preparation)

Normalizing flows **guarantees** revealing all modes [1]

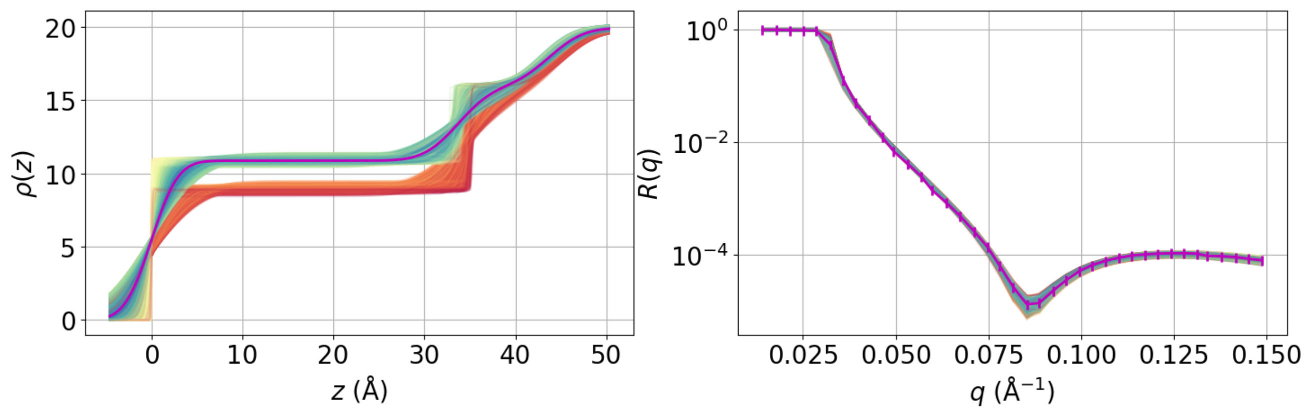
Corner plot with 4 parameters out of 10



Can be refined to provide **accurate** distribution in mere seconds

Can it miss any modes?

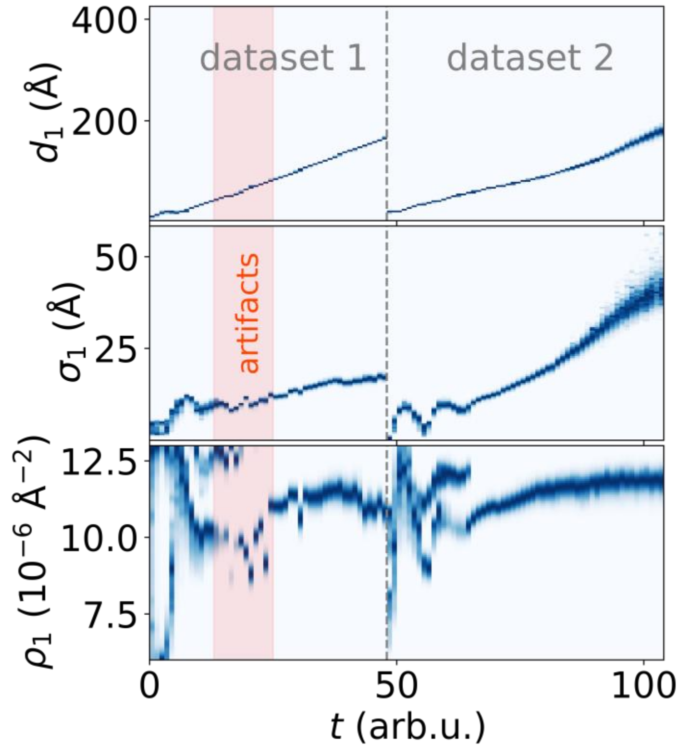
It can be **wider** than the correct solution (low sample efficiency), but **not narrower**



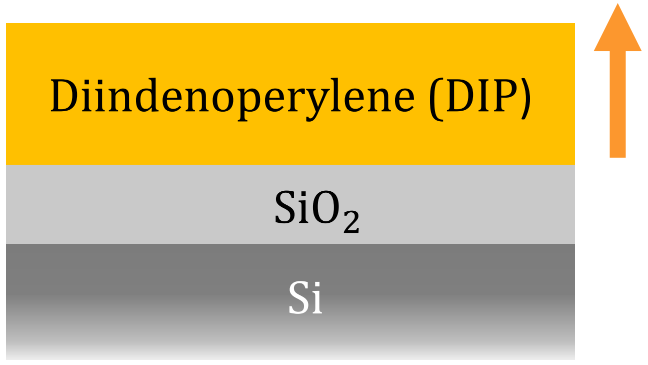
[1] Dax, M., et al. *Physical Review Letters* 130.17 (2023): 171403.

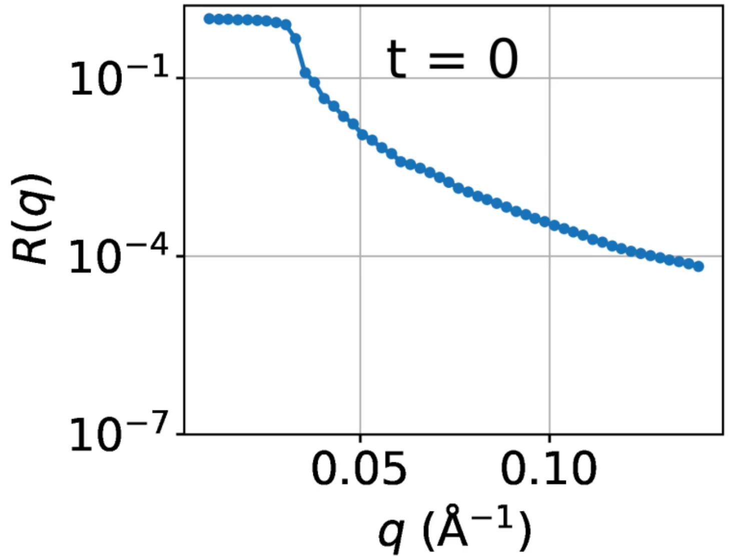
Analysis of experimental *in situ* XRR data: `mlreflows` vs conventional Bayesian methods

🟦 `mlreflows` + Importance Sampling



In situ data (**DIP growth**)





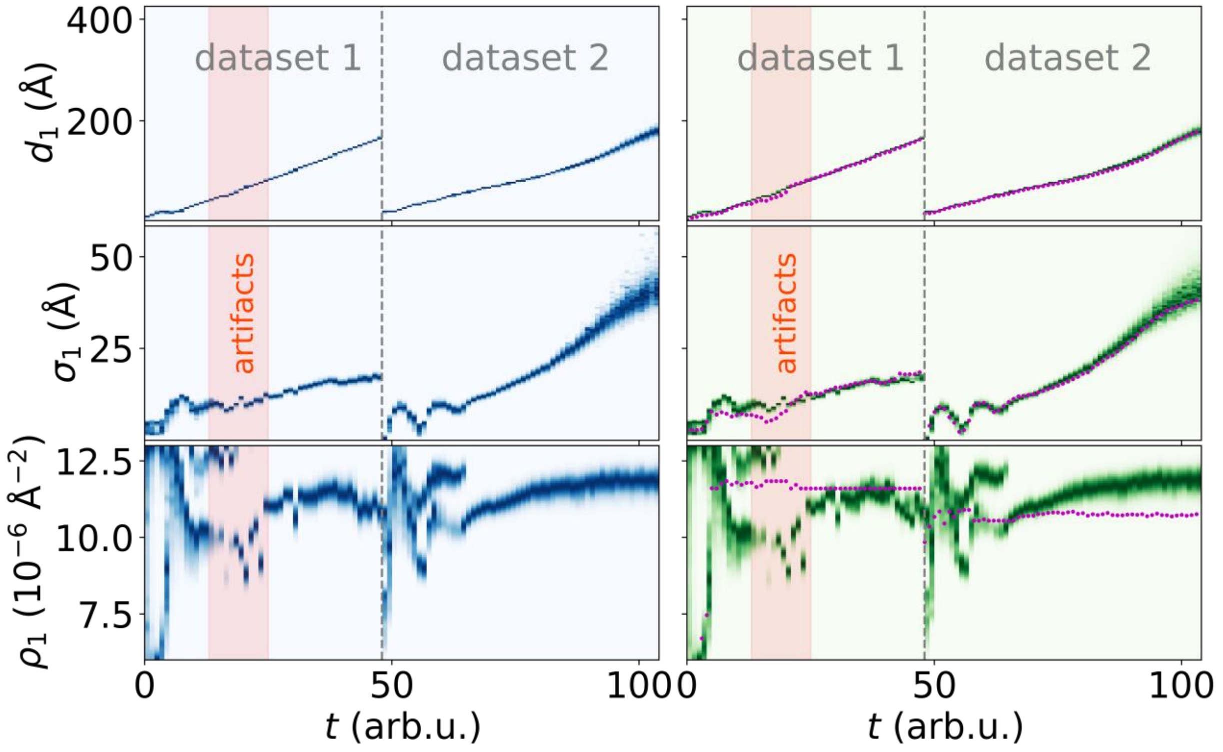
3 open parameters: thickness, roughness, SLD of organic layer

Analysis of experimental *in situ* XRR data: `mlreflows` vs conventional Bayesian methods

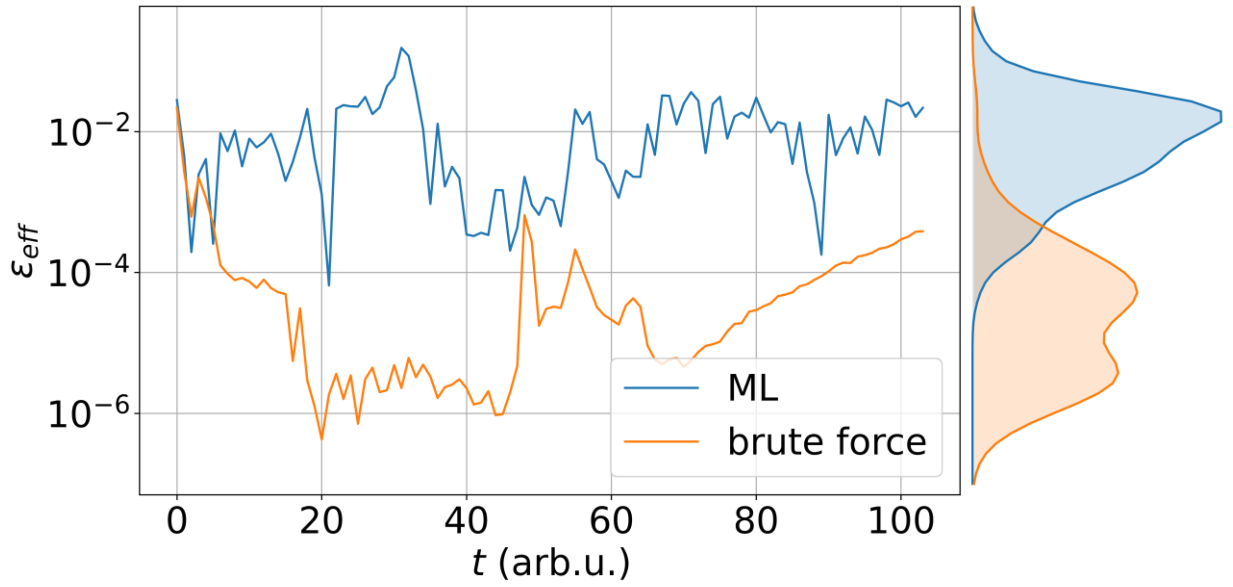
🟦 `mlreflows` + Importance Sampling

🟩 Monte Carlo Importance Sampling (reliable conventional method)

● Manual fit via differential evolution

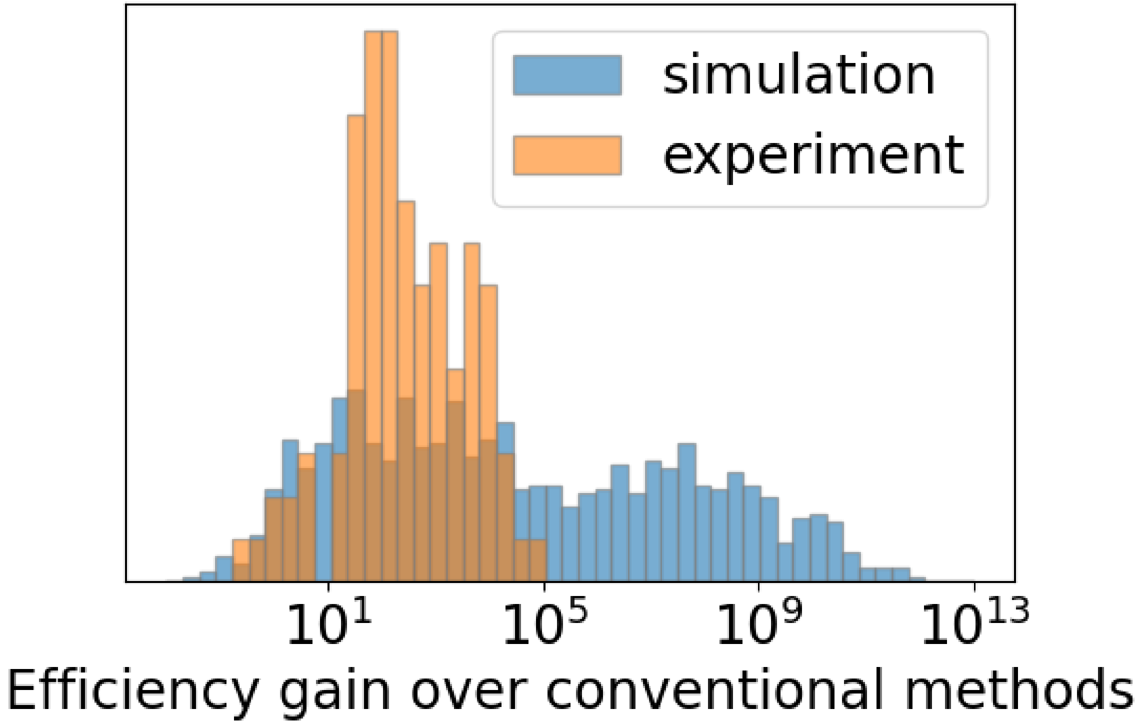


Sample efficiency (reflects required number of reflectivity calculations)



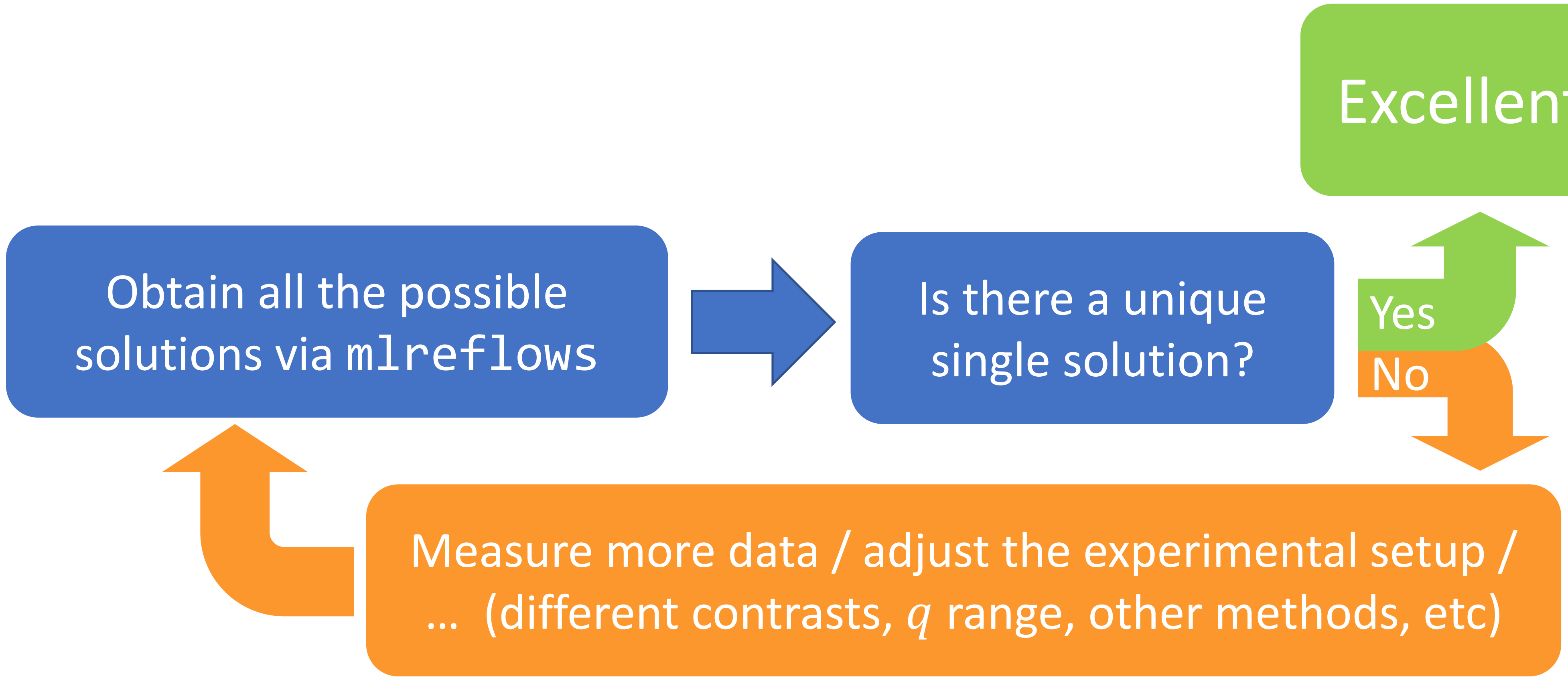
3 open parameters: thickness, roughness, SLD of organic layer

Efficiency gain over conventional methods



More open parameters / wider parameter ranges →

Reliable approach to reflectometry analysis


Excellent!
Obtain all the possible solutions via `mlreflows`
Is there a unique single solution?
Yes
No
Measure more data / adjust the experimental setup / … (different contrasts, q range, other methods, etc)

Summary

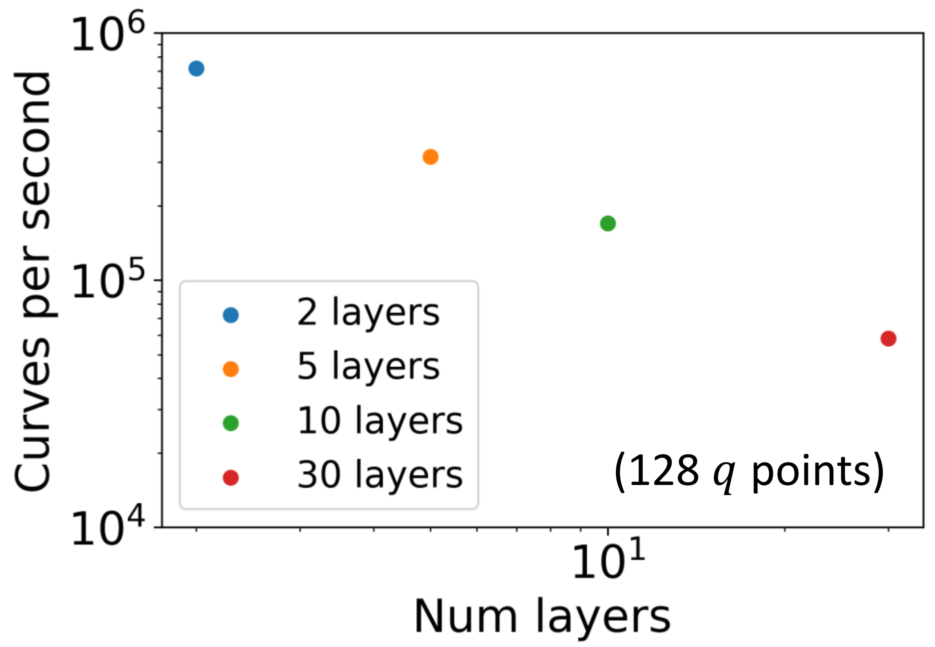
- Ambiguity is one of the main challenges in the reflectometry analysis both for ML and conventional methods
- We present two novel prior-aware ML methods for reflectometry analysis:

  1. **`mlreflect 2.0`** – a lightweight ML approach – fast alternative to conventional fit that can even be trained during the beamtime and works in complex scenarios if priors are narrow enough.
  2. **`mlreflows (3.0)`** – a probabilistic solution based on normalizing flows. Coupled with importance sampling / MCMC, it enables **reliable and accurate** real-time Bayesian analysis, achieving up to 10^{12} efficiency gain over conventional methods.

- Additionally, we present PyTorch implementation of Abeles & MCMC that speeds up ML training, ML inference, and conventional analysis on GPU.

Supporting Information

How to train NN during a beamtime? GPU implementation of Abeles



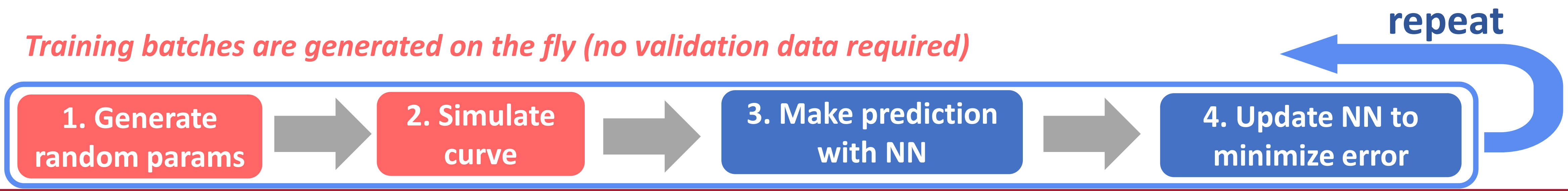
Reflectometry simulations (Abeles) in PyTorch:

≈ **100k – 1M curves per second** (Nvidia GeForce RTX 2080 Ti)

Markov Chain Monte Carlo (MCMC) in PyTorch:

Approx. × 600 acceleration over CPU (30 min → 3 sec)

Training batches are generated on the fly (no validation data required)


repeat
1. Generate random params
2. Simulate curve
3. Make prediction with NN
4. Update NN to minimize error

emcee package [2] in PyTorch – MCMC on GPU

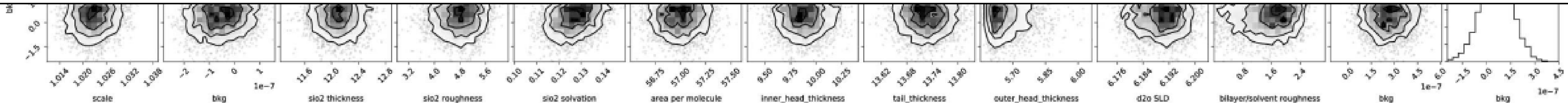


Figure 4
A corner plot for the varying parameters of DMPC bilayers supported on a silicon crystal, measured at three contrasts. The sampling took ~33 min on a 2.8 GHz quad-core computer for 20 saved steps, corresponding to 4000 samples, with the steps being thinned by a factor of 400. A larger-scale image is available in the supporting information. Source [1]

GPU implementation allows to reduce 30 minutes to ≈ 3 seconds ($\times\ 600$ acceleration)

Real-time Bayesian analysis, we just need a good initialization!

Modern GPU (Nvidia GeForce RTX 2080 Ti) + vectorized PyTorch implementation heavily relied on *refnx* [1]:

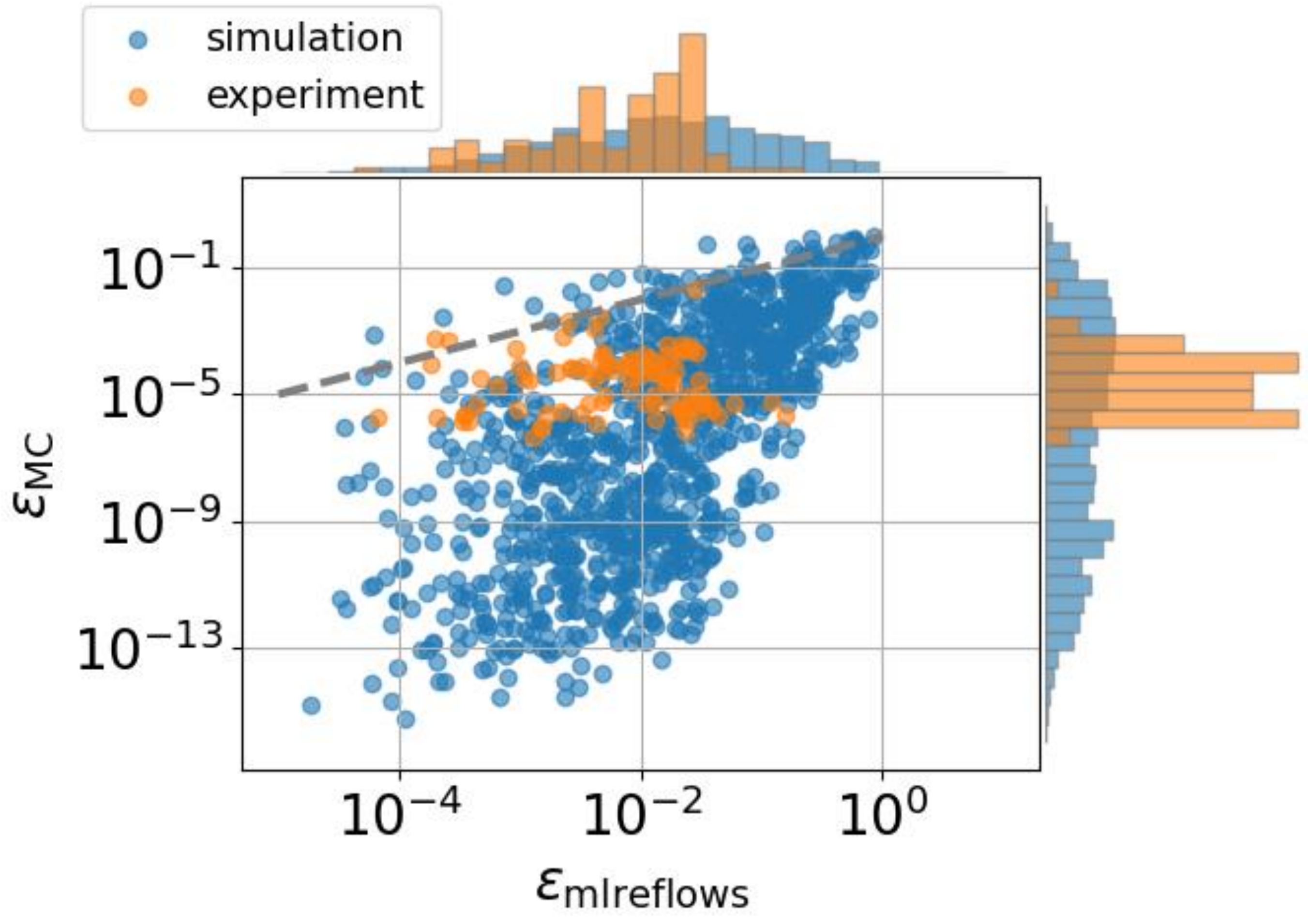
$\approx$ 100k – 1M curves per second

Probably can be pushed further by compiled implementations (JAX, Julia, etc)

[1] Nelson, A. R. J. & Prescott, S. W. (2019). J. Appl. Cryst. 52, 193-200

[2] Foreman-Mackey, Daniel, et al. (2013) Publ. Astron. Soc. Pac.125, 306–312

Efficiency gain over conventional methods

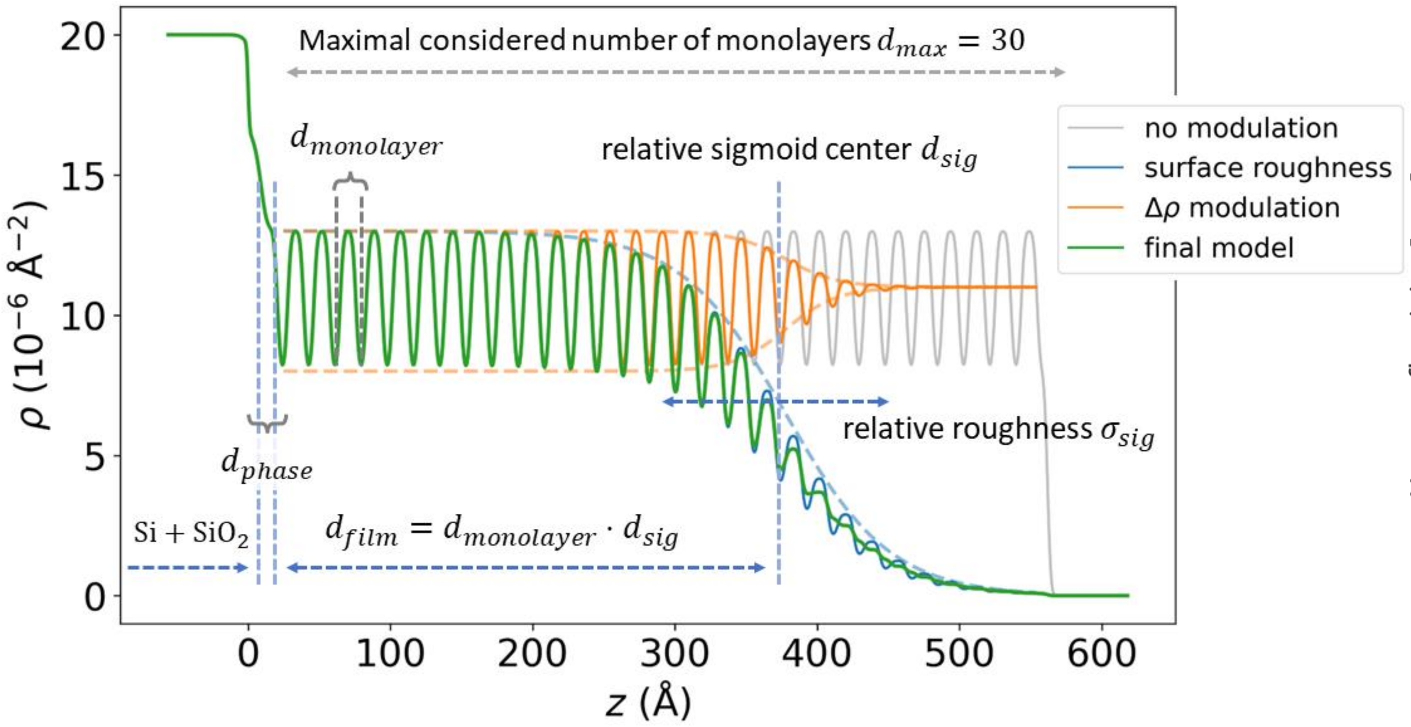


Time estimate for different sample efficiency ϵ
(1000 effective samples and **1M curves / sec** speed)

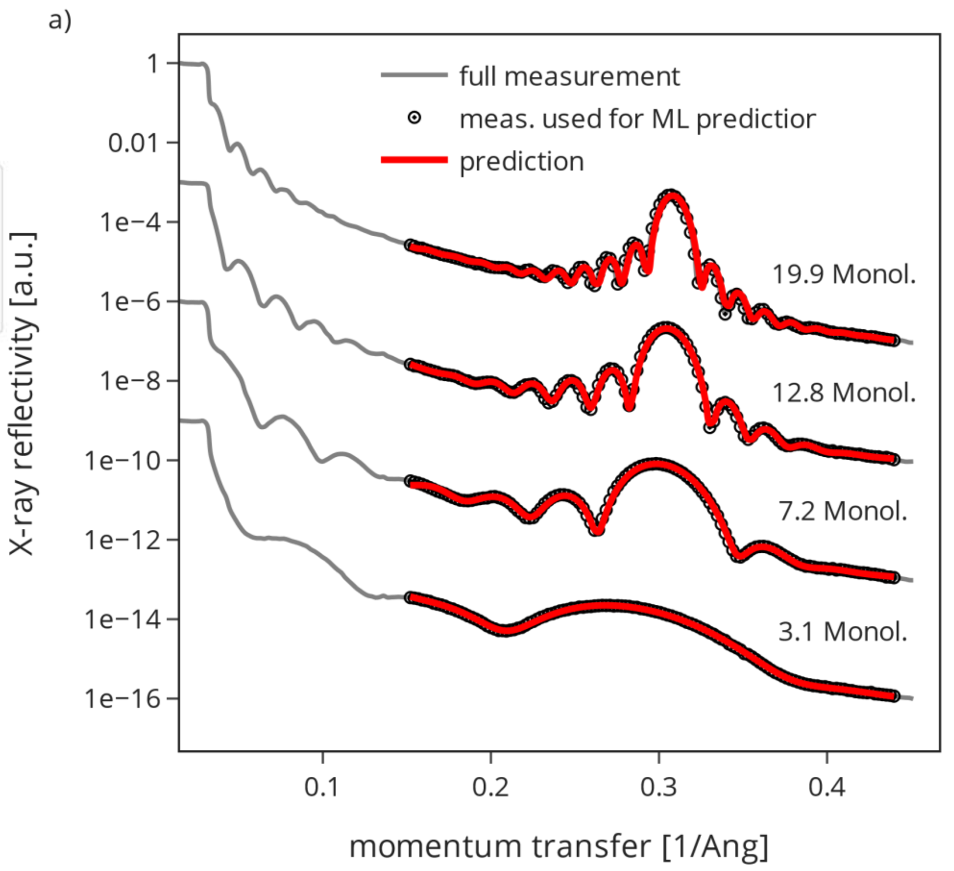
ϵ	**Num samples**	**Time / 1 eff. sample**	**Total time**
$\epsilon = 10^{-2}$	10^{5}	0.1 ms	100 ms
$\epsilon = 10^{-4}$	10^{7}	10 ms	10 sec
$\epsilon = 10^{-6}$	10^{9}	1 sec	17 min
$\epsilon = 10^{-8}$	10^{11}	1.5 min	27 hours
$\epsilon = 10^{-10}$	10^{13}	2.5 hours	120 days
$\epsilon = 10^{-12}$	10^{15}	11 days	30 years

`mlreflect 2.0` – fitting Bragg peaks with multilayer parameterization

Parameterization scheme with 17 parameters of periodic monolayer structure



Real-time AI-based fit on in situ experimental data

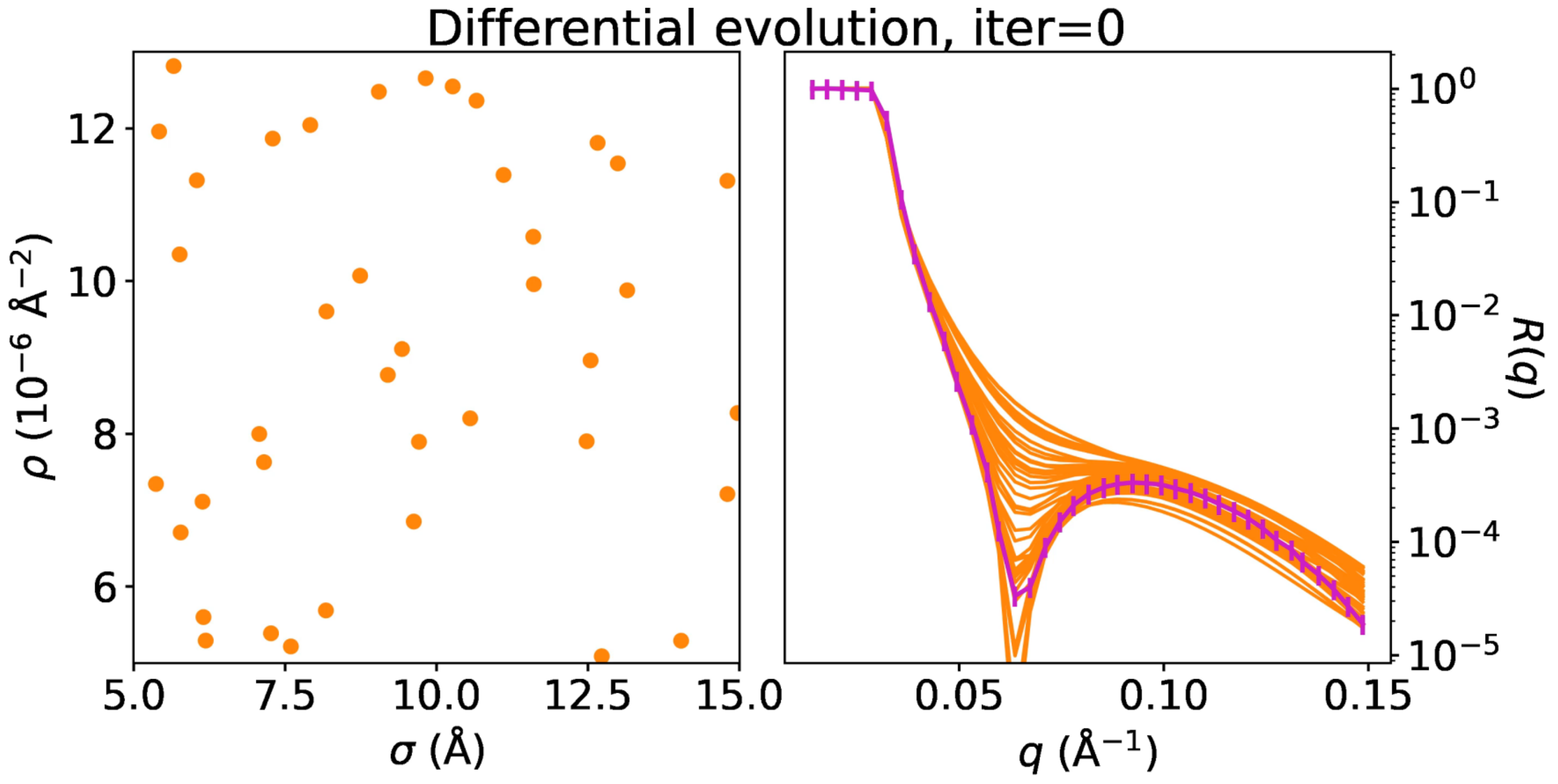


Conventional methods for Bayesian reflectometry analysis

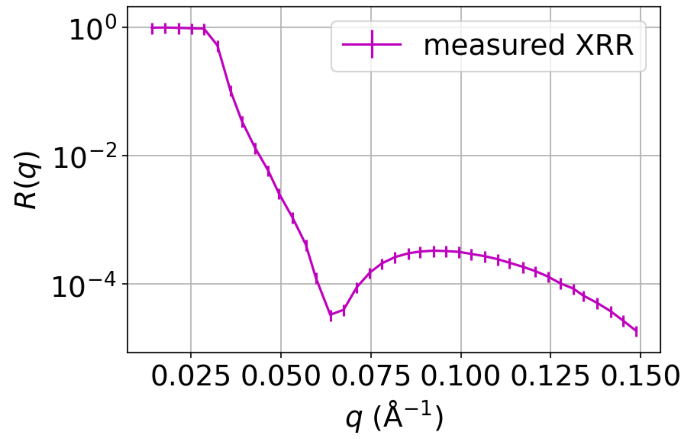
Sample efficiency and the curse of dimensionality

Conventional algorithms (example with *refnx* [1])

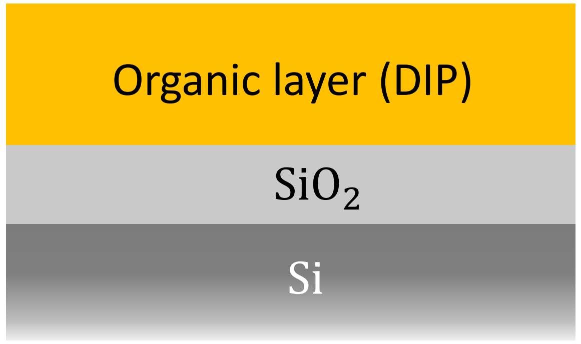
Fit via differential evolution (DE) → **Run MCMC around the fit**



Experimental XRR data:
DIP on silicon substrate



DIP density ρ and roughness σ unknown



[1] Nelson, A. R. J. & Prescott, S. W. *refnx*: neutron and X-ray reflectometry analysis in Python. (2019). J. Appl. Cryst. 52, 193-200

Conventional methods for Bayesian reflectometry analysis

1 **Find the solution(s)**

MCMC does not help – on the opposite, it requires proper initialization

Gradient-based methods are not applicable - they stuck in local minima

Random exploration

- Differential evolution
- Monte Carlo Importance Sampling

How long does it take to get reliable result?

2 **Refine the distribution**

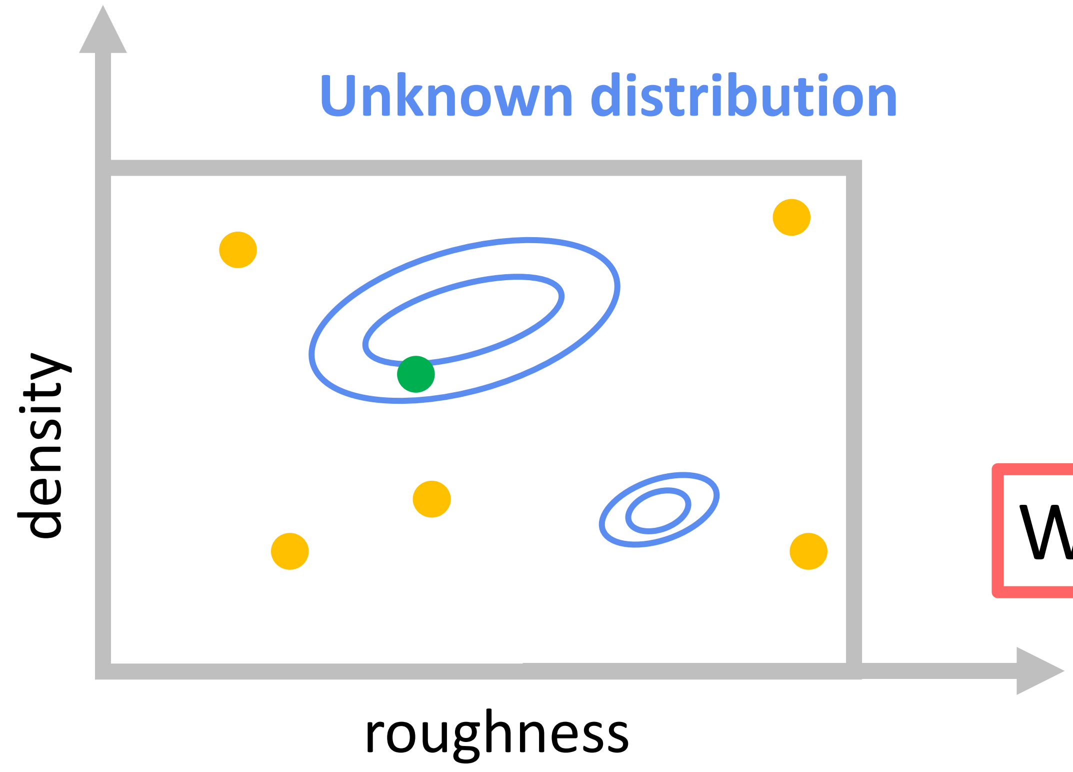
Affine Invariant Markov Chain Monte Carlo

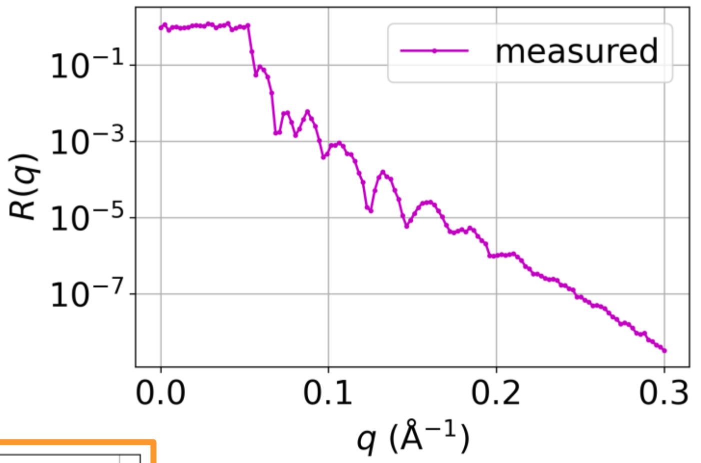
Gradient-based methods (Hamiltonian Monte Carlo, etc.)

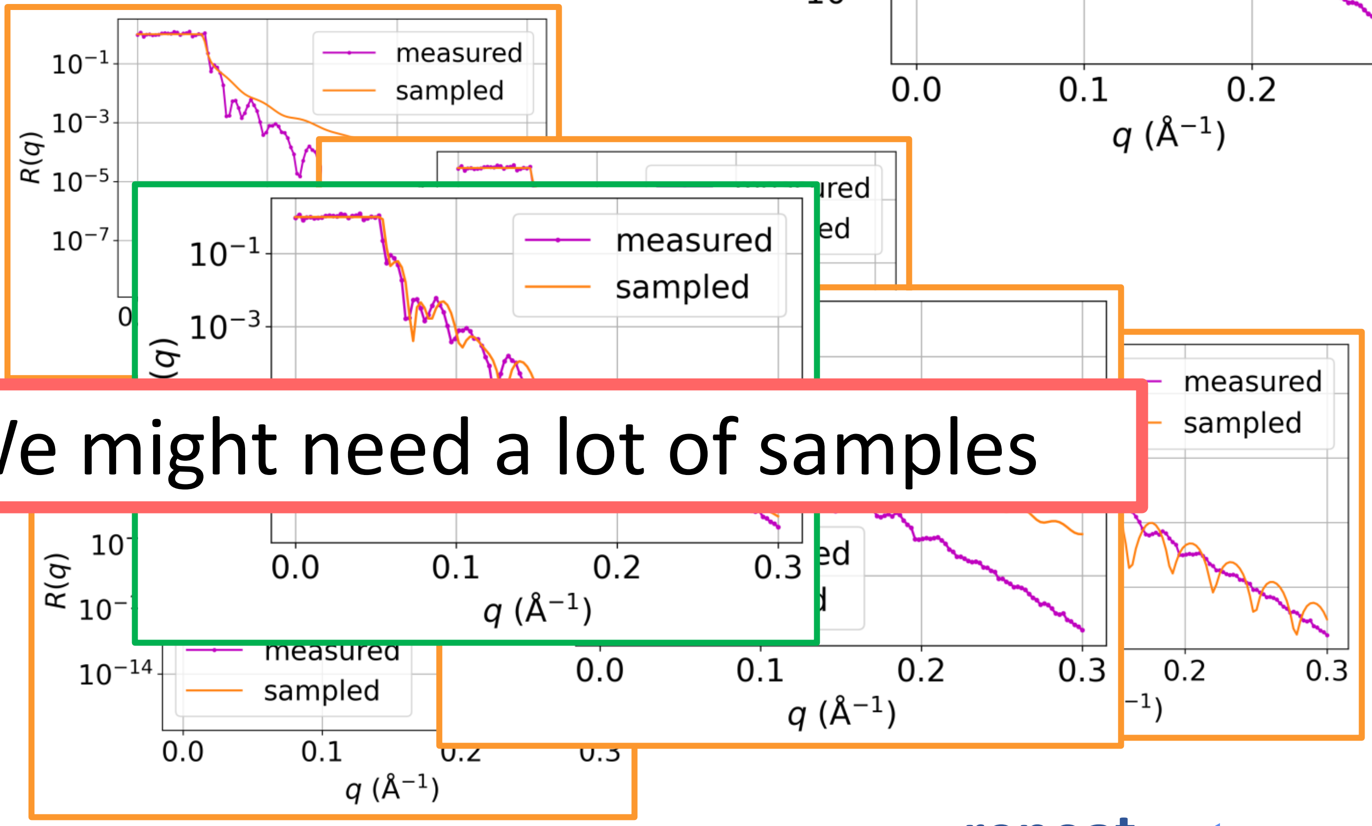
… many other methods

Takes seconds / minutes on GPU

Monte Carlo Importance Sampling (random exploration)







We might need a lot of samples

repeat

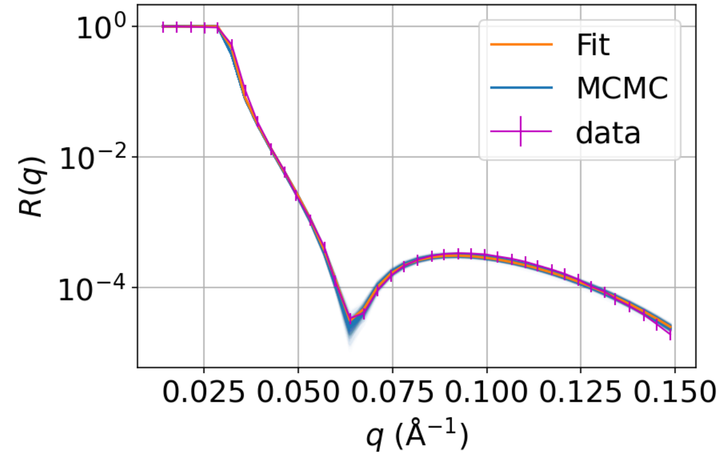
1. Generate random params → 2. Simulate curve → 3. Calculate likelihood (unnormalized posterior) → 4. Assign a sample with importance weight

Conventional algorithms (example with *refnx* [1])

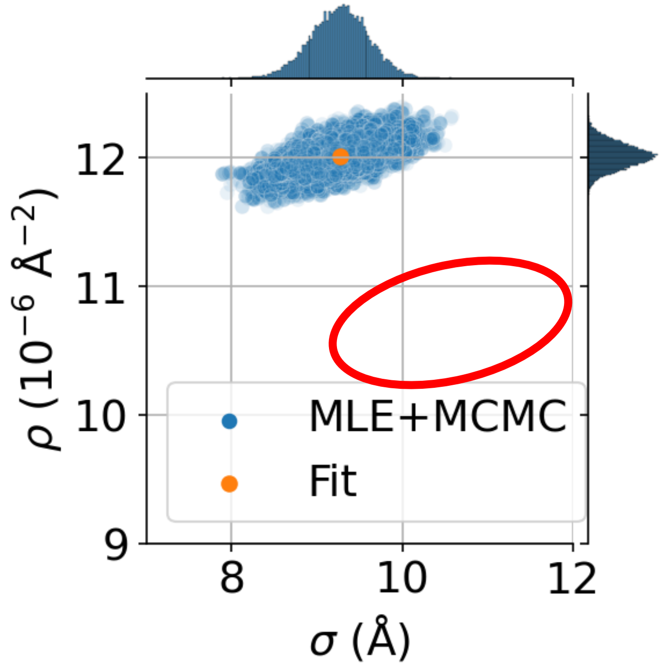
Fit via differential evolution (DE) → **Run MCMC around the fit**

MCMC can miss secondary modes depending on initialization [2]

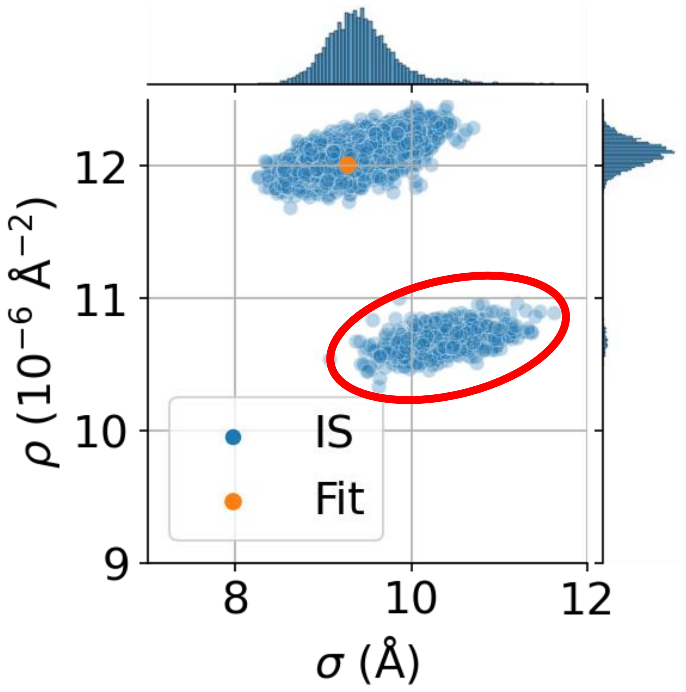
Experimental XRR data:
DIP on silicon substrate



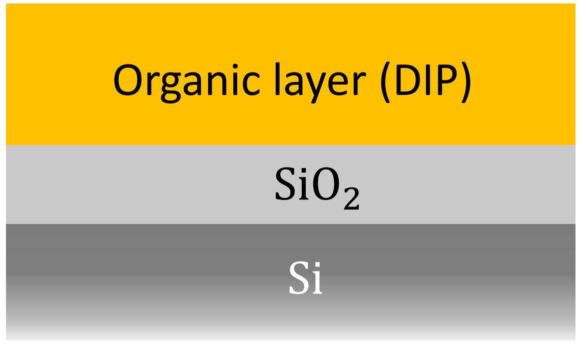
MCMC solution:



Monte Carlo solution:



DIP density ρ and roughness σ unknown



[2] Robert, C. & Casella, G. Monte Carlo Statistical Methods (2013) Springer Science & Business Media

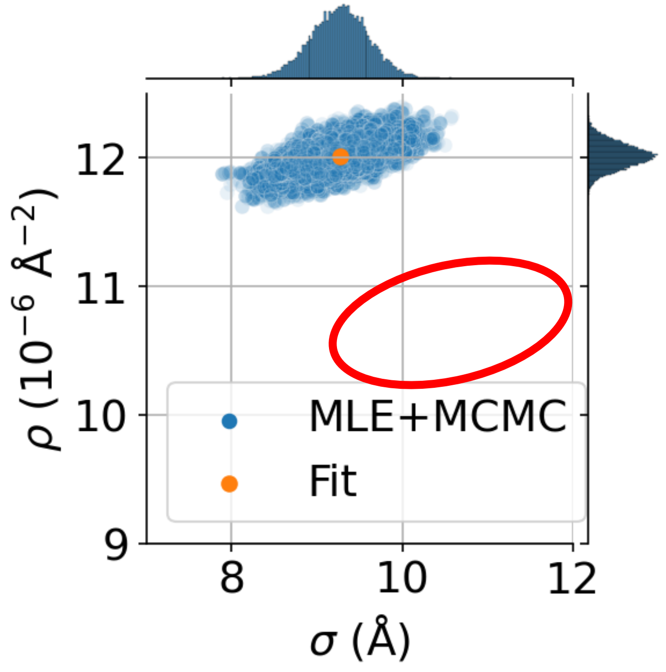
[1] Nelson, A. R. J. & Prescott, S. W. *refnx*: neutron and X-ray reflectometry analysis in Python. (2019). J. Appl. Cryst. 52, 193-200

Conventional algorithms (example with *refnx* [1])

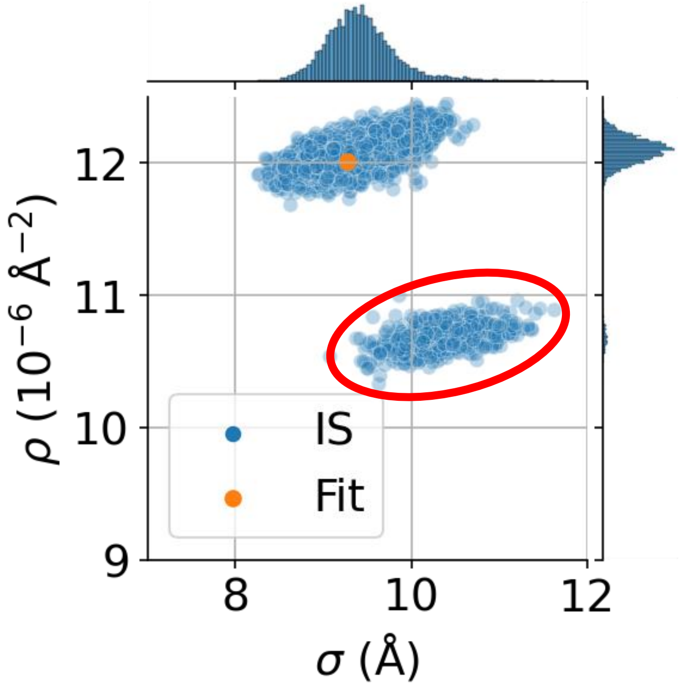
Fit via differential evolution (DE) → **Run MCMC around the fit**

MCMC can miss secondary modes depending on initialization [2]

MCMC solution:



Monte Carlo solution:



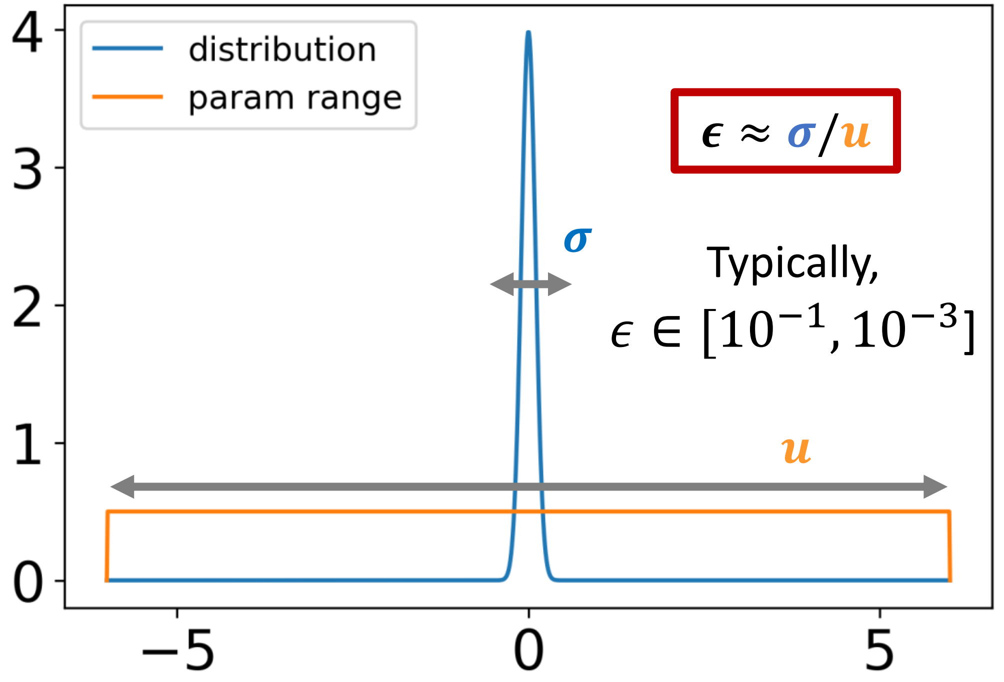
- Differential evolution is not reliable, does not address ambiguity.
- MCMC cannot fix it.
- However, there are conventional method that resolve all the modes.
- Random exploration is the key strategy.

[2] Robert, C. & Casella, G. Monte Carlo Statistical Methods (2013) Springer Science & Business Media

[1] Nelson, A. R. J. & Prescott, S. W. *refnx*: neutron and X-ray reflectometry analysis in Python. (2019). J. Appl. Cryst. 52, 193-200

Sample efficiency ϵ for random exploration (MC IS)

ϵ reflects number of samples required to find solution



$\boxed{\epsilon \approx \sigma / u}$

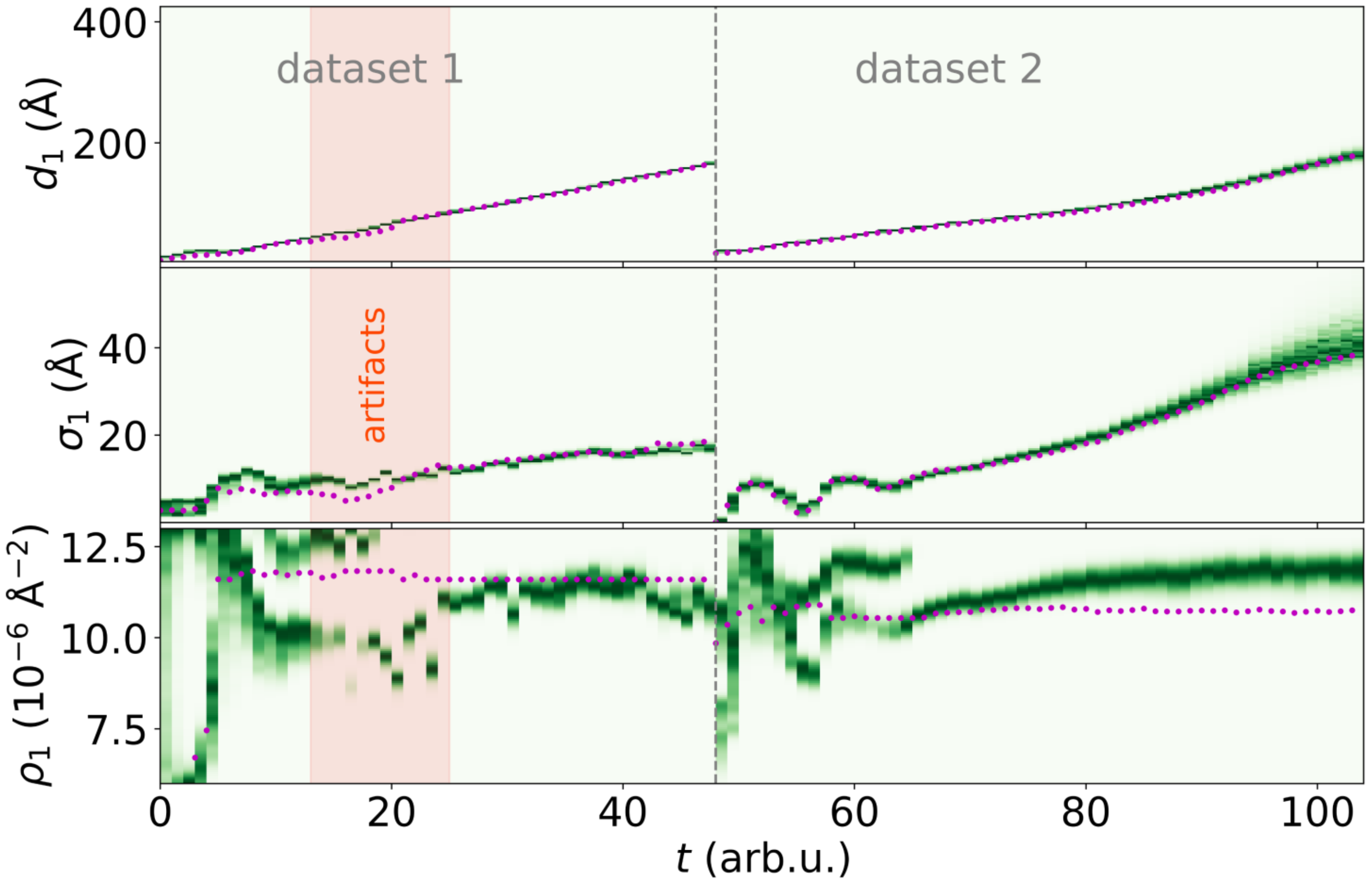
Typically,
$\epsilon \in [10^{-1}, 10^{-3}]$

For more open parameters (higher dimensionality), $\epsilon \approx \left(\frac{\sigma}{u}\right)^{dim}$ $\quad \left(\epsilon \in \left[10^{-dim}, 10^{-3 * dim}\right]\right)$

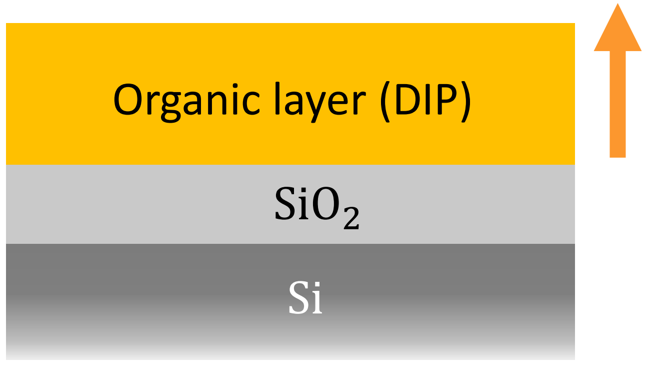
Sample efficiency ϵ on *in situ* experimental XRR data

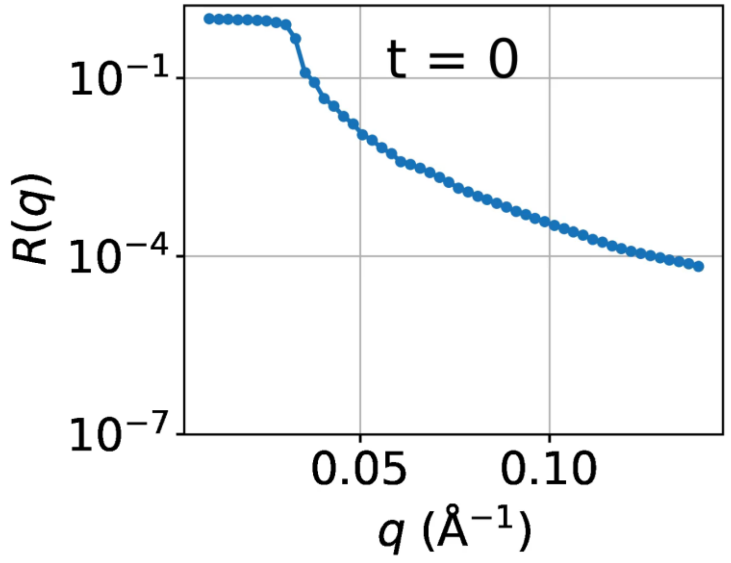
Parameter distributions for thickness, roughness, SLD of DIP (Monte Carlo importance sampling)

● Manual fit with differential evolution



In situ data (**DIP growth**)

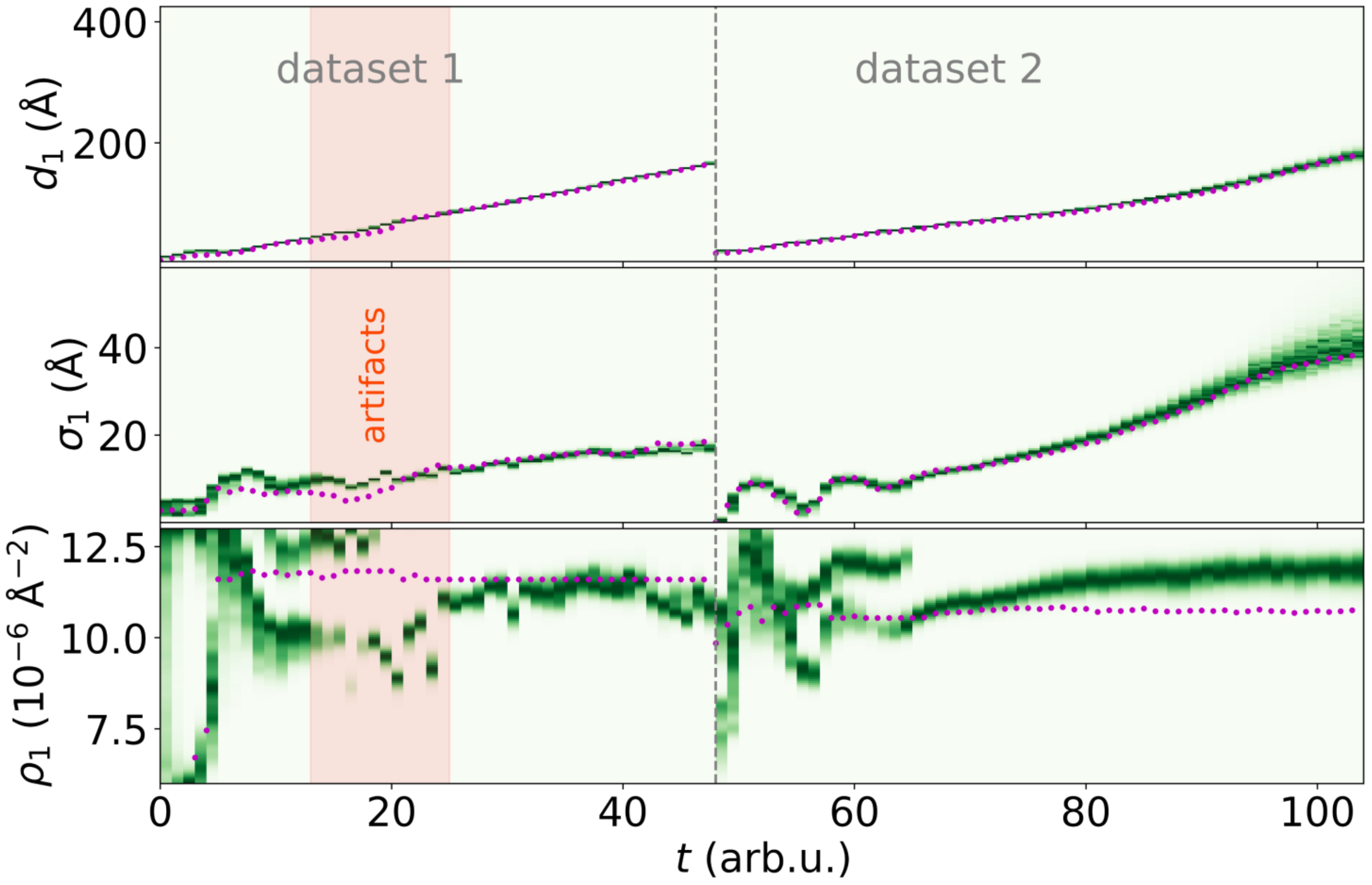




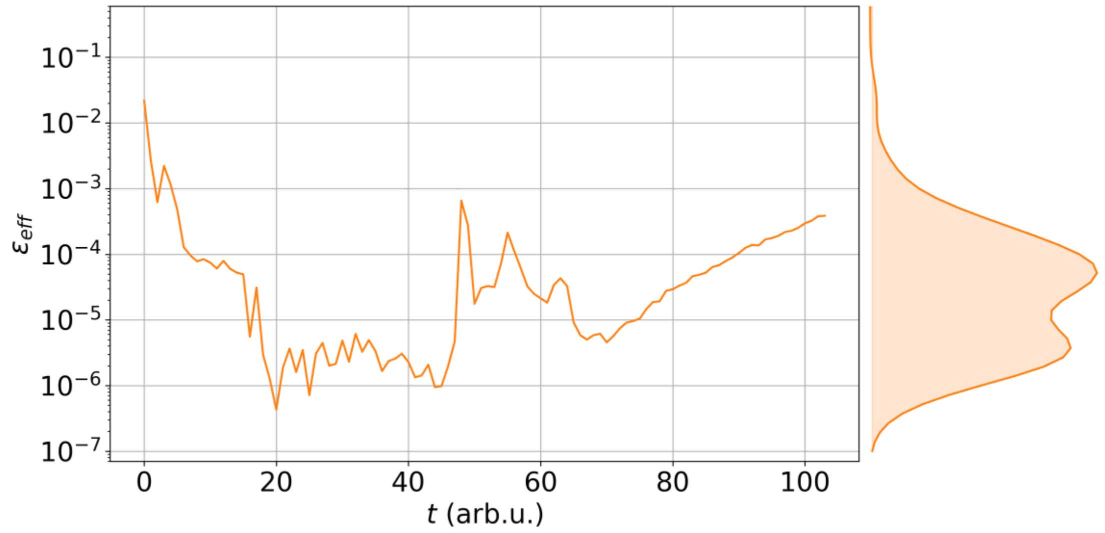
Sample efficiency ϵ on *in situ* experimental XRR data

Parameter distributions for thickness, roughness, SLD of DIP
(Monte Carlo importance sampling)

● Manual fit with differential evolution



Monte Carlo sampling efficiency
(only 3 open parameters)



Summary on conventional methods

- Conventional analysis primarily relies on random exploration.
- It becomes practically unfeasible **exponentially fast** with increasing number of parameters & parameter ranges.
- Even for a small number of parameters, conventional fits based on differential evolution are unreliable (only find one solution).
- For small number of parameters & narrow bounds, we can use Monte Carlo
- In typical experimental scenarios, sample efficiency is likely to be extremely small.