Université Grenoble Alpes

Inria

PhD Thesis Defence Presentation

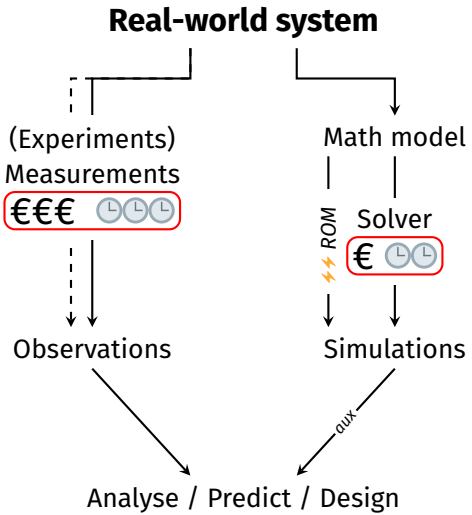
High Performance Online Deep Neural Network Training from Synthetic Data with Active Learning

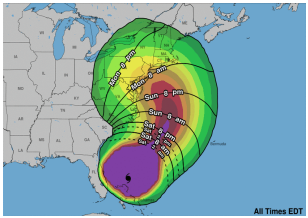
by **Sofya DYMCHENKO**

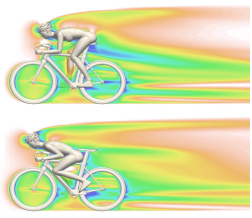
April 21, 2026 — Université Grenoble Alpes & INRIA

Reviewers:	**Mathias NIEPERT**, Univ. of Stuttgart
	Ronan FABLET, IMT Atlantique
Examiners:	**Kyle CRANMER**, Univ. of Wisconsin-Madison
	Eric BLAYO, Univ. Grenoble Alpes
Thesis Director:	**Bruno RAFFIN**, Univ. Grenoble Alpes & INRIA

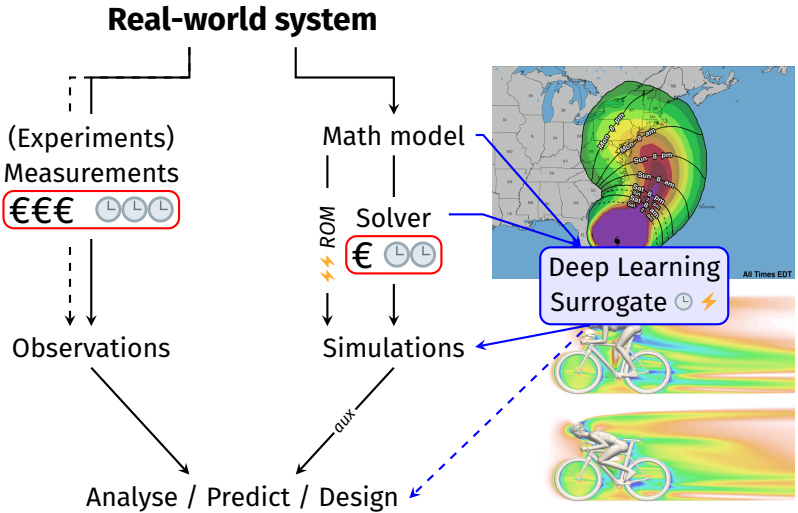
Scientific computing: simulate real-world



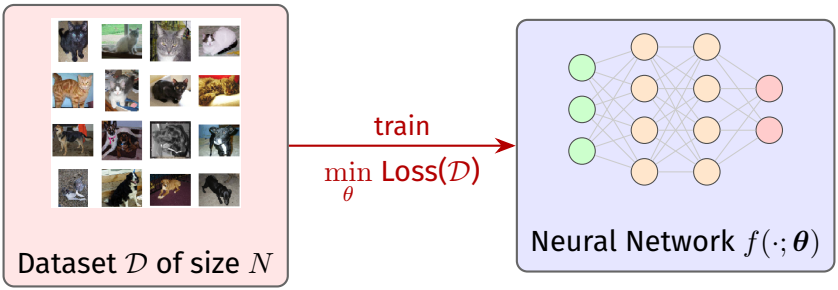




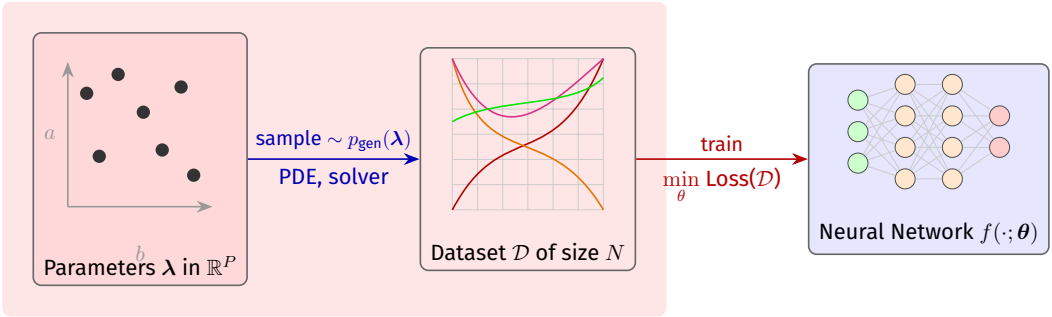
Scientific computing: simulate real-world



Deep learning training: observational data



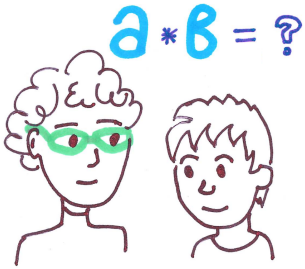
Deep learning training: synthetic data



💡 Which data to create to train well?

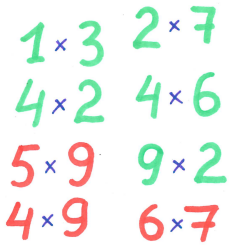
Analogy: active learning

Exam: multiplication for $a, b \in \{1, 2, ..., 20\}$



📝 Choose a and b?

Passive: no prior information, sample many uniform a and b



😐 Redundant exercises

Active: acquire information, prepare relevant a and b

💯 Only useful practice

Active learning (AL) for deep learning

(Settles, 2009)

Given set of data without labels

select

Pool-based AL

Train NN

Label data

CAT

Continuous parameter space Λ

sample

Discrete set

select

Pool-based AL

Train NN

Create data

sample

Query-synthesis AL

🤔 How to apply query-synthesis AL for surrogate training?

C1: *Breed* method

Query-synthesis active learning for NN training

Dynamical system example: a rolling ball on the surface

Parameter: initial position $x \in X \subset \mathbb{R}^D$

Black-box oracle $\mathcal{O}$ output:

$$x \xrightarrow{x^{(i+1)}=x^{(i)}-\eta\nabla f(x^{(i)})} x^{\min} \in \{x^A, x^B\}$$

NN predicts $f_\theta(x) \approx \mathcal{O}(x)$ for any $x \in X$

Loss: $\mathbb{E}_{\mathcal{D}}\left[\ell(f_\theta(x_i) - x_i^{\min})\right] \xrightarrow{\theta} 0$

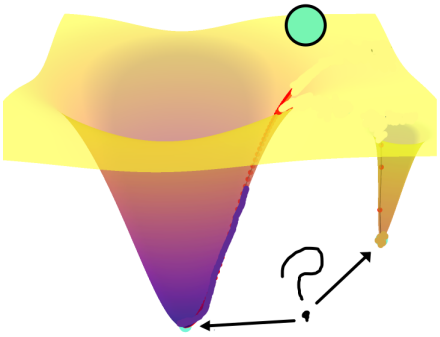
Training data:

(0) Choose prior $p_{\text{gen}}(X)$

(1) Sample $\mathcal{P} = \{x_i \sim p_{\text{gen}}(X)\}_{i=1}^N$

(2) Create $\mathcal{D} = \{(x_i, \mathcal{O}(x_i) = x_i^{\min}) \mid x_i \in \mathcal{P}\}_{i=1}^N$

What samples are useful and not?



Dynamical system example: a rolling ball on the surface

Parameter: initial position $x \in X \subset \mathbb{R}^D$

Black-box oracle $\mathcal{O}$ output:

$$x \xrightarrow{x^{(i+1)}=x^{(i)}-\eta\nabla f(x^{(i)})} x^{\min} \in \{x^A, x^B\}$$

NN predicts $f_\theta(x) \approx \mathcal{O}(x)$ for any $x \in X$

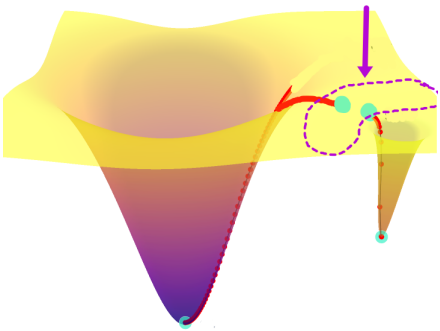
Loss: $\mathbb{E}_{\mathcal{D}}\left[\ell(f_\theta(x_i) - x_i^{\min})\right] \xrightarrow{\theta} 0$

Training data:

(0) Choose prior $p_{\text{gen}}(X)$

(1) Sample $\mathcal{P} = \{x_i \sim p_{\text{gen}}(X)\}_{i=1}^N$

(2) Create $\mathcal{D} = \{(x_i, \mathcal{O}(x_i) = x_i^{\min}) \mid x_i \in \mathcal{P}\}_{i=1}^N$



What samples are useful and not?

💡 Uncertain = useful!

🤔 How to get $p_{\text{gen}} \sim p(\text{uncertainty})$

Challenges and inspired solutions

Estimation:

No closed form for uncertainty, can't sample from it
↪ approximate with a weighted density estimator

Uncertainty estimates are costly (Nabian et al., 2021)
↪ cheaper proxy NN loss

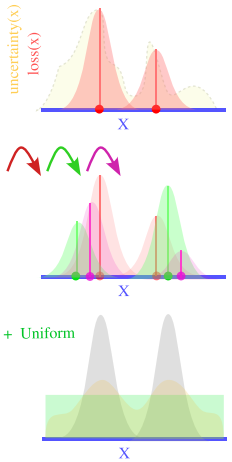
Adaptation:

NN evolves, can't fix $p_{\text{gen}}(x)$
↪ update training data in rounds

Sudden data distribution shift, NN destabilisation (Cappé et al., 2004)
↪ generate new samples close to previous

Regularisation:

Overfitting, closed loop, mode collapse (Sutton and Barto, 2018)
↪ mix with uniform distribution to explore



(these ideas intersect with: population Monte-Carlo, reinforcement learning exploration-exploitation trade-off, inference, and more)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

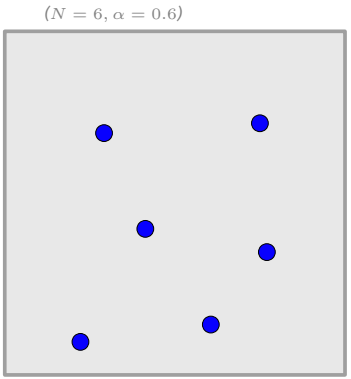
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$

(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

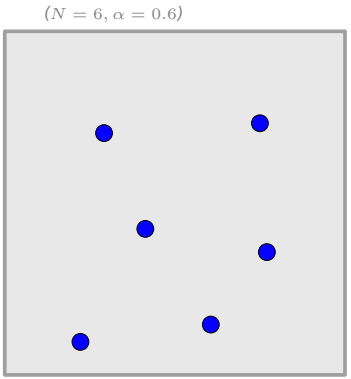
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$

(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

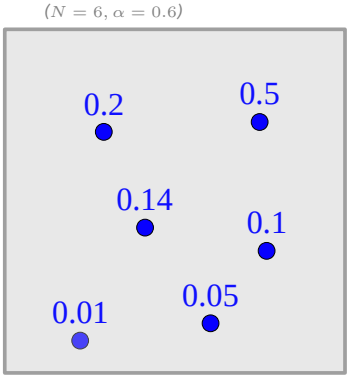
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$

(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

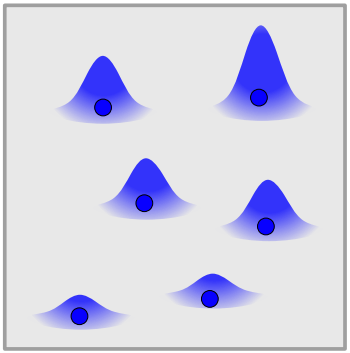
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$
(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

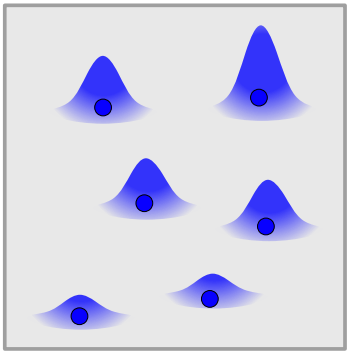
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$
(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

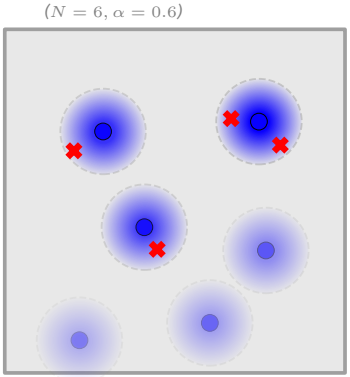
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$

(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

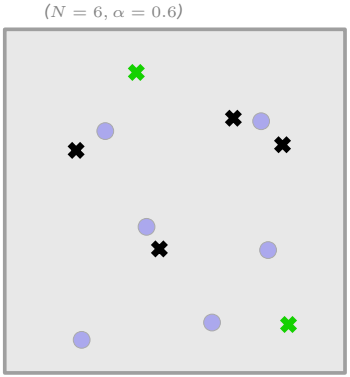
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat



p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$

(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Breed: progressively populate high-error regions

Initialise dataset $\mathcal{S} = \{N \text{ samples } \sim \mathrm{Unif}(X)\}$

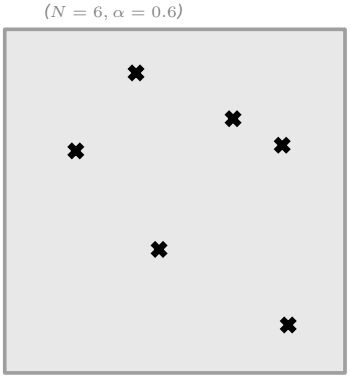
1. Train NN for p iterations on $\mathcal{S}$
2. Compute loss l_i for each sample $x_i \in \mathcal{S}$
3. Build proposal $\mathrm{GM}(x) \propto \sum_1^N l_i \cdot \mathcal{N}(x_i; \sigma\mathbb{I})$
4. Create mixture $\alpha\,\mathrm{GM}(x) + (1-\alpha)\,\mathrm{Unif}(X)$
5. Sample $N_c = \alpha \cdot N$ concentrated samples:

$$\mathcal{S}_c = \{N_c \text{ samples } \sim \mathrm{GM}(x)\}$$

6. Sample remaining $N - N_c$ uniform samples:

$$\mathcal{S}_u = \{(N - N_c) \text{ samples } \sim \mathrm{Unif}(X)\}$$

7. Update dataset $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$, repeat

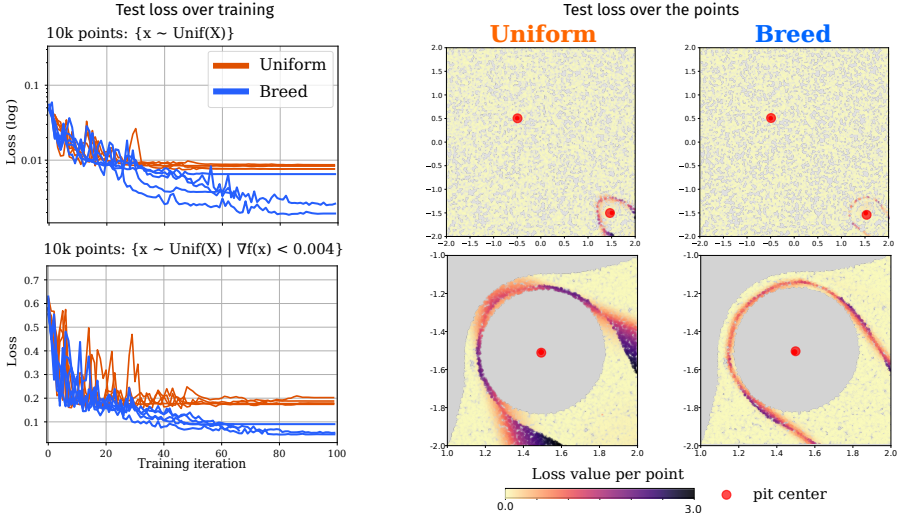


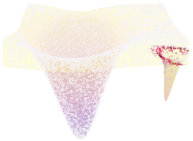
p: adaptivity control (fixed)
σ: diversity control (fixed)
α: mixing control, linearly increasing:
$\alpha = \max(\alpha_1 \cdot i + \alpha_0, \alpha_{\max}) \in [0, 1]$

(curriculum learning: from exploration to exploitation)

(Dymchenko and Raffin, 2023)

Toy benchmark results: two validation sets





(Raissi et al., 2017)

Physics-Informed NNs: learn from governing equations

Idea: use a mathematical model for training

System example: *1D Allen-Cahn equation*

$$F[u](x,t) = u_t - 0.0001u_{xx} + 5u^3 - 5u = 0$$

(reaction-diffusion, sharp gradients) for $(\boldsymbol{x}, t) \in \Omega \subset \mathbb{R}^2$

NN approximates unknown solution:

$$u_\theta(\boldsymbol{x}, t) \approx u(\boldsymbol{x}, t) \text{ over } \Omega$$

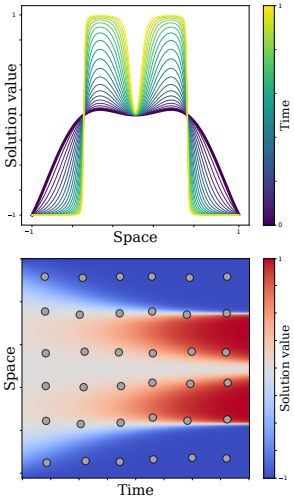
Constraint: *(generally, can include initial/boundary conditions)*

u_θ satisfies system F across the domain

Imposing constraint:

Collocation points: $\mathcal{S} = \{(\boldsymbol{x}_i, t_i) \sim p_{\text{gen}}(\Omega)\}_{i=1}^N$

Residual loss: $\frac{1}{N}\sum_{i=1}^N F[u_\theta](\boldsymbol{x}_i, t_i) \overset{\boldsymbol{\theta}}{\to} 0$



Physics-Informed NNs: learn from governing equations

(Raissi et al., 2017)

Idea: use a mathematical model for training

System example: *1D Allen-Cahn equation*

$$F[u](x,t) = u_t - 0.0001u_{xx} + 5u^3 - 5u = 0$$

(reaction-diffusion, sharp gradients) for $(\boldsymbol{x}, t) \in \Omega \subset \mathbb{R}^2$

NN approximates unknown solution:

$$u_\theta(\boldsymbol{x}, t) \approx u(\boldsymbol{x}, t) \text{ over } \Omega$$

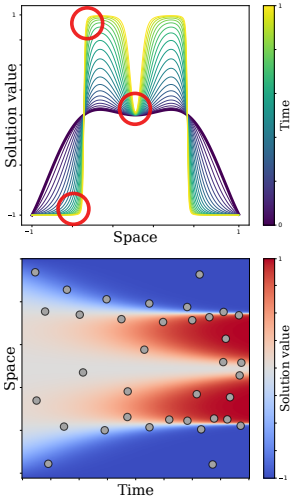
Constraint: *(generally, can include initial/boundary conditions)*

u_θ satisfies system F across the domain

Imposing constraint:

Collocation points: $\mathcal{S} = \{(\boldsymbol{x}_i, t_i) \sim p_{\text{gen}}(\Omega)\}_{i=1}^N$

Residual loss: $\frac{1}{N}\sum_{i=1}^N F[u_\theta](\boldsymbol{x}_i, t_i) \overset{\boldsymbol{\theta}}{\to} 0$



Data-driven surrogates: generation bottleneck (Kovachki et al., 2021)

Idea: use simulation data, reuse DL methodology

↪ create ground-truth data with solver

Solver S_λ for system $F[u](\boldsymbol{x}, t; \boldsymbol{\lambda})$

Solution state is discretised spatially:

$$u(\boldsymbol{x}, t; \boldsymbol{\lambda}) \overset{h}{=} \boldsymbol{X}_\lambda^{(t)} \in \mathbb{R}^{N_x^D}$$

Trajectory of solution states $t \to t + \Delta t$:

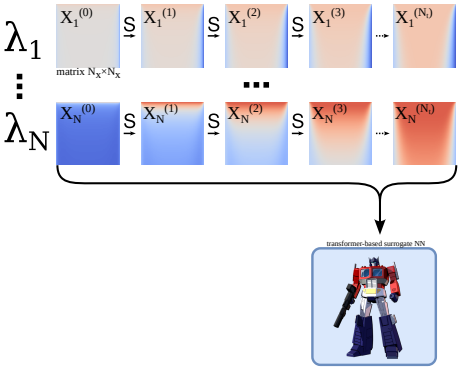
$$S_\lambda(\boldsymbol{X}_\lambda^{(i)}) = \boldsymbol{X}_\lambda^{(i+1)} \text{ for } i = 0, \ldots, N_t$$

Simulation data:

↪ many $\boldsymbol{\lambda}_i \in \Lambda \subseteq \mathbb{R}^P$

↪ large series of meshes

↪ computationally expensive



Every data sample comes at a **large cost**!

Data-driven surrogates: generation bottleneck (Kovachki et al., 2021)

Idea: use simulation data, reuse DL methodology

↪ create ground-truth data with solver

Solver S_λ for system $F[u](\boldsymbol{x}, t; \boldsymbol{\lambda})$

Solution state is discretised spatially:

$$u(\boldsymbol{x}, t; \boldsymbol{\lambda}) \overset{h}{=} \boldsymbol{X}_\lambda^{(t)} \in \mathbb{R}^{N_x^D}$$

Trajectory of solution states $t \to t + \Delta t$:

$$S_\lambda(\boldsymbol{X}_\lambda^{(i)}) = \boldsymbol{X}_\lambda^{(i+1)} \text{ for } i = 0, \ldots, N_t$$

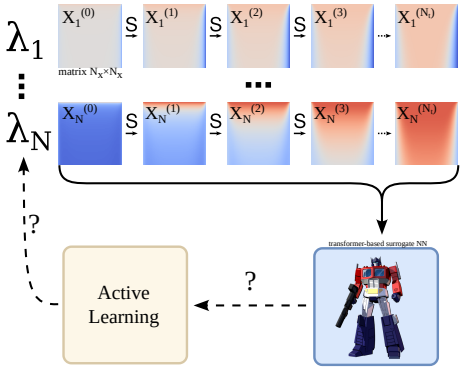
Simulation data:

↪ many $\boldsymbol{\lambda}_i \in \Lambda \subseteq \mathbb{R}^P$

↪ large series of meshes

↪ computationally expensive

Every data sample comes at a **large cost**!



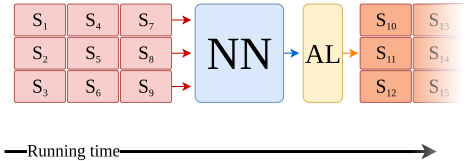
🤔⚡ How to make every solver call $S_{\boldsymbol{\lambda}_i}$ count?

C2: Online active learning

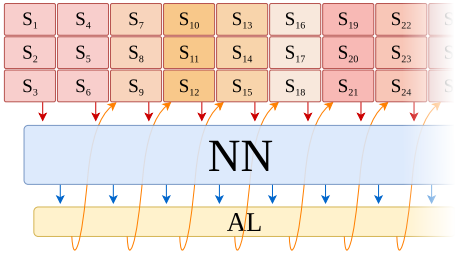
Active sampling mechanism in Melissa and online Breed

Offline rounds vs. online continuous integration

Instead of proceeding in rounds...



...everything runs simultaneously?



🔀🔄 Continuous feedback loop ⇒ increase impact of each solver run

🤔 How to achieve this?

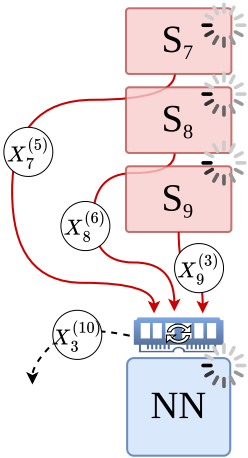
Online training: setting

(Schouler et al., 2023)

MELISSA framework: online surrogate training

- Asynchronous parallel:
  K concurrent solvers clients and training server
- Avoids storage:
  produced states $\boldsymbol{X}_{\lambda_j}^{(t)}$ are sent directly to a fixed capacity memory buffer $\mathcal{B}$
- Avoids idle resources:
  training is on while $\mathcal{B} \neq \emptyset$, solvers are on while $\mathcal{B}$ contains "seen" samples

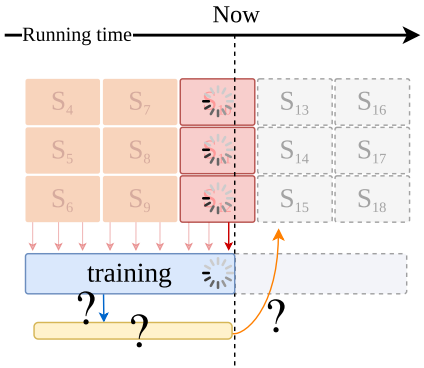
🤔🔄 How to implement an infrastructure for active learning?



Online training: challenges

Goal for AL process:

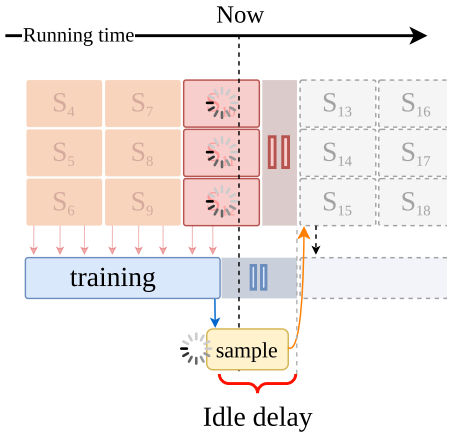
- no experiment walltime overheads
- low latency between feedback and data exposure



Online training: challenges

Goal for AL process:

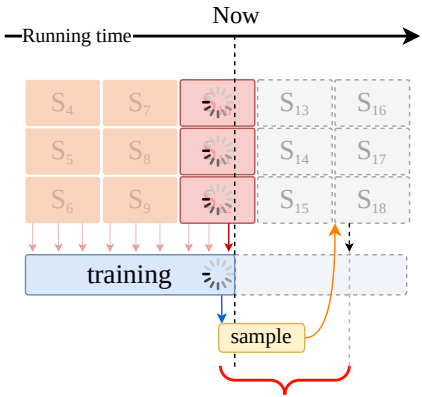
- no experiment walltime overheads
- low latency between feedback and data exposure



Online training: challenges

Goal for AL process:

- no experiment walltime overheads
- low latency between feedback and data exposure



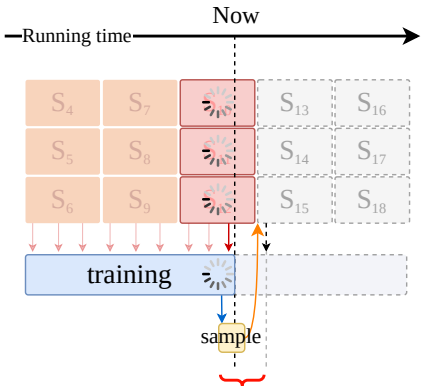
Online training: challenges

Goal for AL process:

- no experiment walltime overheads
- low latency between feedback and data exposure

NEW Implemented active sampling mechanism in Melissa!

Separate server rank, optimised to continuously:
- receive AL information from training process
- update solver parameters for immediate next clients



(Dymchenko et al., 2024)

Breed online: moving population aggregated historical losses

Obstacle: No set of all past samples, no additional loss scoring allowed

Adaptations:

↪ $\mathcal{Q}$ population of M parameters linked to the most-recently-seen training samples
↪ l_{jti}: use already computed loss from training iterations
↪ ∂l_{jti}: normalise by batch statistics and threshold harder samples
↪ ∂l_j: aggregate over trajectory and training steps

$$l_{jti} = \ell(\boldsymbol{X}_{\lambda_j}^{(t)}; \boldsymbol{\theta}_i), \quad m_i = \underset{\boldsymbol{X} \in \text{batch}}{\text{mean}}\ \ell(\boldsymbol{X}; \boldsymbol{\theta}_i), \quad s_i = \underset{\boldsymbol{X} \in \text{batch}}{\text{deviation}}\ \ell(\boldsymbol{X}; \boldsymbol{\theta}_i)$$

$$\partial l_{jti} = \max\left(\frac{l_{jti} - m_i}{s_i}, 0\right), \quad \partial l_j = \underset{t,i}{\text{mean}}\ \partial l_{jti}, \quad \text{GM}(\boldsymbol{\lambda}) \propto \sum_{\lambda_j \in \mathcal{Q}} \partial l_j \cdot \mathcal{N}(\boldsymbol{\lambda}_j; \sigma \mathbb{I}_P)$$

✅ Up-to-date proposal focused on samples with above-average loss over trajectory exhibited during training

Experiments: 1D PDEs of different difficulty and dynamics (using APEBench)

(Koehler et al., 2024)

(we combine in total 10 cases with fixed a_i, b, including conservative KS with $(u^2)_x$)

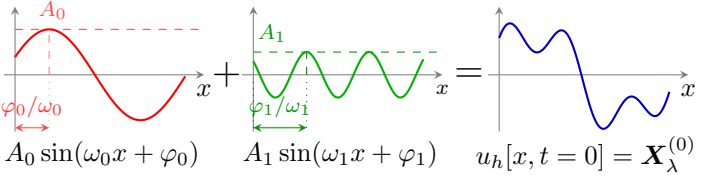
Nonlinear *Kortweg-de Vries* (KdV)

$$u_t = a_3 u_{xxx} - a_4 u_{xxxx} - 0.5b(u^2)_x$$

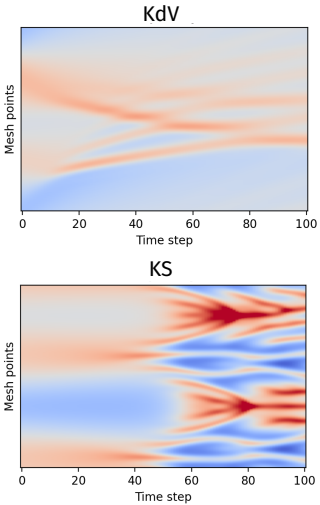
Chaotic *Kuramoto-Sivashinsky* (KS)

$$u_t = -a_2 u_{xx} - a_4 u_{xxxx} - 0.5b(u_x)^2$$

Initial condition: waves superposition

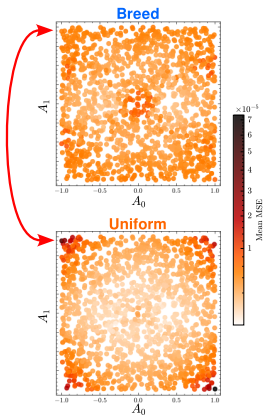


AL: curate initial state – parameters $\boldsymbol{\lambda} = (A_0, A_1, \varphi_0, \varphi_1)$

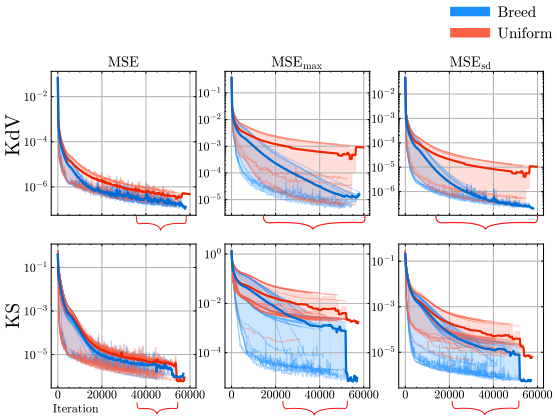


Experiments: results *(not published work)*

Test loss over A_0, A_1



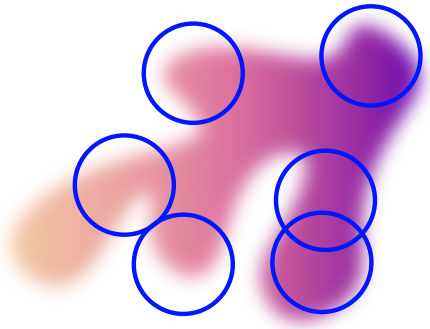
Test loss over training (9 seeds)



⚡🏆 More reliable performance with fewer data ⇒ lower cost!

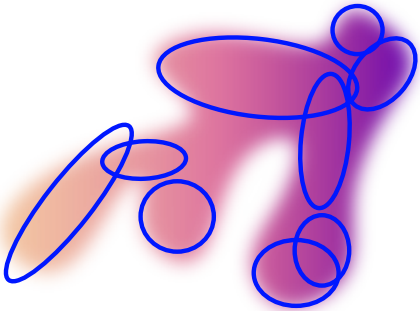
Limitation of Breed

- **Local**:
  proposal is anchored to visited samples
  $\hookrightarrow$ no global view
- **Rigid**:
  fixed isotropic Gaussian kernels
  $\hookrightarrow$ blind to local geometry
- **Statistically poor**:
  degrades as $D = \dim(\Lambda)$ grows;
  full-covariance requires $O(D^3)$
  $\hookrightarrow$ not scalable



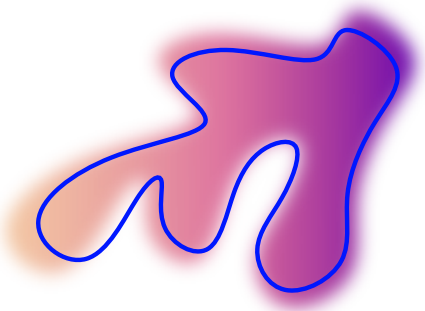
Limitation of Breed

- **Local**:
  proposal is anchored to visited samples
  $\hookrightarrow$ no global view
- **Rigid**:
  fixed isotropic Gaussian kernels
  $\hookrightarrow$ blind to local geometry
- **Statistically poor**:
  degrades as $D = \dim(\Lambda)$ grows;
  full-covariance requires $O(D^3)$
  $\hookrightarrow$ not scalable



Limitation of Breed

- **Local**:
  proposal is anchored to visited samples
  ↪ no global view
- **Rigid**:
  fixed isotropic Gaussian kernels
  ↪ blind to local geometry
- **Statistically poor**:
  degrades as $D = \dim(\Lambda)$ grows;
  full-covariance requires $O(D^3)$
  ↪ not scalable



🧠🤔 How can we approximate it better?

C3: *OGAS* method

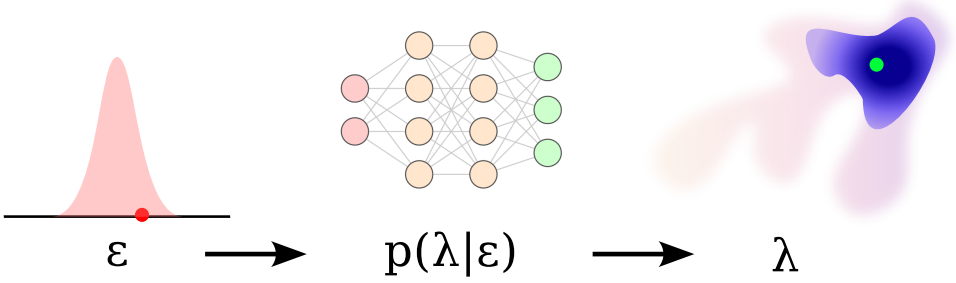
Generative AL for high-dimensional parameter spaces

Neural proposal: learn where to sample

Define: $\varepsilon \propto$ expected error level

Train: a conditional generative NN $\hat{p}(\boldsymbol{\lambda}|\varepsilon)$ on pairs $(\boldsymbol{\lambda}, \ell_{jti})$

Proposal: sample desired ε, obtain $\boldsymbol{\lambda}$ from generator



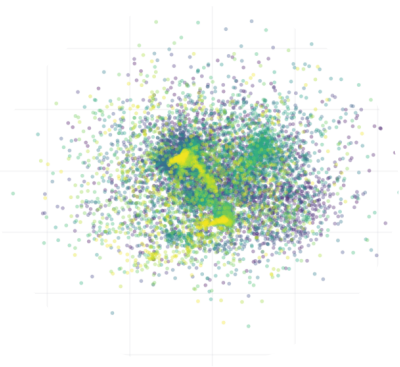
OGAS: Online Generative Active Sampling *(under review for ICML'26)*

Generator:
fast MLP-based conditional diffusion model (Ho et al., 2020)

Training:
ε is a z-standardised $\ell(\boldsymbol{X}_j^{(t)}; \boldsymbol{\theta}_i)$ with moving statistics
pairs $(\boldsymbol{\lambda}, \varepsilon)$ sent to a FIFO buffer for generator training
objective is corrected with importance weights

New parameters:
proportional prior over ε to focus on harder data
mix with $\mathrm{Unif}(\Lambda)$ to ensure exploration

Evaluation:
three 2D PDEs, three surrogate architectures
AL: curate IC and PDE parameters with $\dim(\Lambda) \leq 308$



Experiments: example results

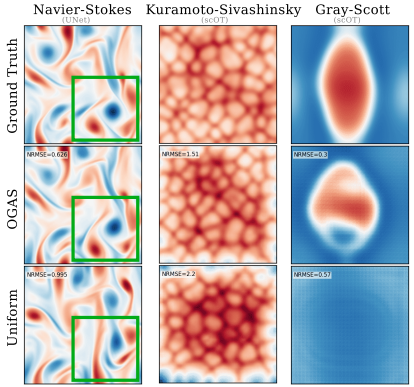
FNO (28M parameters) on Kuramoto-Sivashinsky PDE

Strategy	Mean ($\times 10^{-3}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)
OGAS-L	9.49 ± 0.28	1.76 ± 0.06	3.26 ± 0.18
Breed	11.32 ± 0.35	2.53 ± 0.41	5.02 ± 0.58
SBAL	9.26 ± 0.36	3.23 ± 0.42	3.34 ± 0.11
Uniform	8.30 ± 0.32	4.02 ± 0.16	5.95 ± 0.12

scOT (180M parameters) on Navier-Stokes PDE

Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)
OGAS-L	1.72 ± 0.19	4.42 ± 0.23	5.45 ± 0.34
Breed	2.15 ± 0.21	7.96 ± 1.07	10.10 ± 1.44
SBAL	1.82 ± 0.18	6.30 ± 0.29	8.32 ± 0.34
Uniform	1.91 ± 0.22	6.54 ± 0.47	8.83 ± 0.99

Overhead delay:
0.2% for OGAS, 0.7% for Breed, 10% for SBAL

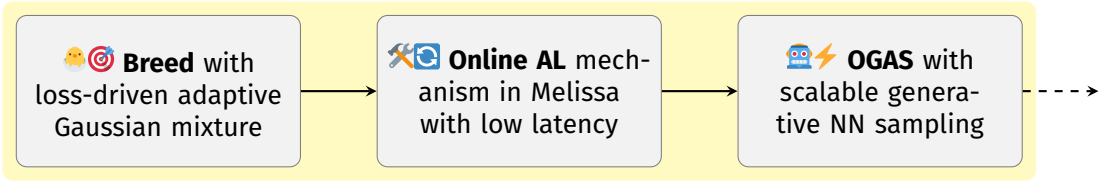


⚡🏆 Minimal delays, robustness for tail-metrics, mean improvement for capable models

Conclusion

Main takeaways

RQ: How to improve training of deep learning surrogates with QSAL?



- ⚙️ Improved training data – not surrogate's NN architectures
- 🚀 Effective navigation of parameters space – reduced data cost
- 📈 AL gains scale with model capacity – the better the model, the more AL helps
- ♾️ Continuous online feedback – each solver call is maximally exploited

What is next?

Short-term for online QSAL in deeper surrogate training:

- **Foundation models:** (McCabe et al., 2025)
  - require huge amount of data for pretraining (Ashton et al., 2025)
  - choosing PDEs and ICs, heterogeneous parameters, ...
- **Geometry-aware AL**: beyond regular grids, choosing geometries or points... (Alkin et al., 2025)
- **Connecting to related frameworks:**
  Simulation-Based Inference (Cranmer et al., 2022), Reinforcement Learning (Rolnick et al., 2019)

Open question:

⚖️ At what point does **learning how to learn become harder than learning itself**?

🌱🌍 Finding the efficient path itself comes at a cost...
...≈**4 tons** CO_2eq emitted along the way of this thesis.

References I

Burr Settles. Active Learning Literature Survey. Technical Report, University of Wisconsin-Madison Department of Computer Sciences, 2009. URL https://minds.wisconsin.edu/handle/1793/60660.

Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *arXiv: Learning*, April 2021.

O Cappé, A Guillin, J. M Marin, and C. P Robert. Population Monte Carlo. *Journal of Computational and Graphical Statistics*, 13(4):907–929, December 2004. ISSN 1061-8600, 1537-2715. doi: 10.1198/106186004X12803. URL http://www.tandfonline.com/doi/abs/10.1198/106186004X12803.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, USA, 2 edition, November 2018. ISBN 978-0-262-03924-6. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

References II

Sofya Dymchenko and Bruno Raffin. Loss-driven sampling within hard-to-learn areas for simulation-based neural network training. In *MLPS 2023 - Machine Learning and the Physical Sciences Workshop at NeurIPS 2023 - 37th Conference on Neural Information Processing Systems*, pages 1–5, New Orleans, United States, December 2023. URL https://hal.science/hal-04305233.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations, November 2017. URL http://arxiv.org/abs/1711.10561.

Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating Propagation Failures in Physics-informed Neural Networks using Retain-Resample-Release (R3) Sampling. In *Proceedings of the 40th International Conference on Machine Learning*, pages 7264–7302. PMLR, July 2023. URL https://proceedings.mlr.press/v202/daw23a.html.

Nikola B. Kovachki, Zong-Yi Li, Burigede Liu, K. Azizzadenesheli, K. Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces. *arXiv.org*, 2021.

References III

Marc Schouler, Robert Alexander Caulk, Lucas Meyer, Théophile Terraz, Christoph Conrads, Sebastian Friedemann, Achal Agarwal, Juan Manuel Baldonado, Bartłomiej Pogodziński, Anna Sekuła, Alejandro Ribes, and Bruno Raffin. Melissa: Coordinating large-scale ensemble runs for deep learning and sensitivity analyses. *Journal of Open Source Software*, 8(86):5291, June 2023. ISSN 2475-9066. doi: 10.21105/joss.05291. URL https://joss.theoj.org/papers/10.21105/joss.05291.

Sofya Dymchenko, Abhishek Purandare, and Bruno Raffin. MelissaDL x Breed: Towards Data-Efficient On-line Supervised Training of Multi-parametric Surrogates with Active Learning. In *SC-W 2024 - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 1–9, Atlanta (Georgia), United States, November 2024. IEEE. URL https://hal.univ-brest.fr/hal-04712480.

Felix Koehler, Simon Niedermayr, Rüdiger Westermann, and Nils Thuerey. APEBench: A Benchmark for Autoregressive Neural Emulators of PDEs, October 2024. URL http://arxiv.org/abs/2411.00180.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020. URL http://arxiv.org/abs/2006.11239.

References IV

Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Regaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze W. K. Wong, Hadi Sotoudeh, Alberto Bietti, Irina Espejo, Rio Fear, Siavash Golkar, Tom Hehir, Keiya Hirashima, Geraud Krawezik, Francois Lanusse, Rudy Morel, Ruben Ohana, Liam Parker, Mariel Pettee, Jeff Shen, Kyunghyun Cho, Miles Cranmer, and Shirley Ho. Walrus: A Cross-Domain Foundation Model for Continuum Dynamics, November 2025. URL http://arxiv.org/abs/2511.15684.

Neil Ashton, Johannes Brandstetter, and Siddhartha Mishra. Fluid Intelligence: A Forward Look on AI Foundation Models in Computational Fluid Dynamics, November 2025. URL http://arxiv.org/abs/2511.20455.

Benedikt Alkin, Maurits Bleeker, Richard Kurle, Tobias Kronlachner, Reinhard Sonnleitner, Matthias Dorfer, and Johannes Brandstetter. AB-UPT: Scaling Neural CFD Surrogates for High-Fidelity Automotive Aerodynamics Simulations via Anchored-Branched Universal Physics Transformers, October 2025. URL http://arxiv.org/abs/2502.09692.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. 2022. doi: 10.1073/pnas.1912789117. URL www.pnas.org/cgi/doi/10.1073/pnas.1912789117.

References V

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience Replay for Continual Learning, November 2019. URL http://arxiv.org/abs/1811.11682.