MelissaDL x Breed: Towards Data-Efficient On-line Supervised Training of Multi-parametric Surrogates with Active Learning

Sofya Dymchenko, Abhishek Purandare, Bruno Raffin

MelissaDL x Breed: Towards Data-Efficient On-line Supervised Training of Multi-parametric Surrogates with Active Learning

Sofya Dymchenko
sofya.dymchenko@inria.fr
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Grenoble, France

Abhishek Purandare
abhishek.purandare@inria.fr
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Grenoble, France

Bruno Raffin
bruno.raffin@inria.fr
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Grenoble, France

Abstract

Artificial intelligence is transforming scientific computing with deep neural network surrogates that approximate solutions to partial differential equations (PDEs). Traditional off-line training methods face issues with storage and I/O efficiency, as the training dataset has to be computed with numerical solvers up-front. Our previous work, the Melissa framework, addresses these problems by enabling data to be created "on-the-fly" and streamed directly into the training process. In this paper we introduce a new active learning method to enhance data-efficiency for on-line surrogate training. The surrogate is direct and multi-parametric, i.e., it is trained to predict a given timestep directly with different initial and boundary conditions parameters. Our approach uses Adaptive Multiple Importance Sampling guided by training loss statistics, in order to focus NN training on the difficult areas of the parameter space. Preliminary results for 2D heat PDE demonstrate the potential of this method, called Breed, to improve the generalization capabilities of surrogates while reducing computational overhead.

CCS Concepts

• **Applied computing** → *Physical sciences and engineering*; • **Computing methodologies** → **Distributed artificial intelligence**; **Online learning settings**.

Keywords

Active Learning, Adaptive Multiple Importance Sampling, Surrogates, On-line Training, Data-efficiency

1 Introduction

The advancement of artificial intelligence (AI) applications in the natural and physical sciences, referred to as *AI4Science (AI4S)*, has noticeably accelerated in recent years [26]. Remarkable examples include molecule docking modeling (AlphaFold 3 [1]) and weather forecasting (NeuralGCM [22], ClimaX [33], Aurora [5]). This progress has been enabled by combining large training data sets, advanced neural architectures, and parallel computational resources.

Particular attention has been given to *deep surrogates* — a class of neural networks (NNs) that approximate the solution of partial differential equations (PDEs) used to describe various scientific phenomena. PDE solutions are classically obtained using numerical methods such as Finite Elements Method or Finite Volumes Method. However, these methods are compute and memory intensive. Deep surrogates are expected to deliver quality solutions during inference at a fraction of the memory and computational cost of these solvers.

Deep surrogates can be trained with little to no data, as in Physics Informed NNs (PINNs) [36, 41], where the PDE residuals, initial and boundary conditions (IC, BC) are imposed as soft constraints through the loss. Yet, a wide variety of neural architectures, ranging from Graph Neural Networks [35] and Neural Operators [2, 27] to Diffusion Models [23] and Visual Transformers [18], are trained with numerical simulations data. A surrogate can also be trained in a multi-parametric context with varying ICs and BCs to develop a more generic model. [18] presented a general purpose PDE foundation model that can be finetuned to obtain a specific PDE solution, while [10] proposed a neural operator architecture that generalizes to unseen geometries. Since all these NN are trained in a supervised manner — using data produced by the PDE solver they aim to substitute — their performance depends on both training set quantity and quality.

A standard approach is to train surrogates *off-line*: first, a dataset is generated with traditional PDE solvers and stored on disk; then, the surrogate is trained in an epoch-based manner by reading back the dataset from disk. When scaled, this approach suffers from two main limitations: 1) storage capabilities limit the dataset size, thus compromising data quantity, fidelity, and/or diversity; 2) writing and reading the dataset creates an I/O bottleneck, thus impairing efficiency during training. In previous works [28, 29, 38], we tackled both limitations by demonstrating an *on-line* training approach for multi-parametric surrogates with the Melissa framework: the dataset is generated and directly sent to training, bypassing the storage. It allowed us to train a surrogate faster and with higher generalization abilities due to a significantly larger training dataset. In addition, on-line training enables the steering of the data creation

process along the NN training, which opens the way to *active learning (AL)* techniques. AL is a data-centric approach that intends to choose informative samples to improve data efficiency for NN training. AL is a paradigm shift from a model-centric to a data-centric approach, which focuses on the data relevance and quality rather than the model to get better performance [37].

In this paper, we present our work-in-progress *active learning* method, called Breed, for data-efficient on-line surrogate training with Melissa. To define the input parameters of the next set of solver instances to run, the algorithm relies on the loss values obtained during training and uses Adaptive Importance Sampling method in order to focus on hard-to-learn parameter space regions. We show that for a 2D Heat PDE, our method chooses initial and boundary conditions with dissimilar temperature values (which are meant to be more challenging to learn by the NN) and, compared to random sampling, the NN overfits noticeably less.

2 Background

2.1 Simulation-based deep learning

Consider partial differential equations (PDE) of the form:

$$\mathcal{N}[u](x,t) = f(x,t) \tag{1}$$

$$\mathcal{B}_i[u](x_i,t) = g_i(x_i,t) \qquad \forall x \in \mathcal{X}, x_i \in \delta\mathcal{X}_i, t \in \mathcal{T} \tag{2}$$

$$\mathcal{I}[u](x,0) = h(x) \tag{3}$$

where $u(x,t)$ is a quantity of interest described by the PDE (e.g., temperature value) defined on a bounded domain $\Omega = \mathcal{X} \cup \mathcal{T} \subset R^{d+1}$. $\mathcal{N}[u]$ is a differential operator acting on $u(x,t)$, $\mathcal{B}_i[u]$ are boundary conditions operators (BCs) for boundaries $\bigcup_i \delta\mathcal{X}_i = \delta\mathcal{X} \subset \mathcal{X}$, and $\mathcal{I}[u]$ is an initial condition operator (IC). Let denote λ the vector that encompasses all parameters of PDE (physical constants, coefficients of f, g, h).

Most common numerical solvers use mesh-based spatial and temporal discretization and produce trajectories sequentially. To obtain the numerical solution for a considered PDE, we have to 1) provide equation-based functional definitions, domain bounds, and vector λ; and 2) select spatial discretization size $M^d \cdot \triangle x = \mathcal{X}$ and temporal discretization size $T \cdot \triangle t = \mathcal{T}$. The second defines the size of spatial coordinates set X and the number of iterations needed for an autoregressive solver. This not only affects the data fidelity level and approximation quality but also the required memory and computation demands. Let us denote a produced solution field at time step $t = i \cdot \triangle t$ for a given λ_j as $x_{ji} = \{\hat{u}(x,t) | x \in X\}$, then the solver produces one-by-one a trajectory:

$$\lambda_j : [x_{j,0} \rightarrow x_{j,1} \rightarrow \cdots \rightarrow x_{j,T}] =: x_j$$

The deep surrogate model u_θ (with weights θ) approximates the PDE solution $u(x,t)$, thereby aiming to substitute a numerical solver. There are different types of surrogate architectures. It can be designed to predict the solution directly, i.e., $u_\theta(X,t) = \hat{x}_t$, or autoregressively, i.e., $u_\theta(x_t) = \hat{x}_{t+\triangle t}$. As we mentioned in Section 1, the surrogate can be multi-parametric: $u_\theta(X,t,\lambda_j) = \hat{x}_j$. In the scope of this paper, we consider multi-parametric direct surrogates; details are provided in Section 4.

2.2 The Melissa DL framework

Melissa DL [28, 29, 38] is an HPC framework designed for deep learning tasks where the NN is trained with simulation data. It consists of three elements: *clients*, a *server*, and a *launcher* (for details, see Appendix A). By default the server uniformly samples input parameters λ for each of S clients across the input parameter space Λ. Each client then runs the solver on the provided inputs and streams the data (timesteps) to the server. The launcher only submits a subset of all clients based on allocated resources. This enables the server to dynamically select new input parameters for *pending'* clients, which we refer to as *global steering* (hereafter, simply *steering*). It is key to unlocking data-efficient surrogate training with active learning methods.

3 Active learning steering of data creation for on-line surrogate training

Active learning is a possible solution for data-efficient surrogate training, as it can help to reduce the number of simulations to execute while maintaining the surrogate quality. Generally, AL's goal is to improve NN training by choosing the most informative examples, based on an acquisition function and a query method, to be labeled by an oracle (or a human) and given to the NN. In our context, "labeling by an oracle" is analogous to "executing a solver" and "choosing examples" — to "sampling solver inputs λ".

However, classical AL methods are not adapted to our on-line training context. Extra computational and memory costs imposed by well-known AL techniques are highly undesirable. First, the input parameters choice decision has to be fast not to pause the surrogate training process as the priority is to keep GPUs busy. Second, the incurred extra memory footprint should be limited to avoid disk storage to keep on-line training efficient.

To develop AL methods for on-line training, we take inspiration from methods proposed for data-free PINNs [12, 25, 43] and extend our previously proposed method called *Breed* [14]. In our supervised setting, instead of choosing collocation points, we have to choose input parameters λ and run an autoregressive solver. In our compute-constrained setting, we aim to use only per-sample loss values as it does not require any extra computation. In our memory-constrained setting, we are not able to recompute loss values for all training points but instead have to use "outdated" loss values, i.e. loss values obtained from the NN at anterior learning steps. Additionally, instead of having a pool of points to choose from, we adaptively sample new points based on previous points loss statistics, inspired by Population Monte Carlo algorithms [9]. We detail the proposed method, *Breed*, in the following.

3.1 Loss-deviation based acquisition metric

We want to define training sample informativeness through NN loss: the higher the loss, the higher the impact on NN training. At iteration i of the fixed-state NN u_{θ_i}, there is an underlying probability distribution of NN failure $\mathcal{L}_{\theta_i}$ over input domain Λ, which we choose to represent through the self-normalized loss function $L(\cdot,\cdot)$:

$$\mathcal{L}_{\theta_i}(\lambda_{j'}) \approx \frac{\sum_{t \in \mathcal{T}} L(u_{\theta_i}(X,t,\lambda_{j'}), x_{j't})}{\sum_{\lambda_j \in \Lambda} \sum_{t \in \mathcal{T}} L(u_{\theta_i}(X,t,\lambda), x_{jt})},$$

where x_{jt} is the t-th timestep of a trajectory produced by a solver with an input parameter λ_j, as defined before. However we cannot calculate the "actual" per-sample loss values (corresponding to NN state θ_i) for neither the whole domain nor approximate them empirically with all the training points. Using per-sample loss values from NN iterations before i is not possible either, as the values are simply not comparable. Hence, we propose to approximate it with *per-sample loss deviation statistics* assuming that the higher the per-sample loss deviation from an average batch loss, the higher the loss. With this metric points from different batches, hence, different NN states, are comparable.

Before running the framework, we define the computational budget by choosing a number of total simulations runs S, hence, creating a set of input parameters $\Lambda_J = \{\lambda_1, \dots, \lambda_j, \dots \lambda_S\}$. At any iteration i', there is a set of inputs $\Lambda_J^{(i')} \subseteq \Lambda_J$ of size S_{done} for which the clients have computed the full trajectories and all points have been seen by the NN.

Let $l_{jt}^{(i)} = L_{\theta_i}(x_{j,t})$ denote per-sample loss. As a sample can potentially be seen by the NN across several batches b_i before iteration i', we denote the set of these batches indexes $I_{jt} := \{i | x_{j,t} \in b_i\} \subset [0 : i']$. Then:

$$\Lambda_J^{(i')} = \{\lambda_j | \exists l_{jt}^{(i)}, \forall i \in I_{jt}, t = [0 : T]\}$$

We compute and store batch-loss mean $\mu(l^{(i)})$ and standard deviation $\sigma(l^{(i)})$, where $l^{(i)} = \{l_{jt}^{(i)} | x_{j,t} \in b_i\}$. Then for any $\lambda_j \in \Lambda_J^{i'}$, we calculate *deviation values* $\delta_{jt} = \{\delta_{jt}^{(i)} | i \in I_{jt}\}$ defined as:

$$\delta_{jt}^{(i)} = \frac{\max(l_{jt}^{(i)} - \mu(l^{(i)}), 0)}{\sigma(l^{(i)})} \tag{4}$$

We then average across timesteps:

$$\hat{\mathcal{L}}_{\theta_{i'}}(\lambda_j) = Q_j := Q(\delta_{jt}) = \frac{1}{T} \sum_{t=1:T} \delta_{jt} = \tag{5}$$

$$= \frac{1}{T} \sum_{t=1:T} \frac{1}{|I_{jt}|} \sum_{i \in I_{jt}} \delta_{jt}^{(i)} \tag{6}$$

Not to store all the values, we iteratively update the statistic δ_{jt} upon the availability of new values.

3.2 Adaptive Multiple Importance Sampling

Instead of choosing points from a pool or a dataset, we want to sample new points according to progressing $\mathcal{L}_\theta$. We propose to use an Adaptive Multiple Importance Sampling (AMIS) algorithm, inspired by the Population Monte Carlo (PMC) algorithm and previously presented in off-line context [14]. An Importance Sampling (IS) goal is to build a proposal probability distribution q, which is easy to sample from, to approximate an unknown target distribution π, which can be evaluated up to a normalizing constant.

In PMC, the proposal is built iteratively. At iteration i $q^{(i)}$ is a mixture (population) of N proposals: $q^{(i)} = \sum_{n=1:N} q_n^{(i)}(\cdot|\mu_n^{(i)}, \Sigma)$. The initial locations $\mu_n^{(0)}$ are given or chosen randomly and $\Sigma = \sigma \mathbb{I}_d$ is a hyperparameter. Next, one random value is sampled from each proposal, and an importance weight is calculated:

$$x_n^{(i)} \sim q_n^{(i)}(\cdot|\mu_n^{(i)}, \sigma) \tag{7}$$

$$w_n^{(i)} = \frac{\pi(x_n^{(i)})}{q_n^{(i)}(x_n^{(i)})} \tag{8}$$

Then a multinomial distribution with weights $\{w_n^{(i)}\}_{n=1:N}$ is trialed N times, i.e., we resample $\{x_n^{(i)}\}_{n=1:N}$ with replacements and obtain $\{x_{n_i}^{(i)}\}_{n_i \in \{1:N\}, i=1:N}$. The resampled values are used as new location parameters $\mu_n^{(i+1)}$.

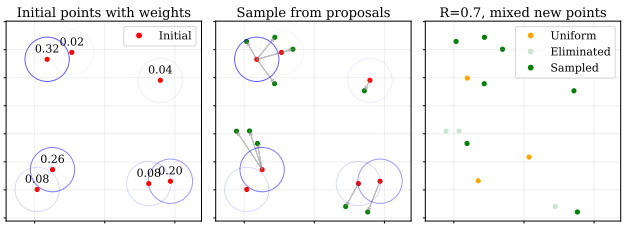


Figure 1: A visual presentation of the sampling algorithm starting with $N = 7$ initial locations, that are next weighted to build the Gaussian mixture proposal and sample from this distribution $K = 10$ new samples. Last, 30% of these points are discarded and substituted with uniform points to maintain exploration capabilities.

In our context, the target is the approximated distribution $\hat{\mathcal{L}}_\theta$. As obtaining $x_n^{(i)}$ at each PMC iteration for the same $\hat{\mathcal{L}}_\theta$ is not feasible — NN training is considerably faster than the creation of the data — we perform only one PMC iteration. We build a proposal once every P NN iterations. Triggering resampling according to metrics such as Effective Sample Size and/or Entropy is left for future work.

At NN iteration $i = 0$, the whole set Λ_J is sampled uniformly. At resampling iteration s, $i = s \cdot P$ (Figure 1), we have a subset of simulations waiting for execution, i.e. $\tilde{\Lambda}_J^{(i)} := \Lambda_J \setminus \Lambda_J^{(i)}$ of size $K = S - S_{\text{done}}$. We want to substitute the corresponding input parameters by sampling K new points. To build a proposal $q^{(s)}$, we use a population of window size $N \leq S_{\text{done}}$, i.e., last $\lambda_j \in \Lambda_J^{(i)}$ in order of Q_j value updates, for which we keep the notation $\Lambda_J^{(i)}$. Here, the points λ_j are the initial locations of the proposal, and Q_j are target distribution evaluations. Then importance weights[1] are:

$$w_{j'} = \frac{Q_{j'}}{\frac{1}{N}\sum_j Q_j} \quad \forall \lambda'_j, \lambda_j \in \Lambda_J^{(i)} \tag{9}$$

and we resample $k = 1:K$ locations to build the proposal:

$$\lambda_{j_k} \sim \text{Mult}(\Lambda_J^{(i)}, w_j, K) \tag{10}$$

$$q^{(s)}(\cdot) = \sum_k q_k^{(s)}(\cdot) = \sum_k \text{Gauss}(\cdot|\lambda_{j_k}, \sigma). \tag{11}$$

[1]Normally, we should divide by a proposal likelihood, but in experiments, we have not noticed if division affects the quality, so it is omitted in this paper.

Finally, we resample new input parameters:

$$\tilde{\Lambda}_J^{(s)} \leftarrow \{\lambda_k \sim q_k^{(s)}(\cdot)\}_k \tag{12}$$

In our implementation, the complexity of one iteration is $O(K)$, though it can be parallelized. If the point sampled appeared out of bounds, we decrease σ by $3e^{-1}$ for its proposal member and sample again. We do it at most five times, and otherwise, the location is left the same. The decreased σ is passed to this proposal member. We use `MultivariateNormal` class from `Pytorch`.

The IS algorithms are known to suffer from mode collapse and underexploration. To tackle this issue and balance a training set, we create *a mixture distribution*: $r^{(s)} \cdot q^{(s)}(\cdot) + (1 - r^{(s)}) \cdot \mathcal{U}(\Lambda)$. In our implementation (Eq. (12)), inputs are substituted with uniform points with probability $r^{(s)}$. *The concentrate-explore value* $r^{(s)} \in [0, 1]$ changes as: $r^{(s)} = \max\left(s \cdot \frac{r_e - r_s}{r_c}, r_e\right)$. The triplet (r_s, r_e, r_c) is a hyperparameter.

3.3 HPC implementation details

We expand the Melissa DL server with the steering mechanism (Figure 2) to apply Breed. The resampling is triggered by the server periodically based on the NN training iteration. Firstly, the server acquires a consistent view of the launcher's job submissions. Secondly, it identifies the simulations that have not yet been submitted for resampling the inputs and, finally, starts the resampling algorithm.

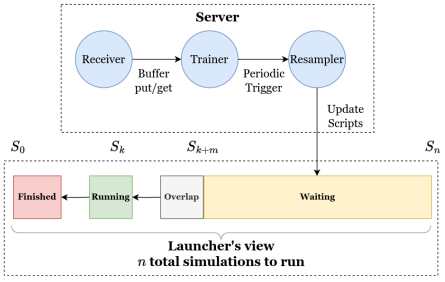


Figure 2: The server's communication with the launcher for the input parameters update mechanism. The number of simulations to run is defined by the budget n.

The Melissa launcher has a limit m on the maximum number of jobs allowed to run simultaneously, determined by the available resources. Assuming the trigger is invoked while currently running or submitted simulation S_k, where k is the highest simulation ID observed from the launcher's perspective. The exact start time of the next simulations from S_{k+1} to S_{k+m-1} cannot be determined due to the inherent uncertainty of the batch scheduler. Therefore, the server always chooses S_{k+m} as the starting point and thus avoids inconsistencies that lead to resampling the parameters that may have been already submitted.

The primary limitation of this approach lies in selecting the appropriate trigger period. For instance, if the period is too frequent, a resampled generation might never execute with the same parameters as it is likely to be overwritten multiple times. Consequently, this value is left to the user's discretion, taking into account the execution speeds of both the solvers and the training process.

4 Experimental study

We experiment with a 2D Heat PDE solver called HeatPDE (Appendix B.1), focusing on the analysis of the method's hyperparameters space and its performance with different NN sizes. We compare our method to an on-line training of a surrogate where input parameters are sampled uniformly, which we refer to as *Random*. We chose the heat PDE case due to its relatively low computational demands and ease of interpretability.

The surrogate is trained to directly predict the discretized temperature field: $u_\theta(\lambda, t) = \hat{u}_\lambda(x, t)$, where $\lambda = [T_0, T_1, T_2, T_3, T_4] \in [100, 500]^5 \subset \Lambda$ are the initial and four boundary temperatures and $t \in [0, 1, \ldots, 100]$. We choose a multilayer perceptron with an input layer of 6 neurons, L hidden layers of H neurons with ReLU activation, and an output of $M^2 = 64^2$ neurons. It is trained using Adam optimizer with a learning rate of $1e^{-3}$ and batch size $B = 128$. The simulations run budget is $S = 800$, and the pre-created fixed validation set has 200 full-trajectory simulations with parameters generated from a quasi-uniform Halton sequence.

4.1 Study description

We conduct two systematic studies: across different model configurations (Figure 3a) and across different Breed hyperparameters (Figure 3b). All other configuration values are fixed for fair comparison (see appendix Table 1). We vary:

(1) The hidden size H and number of layers L of the fully connected NN;
(2) The 3 parameters associated with sampling: window size N and period P (implementation), width σ (PMC);
(3) The 3 parameters associated with r value: (r_s, r_e, r_c).

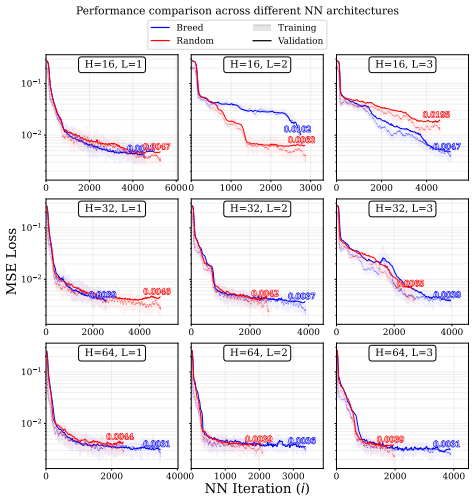
The model configuration affects its expressivity and capability to capture more complex data. It directly connects to overfitting and underfitting phenomena, which in an on-line setting are specifically important. We study values $H = [16, 32, 64]$ and $L = [1, 2, 3]$, run two experiments, with Random and Breed steering.

The window size defines the size of the proposal population, which might affect distribution approximation: smaller values can make it "unstable" while bigger values can make it "outdated". We study values $[50, 600, 1000]$.

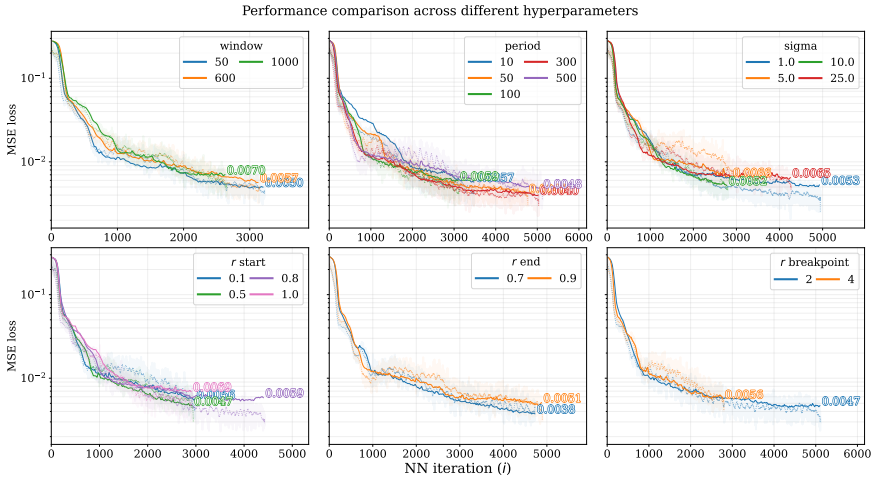
The period affects the computational load — how often we perform resampling . It can also affect the distribution approximation quality as Breed is trying to follow a dynamically changing target $\mathcal{L}_{\theta_i}$. Currently, the period is static, but we expect to extend it to an adaptive trigger that uses the usual MCMC modeling metrics, e.g., effective size and expected improvement. We study values $[10, 50, 100, 300, 500]$.

The tuning of width σ is a known issue in PMC algorithms. Smaller values might make the sampling too "myopic" while bigger values might make it not concentrative enough. Finding a golden middle is challenging. We study values $[1.0, 5.0, 10.0, 25.0]$.

The biggest tuning burden is created by r-value, as it is specific to the problem, the model, and parameter P. However, this mixing ratio was the simplest technique for the exploitation-exploration dilemma, which also appears in MCMC modeling and reinforcement learning. We chose a "linear-constant" change scheme based on heuristics from our previous work, where we noticed that a



(a) Comparative study of models: each plot represents a run with varying fully connected NN configurations, i.e., hidden layer size H (different rows) and number of layers L (different columns).



(b) Comparative study of Breed: each plot represents a run with one varying hyperparameter value, while every other is fixed, including model configuration.

Figure 3: Experimental study over hyperparameters. The changing parameter is indicated in each legend box. The training curve is averaged with a moving window of 40 iterations (dotted line) for visibility. The Y-axis is a logarithmic scale, and it is shared across all plots. Values presented near the curves are the last validation loss values.

"warming up" period was needed to prevent convergence destabilization. In future work, our main goal is to adopt adaptive solutions. We study values $r_s = [0.1, 0.5, 0.8, 1.0]$, $r_e = [0.7, 0.9]$, $r_c = [2, 4]$.

4.2 Results discussion

While overall Breed performance is not clearly better than Random, which can be explained by HeatPDE case simplicity, we see a specific pattern. Given higher expressivity to the model, Random experiments tend to show overfitting, which is especially noticeable for $H = 16$, $L = 3$, while *Breed training and validation curves stay close* [2] (Figure 3a).

We see overfitting for some hyperparameter values (Figure 3b), i.e., high r_s and low σ. As for the convergence, we observe that higher window sizes and lower periods tend to make training divergent at the beginning, which affects further iterations. Within the r_s study, we see that value 0.5 also shows divergence at earlier iterations, but at later ones, converges faster, and its training loss is higher than validation loss, which is a good sign of generalization abilities. Consequently, r_e and r_c affect the training as well.

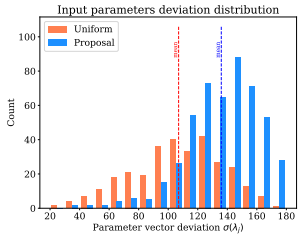
Apart from performance analysis, we conducted explorative analysis across training statistics. Our central insight is that the conditional distribution of input parameters created overall for the run is clearly shifted when we use Breed (Figure 4). We calculated a per input vector deviation, which represents how large is the difference between the temperatures $T_{0:5}$, and built a histogram. In Figure 4b, we compare the fixed configuration run with the Random and Breed methods. The mean of the latter distribution is shifted toward higher deviation values. It means that *Breed focuses sampling in Λ regions where temperatures are more diverse.* It makes sense for the HeatPDE case, as diverse temperatures bring more dynamicity to the trajectory, which should be harder to learn. To see this phenomenon clearly, we compared the histograms for the uniform and proposal samples for one Breed run in Figure 4a.

Additionally, we calculated the correlation coefficients between the NN iteration, per-sample and batch losses, and our proposed deviation loss metric Q_j. We noticed that our metric has no correlation to the NN iteration (-0.02) but has a positive correlation (0.27) with per-sample loss, while batch loss and sample loss correlate with the NN iteration (0.4, 0.31). It means, *we constructed a metric that is comparable between NN iterations and partially representative of per-sample loss.* See a visual representation of the correlation matrix in appendix Figure 6.

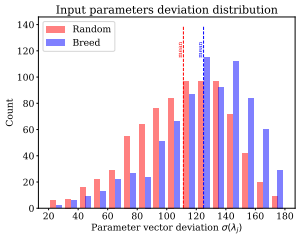
5 Related work

The question of active learning originated in the context of training NNs with a finite dataset [39]. The goal is to "select a small subset of unlabeled samples from a large pool of data for labeling and training, while achieving comparable generalization performance to learning on the entire dataset" [6]. Active learning in that context relies on two main criteria. The first one is based on *uncertainty*, choosing samples that the neural models are most uncertain about. The second is based on *diversity*, selecting samples bringing diversity in the feature space compared to the already labeled ones [6]. It is tackled in various ways: measuring samples uncertainty by approximating training dynamics [24, 40], calculating samples influence [19],

[2] In Melissa, a training thread may operate more frequently than a receiving thread. It can result in more training iterations independent of the run configuration, which we observe in figures.



(a) Breed run: uniform (orange) against proposal (light blue)



(b) Random run (red) against Breed run (blue)

Figure 4: Input parameter deviation histogram obtained from one run of 800 input parameters. On the left (a), comparison per source of point (whether uniform or proposal) for one Breed run; on the right (b), comparison of two runs (Random and Breed). The mean is plotted to better see the distribution shift.

selecting representative subsets with use of gradients [17, 20, 21], with also extensions to data streams [7].

In the AI4Science domain, active learning has recently seen a surge of interest to improve PINNs training. PINNs are trained by choosing collocation points. Uniform sampling is the standard approach, but alternative adaptive sampling strategies have been proposed in an attempt to improve noticeably hard-to-train networks. Approach ranges from re-weighting sample importance [32, 44], creating a training subset based on a probability distribution calculated from the normalized losses [43], or retaining points whose loss is higher than average and resampling the remaining ones uniformly for each iteration [12]. [25] proposes to use values calculated with the Neural Tangent Kernel (NTK) instead of the loss, giving a more precise insight into the influence of each sample. The compute overhead is shown to be compensated by the gain on the convergence speed.

The paper [30] proposes to apply classical active learning algorithms to multi-parametric surrogate training. Their approach is fully off-line, using active learning with lightweight NNs to gather metrics to build a static data set, which is next used for training the full-featured surrogate. Simulation-based inference (SBI) [11] trains a NN from simulations to solve an inverse problem. The NN often relies on normalizing flows to learn the posterior distribution, which can be used in turn to select the input parameters of the next set of simulations to run. We can also mention a former work [4] that addressed a similar inference issue from simulation runs, using an LSTM neural architecture with the particularity of experiments at large scale on more than thousands of CPU nodes. Another work proposed an HPC framework that relies on machine learning to adaptively select the members of an ensemble to run to accelerate the parameter space exploration according to some given objective [42].

Our proposed sampling method is inspired by a PMC algorithm. To our knowledge, PMC has not been used for active learning. More advanced PMC versions exist that instead of sampling within the vicinity of high probability points with a fixed standard deviation, exploit geometric information of the target to adapt the location and scale parameters of those proposals [15, 16] or use normalizing flows instead of a Gaussian proposal [34]. Deploying such an algorithm in our context is left for future work.

6 Conclusion

In this paper, we introduced an active learning method for data-efficient on-line training of deep surrogates using the Melissa framework. Our approach steers data creation by leveraging loss statistics and Importance Sampling: it guides the solvers to compute trajectories with input parameters in hard-to-learn regions. Preliminary results with the heat equation showed Breed's potential to improve NN generalization ability without computational overhead, as well as an interpretable choice of points. Future work will focus on conducting experiments for larger-scale and more complex dynamic PDEs, refining the method with advanced sampling techniques, and reducing the set of hyperparameters by developing self-adaptive techniques.

Acknowledgements

This work was supported by project Exa-DoST, NumPEx PEPR program, France 2030 state grant reference ANR-22-EXNU-0004. This work was granted access to the HPC resources of IDRIS under the allocations AD010610366R3 attributed by GENCI (Grand Equipement National de Calcul Intensif), and benefited from access to the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations. We are thankful to Quentin Guilloteau and Fernando Ayats for their help with setting the environment for reproducible experiments.

References

[1] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J. Ballard, Joshua Bambrick, Sebastian W. Bodenstein, David A. Evans, Chia-Chun Hung, Michael O'Neill, David Reiman, Kathryn Tunyasuvunakool, Zachary Wu, Akvilė Žemgulytė, Eirini Arvaniti, Charles Beattie, Ottavia Bertolli, Alex Bridgland, Alexey Cherepanov, Miles Congreve, Alexander I. Cowen-Rivers, Andrew Cowie, Michael Figurnov, Fabian B. Fuchs, Hannah Gladman, Rishub Jain, Yousuf A. Khan, Caroline M. R. Low, Kuba Perlin, Anna Potapenko, Pascal Savy, Sukhdeep Singh, Adrian Stecula, Ashok Thillaisundaram, Catherine Tong, Sergei Yakneen, Ellen D. Zhong, Michal Zielinski, Augustin Žídek, Victor Bapst, Pushmeet Kohli, Max Jaderberg, Demis Hassabis, and John M. Jumper. 2024. Accurate Structure Prediction of Biomolecular Interactions with AlphaFold 3. *Nature* 630, 8016 (June 2024), 493–500. https://doi.org/10.1038/s41586-024-07487-w

[2] Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. 2024. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics* 6, 5 (May 2024), 320–328. https://doi.org/10.1038/s42254-024-00712-5 arXiv:2309.15325 [cs.LG]

[3] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, et al. 2013. Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science: Second International Conference, CLOSER 2012, Porto, Portugal, April 18-21, 2012. Revised Selected Papers 2*. Springer, 3–20.

[4] Atılım Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip Torr, Victor Lee, Kyle Cranmer, Prabhat, and Frank Wood. 2019. Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale. (2019). https://doi.org/10.1145/3295500.3356180 arXiv:1907.03382 Publisher: IEEE Computer Society.

[5] Cristian Bodnar, Wessel Bruinsma, Ana Lucic, Megan Stanley, Johannes Brandstetter, Patrick Garvan, Maik Riechert, Jonathan Weyn, Haiyu Dong, Anna Vaughan, Jayesh Gupta, Kit Thambiratnam, Alex Archibald, Elizabeth Heider, Max Welling, Richard Turner, and Paris Perdikaris. 2024. *Aurora: A Foundation Model of the Atmosphere.* Technical Report MSR-TR-2024-16. Microsoft Research

AI for Science. https://www.microsoft.com/en-us/research/publication/aurora-a-foundation-model-of-the-atmosphere/

[6] Dake Bu, Wei Huang, Taiji Suzuki, Ji Cheng, Qingfu Zhang, Zhiqiang Xu, and Hau-San Wong. 2024. Provably Neural Active Learning Succeeds via Prioritizing Perplexing Samples. In *Forty-First International Conference on Machine Learning*.

[7] Davide Cacciarelli and Murat Kulahci. 2024. Active Learning for Data Streams: A Survey. *Machine Learning* 113, 1 (Jan. 2024), 185–239. https://doi.org/10.1007/s10994-023-06454-2

[8] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. 2005. A batch scheduler with high level components. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, Vol. 2. 776–783 Vol. 2. https://doi.org/10.1109/CCGRID.2005.1558641

[9] Olivier Cappé, Arnaud Guillin, Jean-Michel Marin, and Christian Robert. 2004. Population Monte Carlo. *Journal of Computational and Graphical Statistics* 13, 4 (2004), 907–929. https://hal.science/hal-01337419

[10] Ze Cheng, Zhongkai Hao, Xiaoqiang Wang, Jianing Huang, Youjia Wu, Xudan Liu, Yiru Zhao, Songming Liu, and Hang Su. 2024. Reference Neural Operators: Learning the Smooth Dependence of Solutions of PDEs on Geometric Deformations. *arXiv e-prints*, Article arXiv:2405.17509 (May 2024), arXiv:2405.17509 pages. https://doi.org/10.48550/arXiv.2405.17509 arXiv:2405.17509 [cs.LG]

[11] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. 2020. The Frontier of Simulation-Based Inference. *Proceedings of the National Academy of Sciences* 117, 48 (Dec. 2020), 30055–30062. https://doi.org/10.1073/pnas.1912789117

[12] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. 2023. Mitigating Propagation Failures in Physics-informed Neural Networks using Retain-Resample-Release (R3) Sampling. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 7264–7302. https://proceedings.mlr.press/v202/daw23a.html ISSN: 2640-3498.

[13] Eelco Dolstra, Merijn De Jonge, Eelco Visser, et al. 2004. Nix: A Safe and Policy-Free System for Software Deployment.. In *LISA*, Vol. 4. 79–92.

[14] Sofya Dymchenko and Bruno Raffin. 2023. 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*. New Orleans, United States, 1–5. https://hal.science/hal-04305233

[15] Víctor Elvira and Émilie Chouzenoux. 2022. Optimized Population Monte Carlo. *IEEE Transactions on Signal Processing* 70 (2022), 2489–2501. https://doi.org/10.1109/TSP.2022.3172619 arXiv:2204.06891 [stat]

[16] Víctor Elvira, Émilie Chouzenoux, Ömer Deniz Akyildiz, and Luca Martino. 2023. Gradient-Based Adaptive Importance Samplers. *Journal of the Franklin Institute* 360, 13 (Sept. 2023), 9490–9514. https://doi.org/10.1016/j.jfranklin.2023.06.041

[17] Mohsen Fayyaz, Ehsan Aghazadeh, Ali Modarressi, Mohammad Taher Pilehvar, Yadollah Yaghoobzadeh, and Samira Ebrahimi Kahou. 2022. BERT on a Data Diet: Finding Important Examples by Gradient-Based Pruning. In *NeurIPS*.

[18] Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. 2024. Poseidon: Efficient Foundation Models for PDEs. https://doi.org/10.48550/arXiv.2405.19101 arXiv:2405.19101 [cs]

[19] Karthikeyan K and Anders Søgaard. 2021. Revisiting Methods for Finding Influential Examples. (2021). https://doi.org/10.48550/arxiv.2111.04683 arXiv:2111.04683

[20] Angelos Katharopoulos and François Fleuret. 2019. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. https://doi.org/10.48550/arXiv.1803.00942 arXiv:1803.00942 [cs]

[21] Krishnateja Killamsetty, Guttu Sai Abhishek, Ganesh Ramakrishnan, Alexandre V Evfimievski, Lucian Popa, and Rishabh Iyer. 2022. AUTOMATA : Gradient Based Data Subset Selection for Compute-Efficient Hyper-parameter Tuning. In *Advances in Neural Information Processing Systems*.

[22] Dmitrii Kochkov, Janni Yuval, Ian Langmore, Peter Norgaard, Jamie Smith, Griffin Mooers, Milan Klöwer, James Lottes, Stephan Rasp, Peter Düben, Sam Hatfield, Peter Battaglia, Alvaro Sanchez-Gonzalez, Matthew Willson, Michael P. Brenner, and Stephan Hoyer. 2024. Neural General Circulation Models for Weather and Climate. *Nature* (July 2024). https://doi.org/10.1038/s41586-024-07744-y

[23] Georg Kohl, Li-Wei Chen, and Nils Thuerey. 2023. Benchmarking Autoregressive Conditional Diffusion Models for Turbulent Flow Simulation. *arXiv e-prints*, Article arXiv:2309.01745 (Sept. 2023), arXiv:2309.01745 pages. https://doi.org/10.48550/arXiv.2309.01745 arXiv:2309.01745 [cs.LG]

[24] Seong Min Kye, Kwanghee Choi, and Buru Chang. 2022. TiDAL: Learning Training Dynamics for Active Learning. (2022). https://doi.org/10.48550/ARXIV.2210.06788 Publisher: arXiv Version Number: 1.

[25] Gregory Kang Ruey Lau, Apivich Hemachandra, See-Kiong Ng, and Bryan Kian Hsiang Low. 2023. PINNACLE: PINN Adaptive ColLocation and Experimental Points Selection. In *The Twelfth International Conference on Learning Representations*.

[26] Alexander Lavin, Hector Zenil, Brooks Paige, David Krakauer, Justin Gottschlich, Tim Mattson, Anima Anandkumar, Sanjay Choudry, Kamil Rocki, Atılım Güneş Baydin, Carina Prunkl, Brooks Paige, Olexandr Isayev, Erik Peterson, Peter L. McMahon, Jakob Macke, Kyle Cranmer, Jiaxin Zhang, Haruko Wainwright, Adi Hanuka, Manuela Veloso, Samuel Assefa, Stephan Zheng, and Avi Pfeffer. 2021. Simulation Intelligence: Towards a New Generation of Scientific Methods. (2021). arXiv:2112.03235 http://arxiv.org/abs/2112.03235

[27] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. 2021. Fourier Neural Operator for Parametric Partial Differential Equations. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=c8P9NQVtmnO

[28] Lucas Meyer, Marc Schouler, Robert Alexander Caulk, Alejandro Ribés, and Bruno Raffin. 2023. High Throughput Training of Deep Surrogates from Large Ensemble Runs. In *SC 2023 - The International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM, Denver, CO, United States, 1–14. https://doi.org/10.1145/3581784.3607083

[29] Lucas Meyer, Marc Schouler, Robert Alexander Caulk, Alejandro Ribés, and Bruno Raffin. 2023. Training Deep Surrogate Models with Large Scale Online Learning. In *ICML 2023 - International Conference on Machine Learning*. 1–17. https://hal.science/hal-04102400

[30] Daniel Musekamp, Marimuthu Kalimuthu, David Holzmüller, Makoto Takamoto, and Mathias Niepert. 2024. Active Learning for Neural PDE Solvers. arXiv:2408.01536 [cs]

[31] F Mölder, KP Jablonski, B Letcher, MB Hall, CH Tomkins-Tinch, V Sochat, J Forster, S Lee, SO Twardziok, A Kanitz, A Wilm, M Holtgrewe, S Rahmann, S Nahnsen, and J Köster. 2021. Sustainable data analysis with Snakemake [version 2; peer review: 2 approved]. *F1000Research* 10, 33 (2021). https://doi.org/10.12688/f1000research.29032.2

[32] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. 2021. Efficient training of physics-informed neural networks via importance sampling. 36, 8 (2021), 962–977. https://doi.org/10.1111/mice.12685 arXiv:2104.12325 [cs, math]

[33] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. 2023. ClimaX: A foundation model for weather and climate. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 25904–25938. https://proceedings.mlr.press/v202/nguyen23a.html

[34] Soumyasundar Pal, Antonios Valkanas, and Mark Coates. 2023. Population Monte Carlo with Normalizing Flow. https://doi.org/10.48550/arXiv.2312.03857 arXiv:2312.03857 [stat].

[35] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. 2021. Learning Mesh-Based Simulation with Graph Networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=roNqYL0_XP

[36] M. Raissi, P. Perdikaris, and G.E. Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

[37] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. 2020. A Survey of Deep Active Learning. *arXiv e-prints*, Article arXiv:2009.00236 (Aug. 2020), arXiv:2009.00236 pages. https://doi.org/10.48550/arXiv.2009.00236 arXiv:2009.00236 [cs.LG]

[38] 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. 2023. Melissa: coordinating large-scale ensemble runs for deep learning and sensitivity analyses. *Journal of Open Source Software* 8, 86 (2023), 5291. https://doi.org/10.21105/joss.05291

[39] Rinyoichi Takezoe, Xu Liu, Shunan Mao, Marco Tianyu Chen, Zhanpeng Feng, Shiliang Zhang, and Xiaoyu Wang. 2023. Deep Active Learning for Computer Vision: Past and Future. *APSIPA Transactions on Signal and Information Processing* 12, 1 (2023), –. https://doi.org/10.1561/116.00000057

[40] Haonan Wang, Wei Huang, Ziwei Wu, Hanghang Tong, Andrew J. Margenot, and Jingrui He. 2022. Deep Active Learning by Leveraging Training Dynamics. https://openreview.net/forum?id=aJ5xc1QB7EX

[41] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. 2023. An Expert's Guide to Training Physics-informed Neural Networks. https://doi.org/10.48550/arXiv.2308.08468 arXiv:2308.08468 [physics].

[42] Logan Ward, Ganesh Sivaraman, J. Gregory Pauloski, Yadu Babuji, Ryan Chard, Naveen Dandu, Paul C. Redfern, Rajeev S. Assary, Kyle Chard, Larry A. Curtiss, Rajeev Thakur, and Ian Foster. 2021. Colmena: Scalable Machine-Learning-Based Steering of Ensemble Simulations for High Performance Computing. (2021), 9–20. https://doi.org/10.1109/MLHPC54614.2021.00007

[43] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. 2022. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. https://doi.org/10.48550/arXiv.2207.10289 arXiv:2207.10289 [physics]

[44] Zijiang Yang, Zhongwei Qiu, and Dongmei Fu. 2022. DMIS: Dynamic Mesh-based Importance Sampling for Training Physics-Informed Neural Networks. https://doi.org/10.48550/arXiv.2211.13944 arXiv:2211.13944 [cs, math]

A Melissa DL Architecture

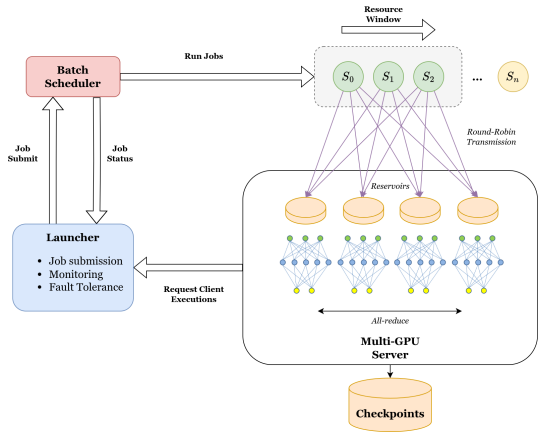


Figure 5: The Melissa framework architecture consisting of *launcher* (orchestrates the process through the cluster's batch scheduler), *clients jobs* (where simulations are executed), and a *server* (trains NN and manages *reservoirs*).

The *launcher* controls the workflow by interacting with the cluster's batch scheduler. It initiates the *server*, a Python-based code using PyTorch for multi-GPU training. The server manages the training process and defines the simulation instances, known as *clients*. The server then sends a request to the launcher for submitting clients based on allocated resources. Once a client starts, it connects dynamically to the server.

The server maintains a memory buffer called the *reservoir*, which goal is to reduce training bias and avoid GPU starvation. Newly received data from the clients are stored in the reservoir, replacing older data randomly. If all reservoir samples are new, client executions are paused temporarily. The server asynchronously creates random batches from the reservoir for NN training, allowing each reservoir sample to be reused multiple times. See [28] for a detailed description of the reservoir algorithm.

B General setup details

B.1 2D HeatPDE with Melissa

The experiments consider *the classical heat equation (HeatPDE)* on a 2D rectangular domain:

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2} \tag{13}$$

$$u((x_1 = 0, x_2), t) = T_1, u((x_1 = L, x_2), t) = T_2 \tag{14}$$

$$u((x_1, x_2 = 0), t) = T_3, u((x_1, x_2 = L), t) = T_4 \tag{15}$$

$$u(x, t = 0) = T_0 \tag{16}$$

where $u(x,t)$ is the field temperature, α is the thermal diffusivity and $[T_0, T_1, T_2, T_3, T_4]$ are the initial and 4 boundary temperatures. The solution is approximated with an in-house solver that implements a finite difference method with an implicit Euler scheme. The temperature field is discretized on a $M \times M$ Cartesian grid and generated for $T = 100$ time steps representing $\triangle t = 0.01$ seconds each. In this study, the thermal diffusivity is fixed to $\alpha = 1m^2.s^{-1}$, and changing this parameter is left for future work. The temperature parameters are the solver input parameters $\Lambda = \mathbb{R}^5$ that we tend to sample, which values we bound to interval $[100, 500]$ K.

The surrogate is trained to directly predict the temperature field. As the initial temperature field is described by the input parameters vector exhaustively, the NN input is not the field itself but just the vector and timestep: $u_\theta(\lambda, t) = \hat{u}_\lambda(x, t)$. The NN architecture is a multilayer perceptron consisting of an input layer of 6 neurons, L hidden layers of H neurons with ReLU activation, and an output of M^2 neurons. It is trained using Adam optimizer with a learning rate of $1e^{-3}$.

In Melissa, we set the simultaneous job limit to $m = 10$ and the *Reservoir* watermark value to 300, meaning that the NN training does not begin until the buffer contains at least 300 unique samples.

B.2 Experiment orchestration

To facilitate a broad analysis study, we employ `Snakemake`[31], a workflow management system that enables the execution and management of scalable, reproducible analysis studies. In our case, the workflow creates configuration files for Melissa runs across chosen grid. To ensure the reproducibility[3] of our experiments, we utilize `Nix` [13] package manager. The experiments were conducted on the Grid5000 [3] cluster using OAR scheduler [8]. Each Melissa client as well as the server run one 48 processes MPI job, each one scheduled on a 48 core node.

C Experiments additional details

Here we provide the Table 1 with exact hyperparameters used in the study, and 6 is the visualisation of correlation matrix.

Table 1: The fixed hyperparameters details according to varying (*) parameter: (1) the model size is varied, (2) σ or P or N is varied, (3) r_s or r_e or r_c is varied.

	σ	P	N	r_s	r_e	r_c	H	L
Study (1)	10.0	300	200	0.5	0.7	3	*	*
Study (2)	*/5.0	*/200	*/200	0.5	0.9	3	16	1
Study (3)	5.0	200	200	*/0.1	*/1.0	*/5	16	1

[3] Code is available at https://gitlab.inria.fr/melissa/ai4s-sc2024-heatpde.git

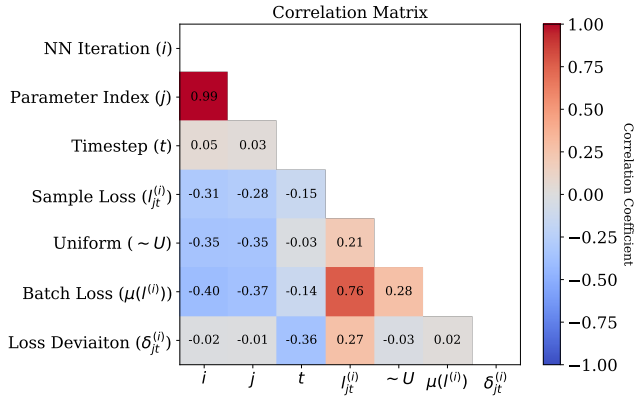


Figure 6: Correlation between per-sample and per-batch dynamics: indicated on the left axis. The upper triangle and diagonal values are omitted for readability. "Uniform" value is an indicator of whether the sample was produced by uniform mixing, "loss deviation" is the proposed metric, which is not dependent on NN iteration but still positively correlates with per-sample loss.