Deep Statistical Solvers

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

This paper introduces Deep Statistical Solvers (DSS), a new class of trainable solvers for optimization problems, arising from system simulations and others. The key idea is to learn a solver that generalizes to a given distribution of problem instances. This is achieved by directly using as loss the objective function of the problem, as opposed to most previous Machine Learning based approaches, which mimic the solutions attained by an existing solver. Though both types of approaches outperform classical solvers with respect to speed for a given accuracy, a distinctive advantage of DSS is that they can be trained without a training set of sample solutions. Focusing on use cases of systems of interacting and interchangeable entities (*e.g.* molecular dynamics, power systems, discretized PDEs), the proposed approach is instantiated within a class of Graph Neural Networks. Under sufficient conditions, we prove that the corresponding set of functions contains approximations to any arbitrary precision of the actual solution of the optimization problem. The proposed approach is experimentally validated on large linear problems, on demonstrating super-generalisation properties; And on AC power grid simulations, on which the predictions of the trained model have a correlation higher than 99.99% with the outputs of the classical Newton-Raphson method (known for its accuracy), while being 2 to 3 orders of magnitude faster.

1 Introduction

In many domains of physics and engineering, Deep Neural Networks (DNNs) have sped up simulations and optimizations by orders of magnitude, replacing some computational bricks based on first principles with data-driven numerical models – see *e.g.*, [1, 2, 3, 4]. However, in general, such data-driven approaches consist in training a *proxy* in a supervised way, to imitate solutions provided by some numerical solver. This is sometimes infeasible due to the high computational cost of existing simulators (*e.g.* in molecular dynamics, car crash simulations, computational fluid dynamics, and power grid simulation). Furthermore, such approaches ignore problem-specific constraints and may end up providing inconsistent solutions, failing to satisfy physical constraints such as energy conservation (which can only be a posteriori checked, see *e.g.* [4]). In order to bypass this weakness, a growing body of work pushes towards an interplay between physics and Machine Learning [5], *e.g.*, incorporating physical knowledge in the loss function during learning [6, 7].

Another important property of natural or artificial systems is that of invariance, a fundamental concept in science, allowing to generalize conclusions drawn from few observations, to whole invariance classes. This work focuses on permutation-invariant problems, which appear in simulations of complex systems of interacting and interchangeable entities [8] (*e.g.,* molecular dynamics, power grids, simulations of partial differential equations (PDEs) with finite elements). Invariance has made its way in machine learning, as illustrated by the success of Convolutional Neural Networks (CNN) [9, 10], and of Graph Neural Networks (GNN) [11, 12]. In particular, implementations of GNNs successfully handle materials dynamics simulations [13], power systems [14], interacting particles

[15] and classical [16] or quantum [17] chemistry. However, all of these works pertain to the *proxy approach* described above.

Our first contribution is to propose, at the interface of optimization and statistics, the Statistical Solver Problem (SSP), a novel formulation for learning to solve a whole class of optimization and system simulation problems. The resulting framework i) directly minimizes the global loss function of the problems during training, thus not requiring any existing solution of the problems at hand, and ii) directly incorporates permutation-invariance in the representation of the problems using a GNN-based architecture, called Deep Statistical Solver (DSS). Our second contribution is to prove that DSS satisfies some Universal Approximation property in the space of SSP solutions. The third contribution is an experimental validation of the approach.

The outline of the paper is the following. Section 2 sets the background, and defines SSPs. Section 3 introduces Deep Statistical Solvers. Section 4 proves the Universal Approximation property for permutation-invariant loss functions (and some additional hypotheses). Section 5 experimentally validates the DSS approach, demonstrating its efficiency w.r.t. state-of-the-art solvers, and unveiling some super-generalization capabilities. Section 6 concludes the paper.

2 Definitions and Problem Statement

This section introduces the context (notations and definitions) and the research goal of this work: The basic problem is, given a network of interacting entities (referred to later as Interaction Graph), to find a state of the network that minimizes a given loss function; From thereon, the main goal of this work is to learn a parameterized mapping that accurately and quickly computes such minimizing state for any Interaction Graph drawn from a given distribution.

2.1 Notations and Definitions

Notations Throughout this paper, for any $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$; Σ_n is the set of permutations of $[n]$; for any $\sigma \in \Sigma_n$, any set Ω and any vector $\mathbf{x} = (x_i)_{i \in [n]} \in \Omega^n$, $\sigma \star \mathbf{x}$ is the vector $(x_{\sigma^{-1}(i)})_{i \in [n]}$; for any $\sigma \in \Sigma_n$ and any matrix $\mathbf{m} = (m_{ij})_{i,j \in [n]} \in \mathcal{M}_n(\Omega)$ (square matrices with elements in Ω), $\sigma \star \mathbf{m}$ is the matrix $(m_{\sigma^{-1}(i)\sigma^{-1}(j)})_{i,j \in [n]}$.

Interaction Graphs We call *Interaction Graph* a system of $n \in \mathbb{N}$ interacting entities, or *nodes*, defined as $\mathbf{G} = (n, \mathbf{A}, \mathbf{B})$, where n is the size of $\mathbf{G}$ (number of nodes), $\mathbf{A} = (A_{ij})_{i,j \in [n]}; A_{ij} \in \mathbb{R}^{d_A}; d_A \geq 1$ represents the interactions between nodes, and $\mathbf{B} = (B_i)_{i \in [n]}; B_i \in \mathbb{R}^{d_B}, d_B \geq 1$ are some local external inputs at each node. Let $\mathcal{G}_{d_A, d_B}$ be the set of all such Interaction Graphs and simply $\mathcal{G}$ when there is no confusion. For any $\sigma \in \Sigma_n$ and any Interaction Graph $\mathbf{G} = (n, \mathbf{A}, \mathbf{B})$, $\sigma \star \mathbf{G}$ denotes the Interaction Graph $(n, \sigma \star \mathbf{A}, \sigma \star \mathbf{B})$.

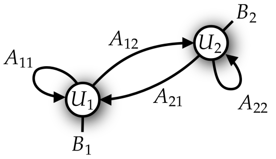

Figure 1: **A sample Interaction Graph** $(2, \mathbf{A}, \mathbf{B})$

Interaction Graphs can also be viewed as "doubly weighted" graphs, *i.e.*, graphs with weights on both the edges (weights A_{ij}) and the nodes (weights B_i), considering that those weights are vectors. For a given $\mathbf{G}$, we will also consider the underlying undirected unweighted graph $\widetilde{\mathbf{G}}$ for which links between nodes i and j exist *iff* either A_{ij} or A_{ji} is non-zero[1]. We will use the notion of neighborhood induced by $\widetilde{\mathbf{G}}$: $j \in \mathcal{N}(i; \mathbf{G})$ *iff* i and j are neighbors in $\widetilde{\mathbf{G}}$ (and $\mathcal{N}^\star(i; \mathbf{G})$ will denote $\mathcal{N}(i; \mathbf{G}) \backslash \{i\}$).

States and Loss Functions Vectors $\mathbf{U} = (U_i)_{i \in [n]}; U_i \in \mathbb{R}^{d_U}, d_U \geq 1$ represent *states* of Interaction Graphs of size n, where U_i is the state of node i. $\mathcal{U}_{d_U}$ denotes the set of all such states ($\mathcal{U}$ when there is no confusion). A *loss function* ℓ is a real-valued function defined on pairs $(\mathbf{U}, \mathbf{G})$, where $\mathbf{U}$ is a state of $\mathbf{G}$ (i.e., of same size).

Permutation invariance and equivariance A loss function on Interaction Graph $\mathbf{G}$ of size n is *permutation-invariant* if for any $\sigma \in \Sigma_n$, $\ell(\sigma \star \mathbf{U}, \sigma \star \mathbf{G}) = \ell(\mathbf{U}, \mathbf{G})$.
A function $\mathcal{F}$ from $\mathcal{G}$ to $\mathcal{U}$, mapping an Interaction Graph $\mathbf{G}$ of size n on one of its possible states $\mathbf{U}$ is *permutation-equivariant* if for any $\sigma \in \Sigma_n$, $\mathcal{F}(\sigma \star \mathbf{G}) = \sigma \star \mathcal{F}(\mathbf{G})$.

[1] A more rigorous definition of the actual underlying graph structure is deferred to Appendix A

2.2 Problem Statement

The Optimization Problem In the remaining of the paper, ℓ is a loss function on Interaction Graphs $\mathbf{G} \in \mathcal{G}$ that is both continuous and permutation-invariant. The elementary question of this work is to solve the following optimization problem for a given Interaction Graph $\mathbf{G}$:

$$\mathbf{U}^\star(\mathbf{G}) = \underset{\mathbf{U}\in\mathcal{U}}{\text{argmin}} \ \ \ell(\mathbf{U}, \mathbf{G}) \tag{1}$$

The Statistical Learning Goal We are not interested in solving problem (1) for just ONE Interaction Graph, but in learning a parameterized *solver*, *i.e.*, a mapping from $\mathcal{G}$ to $\mathcal{U}$, which solves (1) for MANY Interaction Graphs, namely all Interaction Graphs $\mathbf{G}$ sampled from a given distribution $\mathcal{D}$ over $\mathcal{G}$. In particular, $\mathcal{D}$ might cover Interaction Graphs of different sizes. Let us assume additionally that $\mathcal{D}$ and ℓ are such that, for any $\mathbf{G} \in \text{supp}(\mathcal{D})$ (the support of $\mathcal{D}$) there is a unique minimizer $\mathbf{U}^*(\mathbf{G}) \in \mathcal{U}$ of problem (1). The goal of the present work is to learn a single solver that best approximates the mapping $\mathbf{G} \mapsto \mathbf{U}^*(\mathbf{G})$ for all $\mathbf{G}$ in $\text{supp}(\mathcal{D})$. More precisely, assuming a family of solvers $Solver_\theta$ parameterized by $\theta \in \Theta$ (Section 3 will introduce such a parameterized family of solvers, based on Graph Neural Networks), the problem tackled in this paper can be formulated as a *Statistical Solver Problem* (SSP):

$$\text{SSP}(\mathcal{G}, \mathcal{D}, \mathcal{U}, \ell) \begin{cases} \text{Given distribution } \mathcal{D} \text{ on space of Interaction Graphs } \mathcal{G}, \text{ space of states } \mathcal{U}, \\ \text{and loss function } \ell, \text{ solve } \theta^\star = \underset{\theta\in\Theta}{\text{argmin}} \ \ \mathbb{E}_{\mathbf{G}\sim\mathcal{D}} \ \left[\ell\left(Solver_\theta(\mathbf{G}), \mathbf{G}\right)\right] \end{cases} \tag{2}$$

Learning phase In practice, the expectation in (2) will be empirically computed using a finite number of Interaction Graphs sampled from $\mathcal{D}$, by directly minimizing ℓ (*i.e.,* without the need for any $\mathbf{U}^\star$ solution of (1)). The result of this empirical minimization is a parameter $\widehat{\theta}$.

Inference The solver $Solver_{\widehat{\theta}}$ can then be used, at inference time, to compute, for any $\mathbf{G} \in supp(\mathcal{D})$, an approximation of the solution $\mathbf{U}^\star(\mathbf{G})$

$$\widehat{\mathbf{U}}(\mathbf{G}) = Solver_{\widehat{\theta}}(\mathbf{G}) \tag{3}$$

Solving problem (1) has been replaced by a simple and fast inference of the learned model $Solver_{\widehat{\theta}}$ (at the cost of a possibly expensive learning phase).

Discussion The SSP experimented with in Section 5.2 addresses the simulation of a Power Grid, a real-world problem for which the benefits of using the proposed approach becomes clear. Previous work [18] used a "proxy" approach, which consists in learning from known solutions of the problem, provided by a classical solver. The training phase is sketched on Figure 2.a. The drawback of such an approach is the need to gather a huge number of training examples (*i.e.*, solutions of problem (1)), something that is practically infeasible for complex problems: either such solutions are too costly to obtain (*e.g.*, in car crash simulations), or there is no provably optimal solution (*e.g.*, in molecular dynamics simulations). In contrast, since the proposed approach directly trains $Solver_\theta$ by minimizing the loss ℓ (Figure 2.b), no such examples are needed.

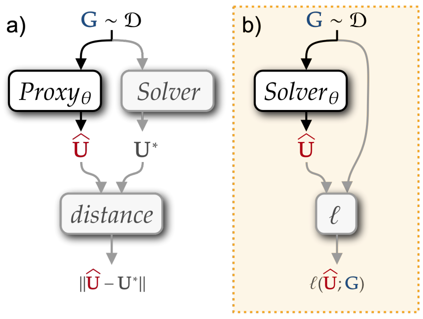

Figure 2: **Proxy approach (a)** ***vs.*** **DSS (b)**

3 Deep Statistical Solver Architecture

In this section, we introduce the class of Graph Neural Networks (GNNs) that will serve as DSSs. The intuition behind this choice comes from the following property (proof in Appendix B.2):

Property 1. *If the loss function ℓ is permutation-invariant and if for any $\mathbf{G} \in supp(\mathcal{D})$ there exists a unique minimizer $\mathbf{U}^*(\mathbf{G})$ of problem (1), then $\mathbf{U}^*$ is permutation-equivariant.*

Graph Neural Networks, introduced in [19], and further developed in [20, 21] (see also the recent surveys [12, 22]), are a class of parameterized permutation-equivariant functions. They are

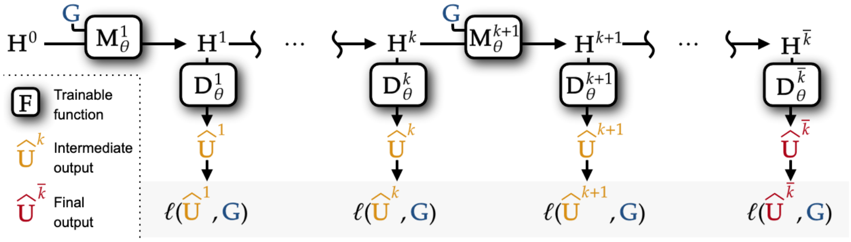

Figure 3: **Graph Neural Network implementation of a DSS**

hence natural candidates to build SSP solutions, since Property 1 states that the ideal solver $\mathbf{U}^*$ is permutation-equivariant.

Overall architecture There are many possible implementations of GNNs [12, 22]. But whatever the chosen type, it is important to make room for information propagation throughout the whole network (see also Section 4). Hence the choice of an iterative process that acts on a latent state $\mathbf{H} \in \mathcal{U}_d$; $H_i \in \mathbb{R}^d, d \geq 1$ for $\overline{k}$ iterations (d and $\overline{k}$ are hyperparameters). For a node $i \in [n]$, the latent state H_i can be seen as an embedding of the actual state U_i.

The overall architecture is described in Figure 3. All latent states in $\mathbf{H}^0$ are initialized to a zero vector. The *message passing* step performs $\overline{k}$ updates on the latent state variable $\mathbf{H}$ using $\mathbf{M}_\theta^k$, spreading information using interaction coefficients $\mathbf{A}$ and external inputs $\mathbf{B}$ of $\mathbf{G}$ (eq. 5–8). After each update, latent state $\mathbf{H}^k$ is decoded into a meaningful actual state $\widehat{\mathbf{U}}^k$ (eq. 9). The last state $\widehat{\mathbf{U}}^{\overline{k}}$ is the actual output of the algorithm $\widehat{\mathbf{U}}$. However, in order to robustify learning, all intermediate states $\widehat{\mathbf{U}}^k$ are taken into account in the training loss through a discounted sum with hyperparameter $\gamma \in [0, 1]$:

$$\text{Training Loss} = \sum_{k=1}^{\overline{k}} \gamma^{\overline{k}-k} \ell(\widehat{\mathbf{U}}^k, \mathbf{G}) \tag{4}$$

Message passing $\mathbf{M}_\theta^k$ For each node i, three different messages are computed, $\phi^k_{\rightarrow,\theta}, \phi^k_{\leftarrow,\theta}, \phi^k_{\circlearrowleft,\theta}$, corresponding to outgoing, ingoing and self-loop links, respectively using trainable mappings $\Phi^k_{\rightarrow,\theta}, \Phi^k_{\leftarrow,\theta}, \Phi^k_{\circlearrowleft,\theta}$, as follows:

$$\phi^k_{\rightarrow,i} = \sum_{j \in \mathcal{N}^\star(i;\mathbf{G})} \Phi^k_{\rightarrow,\theta}(H_i^{k-1}, A_{ij}, H_j^{k-1}) \qquad \textit{outgoing edges} \tag{5}$$

$$\phi^k_{\leftarrow,i} = \sum_{j \in \mathcal{N}^\star(i;\mathbf{G})} \Phi^k_{\leftarrow,\theta}(H_i^{k-1}, A_{ji}, H_j^{k-1}) \qquad \textit{ingoing edges} \tag{6}$$

$$\phi^k_{\circlearrowleft,i} = \Phi^k_{\circlearrowleft,\theta}(H_i^{k-1}, A_{ii}) \qquad \textit{self loop} \tag{7}$$

Latent states H_i^k are then computed using trainable mapping Ψ_θ^k, in a ResNet-like fashion:

$$\mathbf{H}^k = \mathbf{M}_\theta^k(\mathbf{H}^{k-1}, \mathbf{G}) := (H_i^k)_{i\in[n]}, \text{ with } H_i^k = H_i^{k-1} + \Psi_\theta^k(H_i^{k-1}, B_i, \phi^k_{\rightarrow,i}, \phi^k_{\leftarrow,i}, \phi^k_{\circlearrowleft,i}) \tag{8}$$

Decoding The decoding step applies the same trainable mapping Ξ_θ^k to every node:

$$\widehat{\mathbf{U}}^k = \mathbf{D}_\theta^k(\mathbf{H}^k) = (\Xi_\theta^k(H_i^k))_{i\in[n]} \tag{9}$$

Training All trainable blocks $\Phi^k_{\rightarrow,\theta}, \Phi^k_{\leftarrow,\theta}, \Phi^k_{\circlearrowleft,\theta}$ and Ψ_θ^k for the message passing phase, and Ξ_θ^k for the decoding phase, are implemented as Neural Networks. They are all trained simultaneously, backpropagating the gradient of the training loss of eq. (4) (see details in Section 5).

Inference Complexity Assuming that each neural network block has a single hidden layer with dimension d, that $d \geq d_A, d_B, d_U$, and denoting by m the average neighborhood size, one inference has computational complexity of order $\mathcal{O}(mn\overline{k}d^3)$, scaling linearly with n. Furthermore, many problems involve very local interactions, resulting in small m. However, one should keep in mind that hyperparameters $\overline{k}$ and d should be chosen according to the charateristics of distribution $\mathcal{D}$.

Equivariance The proposed architecture defines permutation-equivariant DSS, as proved in Appendix B.1.

4 Deep Statistical Solvers are Universal Approximators for SSPs Solutions

This Section proves, heavily relying on [23], a Universal Approximation Theorem for the class of DSSs with Lipschitz activation function (*e.g.* ReLU) in the space of the solutions of SSPs.

The space of Interaction Graphs is a metric space for the distance

$$d(\mathbf{G}, \mathbf{G}') = \|\mathbf{A} - \mathbf{A}'\| + \|\mathbf{B} - \mathbf{B}'\| \text{ if } n = n' \text{ and } +\infty, \text{ otherwise}$$

Universal Approximation Property Given metric spaces $\mathcal{X}$ and $\mathcal{Y}$, a set of continuous functions $\mathcal{H} \subset \{f : \mathcal{X} \to \mathcal{Y}\}$ is said to satisfy the *Universal Approximation Property* (UAP) if it is dense in the space of all continuous functions $\mathcal{C}(\mathcal{X}, \mathcal{Y})$ (with respect to the uniform metric).
Denote by $\mathcal{H}_{d_{in}}^{d_{out}}$ a set of neural networks from $\mathbb{R}^{d_{in}}$ to $\mathbb{R}^{d_{out}}$, for which the UAP holds. It is known since [24] that the set of neural networks with at least one hidden layer, an arbitrarily large amount of hidden neurons, and an appropriate activation function, satisfies these conditions.

Hypothesis space Let $\overline{\overline{k}} \in \mathbb{N}$. We denote by $\mathcal{H}^{\overline{\overline{k}}}$ the set of graph neural networks defined in Section 3 such that $\overline{k} \leq \overline{\overline{k}}$, $d \in \mathbb{N}$ and for any $k = 1, \ldots, \overline{k}$, we consider all possible $\Phi^k_{\rightarrow,\theta}, \Phi^k_{\leftarrow,\theta} \in \mathcal{H}^d_{d_A+2d}$, $\Phi^k_{\circlearrowleft,\theta} \in \mathcal{H}^d_{d_A+d}$, $\Psi^k_\theta \in \mathcal{H}^d_{d_B+4d}$ and $\Xi^k_\theta \in \mathcal{H}^{d_U}_d$.

Diameter of an Interaction Graph Let $\mathbf{G} = (n, \mathbf{A}, \mathbf{B}) \in \mathcal{G}$, and let $\widetilde{\mathbf{G}}$ be its undirected and unweighted graph structure, as defined in Section 2.1. We will write diam$(\mathbf{G})$ for diam$(\widetilde{\mathbf{G}})$, the diameter of $\widetilde{\mathbf{G}}$ [25].

Hypotheses over distribution $\mathcal{D}$ We introduce the four following hypotheses over supp$(\mathcal{D})$:

- *Permutation-invariance.* For any $\mathbf{G} \in$ supp$(\mathcal{D})$ and $\sigma \in \Sigma_n$, $\sigma \star \mathbf{G} \in$ supp$(\mathcal{D})$;
- *Compactness.* supp$(\mathcal{D})$ is a compact subset of $\mathcal{G}$;
- *Connectivity.* For any $\mathbf{G} \in$ supp$(\mathcal{D})$, $\widetilde{\mathbf{G}}$ has only one connected component.
- *Separability of external inputs.* There exist $\delta > 0$ such that for any $\mathbf{G} = (n, \mathbf{A}, \mathbf{B}) \in$ supp$(\mathcal{D})$ and any $i \neq j \in [n]$, $\|B_i - B_j\| \geq \delta$

The *compactness* implies that there is an upper bound $\overline{n}$ over the size n of Interaction Graphs in supp$(\mathcal{D})$. Also, these hypotheses imply that there is a finite upper bound on the diameters of all $\mathbf{G}$s. In the following, Δ will denote such upper bound. We denote by $\mathcal{C}_{\text{eq.}}$(supp$(\mathcal{D})$) the set of continuous and permutation-equivariant functions over supp$(\mathcal{D})$.

Theorem 1. *Let $\mathcal{D}$ be a distribution over $\mathcal{G}$ for which the above hypotheses hold.*

$$\textit{Then if } \overline{\overline{k}} \geq \Delta + 2, \ \mathcal{H}^{\overline{\overline{k}}} \textit{ is dense in } \mathcal{C}_{eq.}(\textit{supp}(\mathcal{D})).$$

Sketch of the proof (see Appendix B.3 for all details) Still following [23], we first prove a modified version of the *Stone-Weierstrass theorem for equivariant functions*. This theorem guarantees that a certain subalgebra of functions is dense in the set of continuous and permutation-equivariant functions if it separates non-isomorphic Interaction Graphs. Following the idea of Hornik et al. [24], we extend the hypothesis space to ensure closure under addition and multiplication. We then prove that the initial hypothesis space is dense in this new subalgebra. Finally, we conclude the proof by showing that the separability property mentioned above is satisfied by this newly-defined subalgebra.

Corollary 1. *Let $\mathcal{D}$ be a distribution over $\mathcal{G}$ for which the above hypotheses hold. Let ℓ be a continuous and permutation-invariant loss function such that for any $\mathbf{G} \in$ supp$(\mathcal{D})$, problem (1) has a unique minimizer $\mathbf{U}^*(\mathbf{G})$, continuous w.r.t $\mathbf{G}$. Then for any $\epsilon > 0$, there exists $Solver_\theta \in \mathcal{H}^{\Delta+2}$, such that*

$$\forall \mathbf{G} \in \textit{supp}(\mathcal{D}), \|Solver_\theta(\mathbf{G}) - \mathbf{U}^*(\mathbf{G})\| \leq \epsilon$$

This corollary is an immediate consequence of Theorem 1 and ensures that there exists a DSS using at most $\Delta + 2$ propagation updates that approximates with an arbitrary precision for all $\mathbf{G} \in$ supp$(\mathcal{D})$ the actual solution of problem (1). This is particularly relevant when considering large Interaction Graphs that have small diameters.

5 Experiments

This section investigates the behavior and performances of DSSs on two SSPs. The first one amounts to solving linear systems, though the distribution of problems is generated from a discretized Poisson

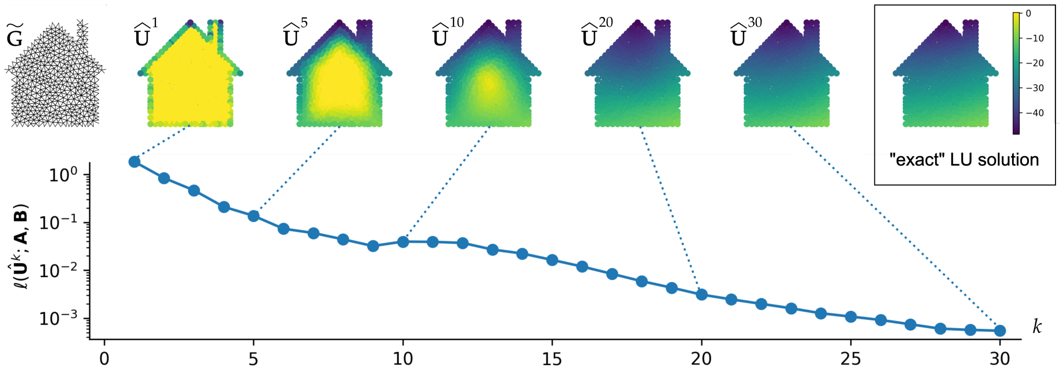

Figure 4: **Intermediate losses and predictions** - Top left: the structure graph $\widetilde{\mathbf{G}}$ (the mesh); Top right: the LU solution; Bottom: evolution of the loss along the $\bar{k} = 30$ updates for a trained DSS, at inference time. The intermediate predictions $\widehat{\mathbf{U}}^k$ are displayed for several values of k.

PDE. The second is the (non-quadratic) AC power flow computation. In all cases, the dataset is split into training/validation/test sets, the hyperparameters that are not explicitely mentioned are found by trial and errors using the validation set, and **all results presented are results on the test set**.

All trainings are performed with the Adam optimizer [26] with the standard hyperparameters of TensorFlow 1.14 [27], running on an Nvidia GeForce RTX 2080 Ti. Gradient clipping was used to avoid exploding gradient issues. **In the following, all experiments were repeated three times, with the same datasets and different initialization seeds** (as reported in Tables 1 and 2). Our code is made available in the supplementary materials, and links to the datasets can be found as references.

5.1 Solving Linear Systems from a Discretized PDE

Problem, and goals of experiments The example SSP considered here comes from the Finite Element Method applied to solve the 2D Poisson equation, one of the simplest and most studied PDE in applied mathematics: the geometry of the domain of the equation is discretized into an unstructured mesh, and computing the vector $\mathbf{U}$ of solution values at each node of the mesh amounts to solving a linear system $\mathbf{AU} = \mathbf{B}$ obtained by assembling local equations [28]. $\mathbf{A}$ and $\mathbf{B}$ encode both the geometry of the problem and the boundary conditions.

For illustration purposes, the Poisson equation can be used to model a field of temperature. In Figure 4, the geometry (house profile) is shown in the Top Left. The result of the optimization is the field of temperature everywhere in the house (shown in the Top Right).

This problem is easily set as an SSP in which each node i corresponds to a node of the mesh, all parameters are scalars ($d_A = d_B = d_U = 1$), and the loss function is

$$\ell(\mathbf{U},\mathbf{G})=\sum_{i\in[n]}\left(\sum_{j\in[n]} A_{ij}U_j - B_i\right)^2 \tag{10}$$

Our goal here is of course not to solve the Poisson equation, nor is it to propose a new competitive method to invert linear systems. As a matter of fact, the proposed approach does not make use of the linearity of the problem. Our goal is actually twofold: i) validate the DSS approach in high dimension ($n \approx 500$ nodes), and ii) analyze how DSS learns the distribution $\mathcal{D}$. Here, the distribution $\mathcal{D}$ is defined by the specific structure of linear systems that result from the discretization of the Poisson equation. In particular, we will carefully study the generalization capability of the learned model in terms of problem size, for similar problem structures.

Experimental conditions The dataset [29] consists of 96180/32060/32060 training/validation/test examples from the distribution generated from the discretization of the Poisson equation: randomly generated 2D geometries and random values for the second-hand function f and boundary condition g are used to compute the $\mathbf{A}$s and $\mathbf{B}$s. Their number of nodes n are around 500 (max 599) (automatic mesh generators don't allow a precise control of n).

The number of updates $\bar{k}$ is set to 30 (average diameter size for meshes with 500 elements). Each NN block has one hidden layer of dimension $d = 10$ and a leaky-ReLU non linearity. The complete DSS has $49,830$ weights. Training is done for $280,000$ iterations with batch size 100, and lasted $48h$.

Two baseline methods are considered [30], the direct LU decomposition, that could be considered giving the "exact" solution for these sizes of matrices, and the iterative Biconjugate Gradient Stabilized methods (BGS), with stopping tolerances of 10^{-3}. These algorithms are run on an Intel Xeon Silver 4108 CPU (1.80GHz) (GPU implementations were not available, they can decrease LU computational cost by a factor 6 [31]).

Method	DSS (3 runs)			LU	BGS (10^{-3})
Correlation w/ LU	$\mathbf{99.99\%}$			-	-
Time per instance (ms)	$\mathbf{1.8}$			2.4	2.3
Loss 10^{th} percentile	3.0 10^{-4}	3.8 10^{-4}	2.7 10^{-4}	4.5 10^{-27}	1.3 10^{-3}
Loss median	6.0 10^{-4}	1.3 10^{-3}	6.9 10^{-4}	6.1 10^{-26}	1.7 10^{-2}
Loss 90^{th} percentile	1.5 10^{-3}	4.0 10^{-3}	2.3 10^{-3}	6.3 10^{-25}	1.1 10^{-1}

Table 1: **Solving specific linear systems** – for similar accuracy, DSS is faster than the iterative BGS, while highly correlated with the "exact" solution as given by LU.

Results Table 1 displays comparisons between a trained DSS and the baselines. First, these results validate the approach, demonstrating that DSS can learn to solve 500 dimensional problems rather accurately, and in line with the "exact" solutions as provided by the direct method LU (99.99% correlation). Second, DSS is slighly but consistently faster than the iterative method BGS for similar accuracy (a tunable parameter of BGS). Further work will explore how DSS scales up in much higher dimensions, in particular when LU becomes intractable.
Figure 4 illustrates, on a hand-made test example (the mesh is on the upper left corner), how the trained DSS updates its predictions, at inference time, along the $\bar{k}$ updates. The flow of information from the boundary to the center of the geometry is clearly visible.

But what did exactly the DSS learn? Next experiments are concerned with the super-generalization capability of DSSs, looking at their results on test examples sampled from distributions departing from the one used for learning.

Super-Generalization We now experimentally analyze how well a trained model is able to generalize to a distribution $\mathcal{D}$ that is different from the training distribution. The same data generation process that was used to generate the training dataset (see above) is now used with meshes of very different sizes, everything else being equal. Whereas the training distribution only contains Interaction Graphs of sizes around 500, out-of-distribution test examples have sizes from 100 and 250 (left of Figure 5) up to 750 and 1000 (right of Figure 5). In all cases, the trained model is able to achieve a correlation with the "true" LU solution as high as 99.99%. Interestingly, the trained model achieves a higher correlation with the LU solutions for data points with a lower number of nodes. Further experiments with even larger sizes are needed to reach the upper limit of such a super generalization. Nevertheless, thanks to the specific structure dictated to the linear system by the Poisson equation, DSS was able to perform some kind of zero-shot learning for problems of very different sizes.
Other experiments (see Appendix C) were performed by adding noise to $\mathbf{A}$ and $\mathbf{B}$. The performance of the trained model remains good for small noise, then smoothly degrades as the noise increases.

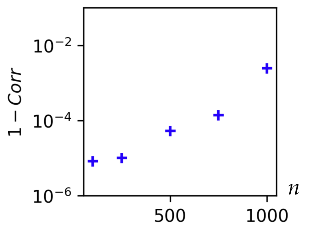

Figure 5: **Varying problem size** n: Correlation (DSS, LU)

5.2 AC power flow experiments

Problem and goals of experiments The second SSP example is the AC power flow prediction. The goal is to compute the steady-state electrical flows in a Power Grid, an essential part of real-time operations. Knowing the amount of power that is being produced and consumed throughout the grid (encoded into $\mathbf{B}$), and the way power lines are interconnected, as well as their physical properties (encoded into $\mathbf{A}$), the goal is to compute the voltage defined at each electrical node $V_i = |V_i|e^{\mathbf{j}\theta_i}$ ($\mathbf{j}$ denotes the imaginary unit), which we encode in the states $\mathbf{U}$. Kirchhoff's law (energy conservation at every node) governs this system, and the violation of this law is directly used as loss function ℓ. Moreover, some constraints over the states $\mathbf{U}$ are here relaxed and included as an additional term of the loss (with factor λ). One should also keep in mind that the main goal is to predict power flows, and not the voltages per se: Both aspects will be taken into account by measuring the correlation w.r.t $|V_i|$, θ_i, P_{ij} (real part of power flow) and Q_{ij} (imaginary part). This problem is highly non-linear, and a substantial overview is provided in [32]. This set of complex equations can be converted into a SSP using $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{U}$ as defined above ($d_A = 2$, $d_B = 5$, $d_U = 2$), and loss function ℓ:

$$\ell(\mathbf{U},\mathbf{G}) = \sum_{i\in[n]}(1-B_i^5)\big(-B_i^1+U_i^1\sum_{j\in[n]}A_{ij}^1U_j^1\cos(U_i^2-U_j^2-A_{ij}^2)\big)^2 \\ +\sum_{i\in[n]}B_i^3\big(-B_i^2+U_i^1\sum_{j\in[n]}A_{ij}^1U_j^1\sin(U_i^2-U_j^2-A_{ij}^2)\big)^2+\lambda\sum_{i\in[n]}(1-B_i^3)\big(U_i^1-B_i^4\big)^2 \tag{11}$$

Dataset	IEEE 14 nodes				IEEE 118 nodes			
Method	**DSS (3 runs)**			NR	**DSS (3 runs)**			NR
Correlation w/ NR	$\mathbf{99.99\%}$			-	$\mathbf{99.99\%}$			-
Time per instance (ms)	$\mathbf{1 \times 10^{-2}}$			2×10^{1}	$\mathbf{9 \times 10^{-2}}$			2×10^{1}
Loss 10^{th} percentile $\times 10^{6}$	25	34	41	$1.4\, 10^{-6}$	0.53	13	1.7	$2.9\, 10^{-8}$
Loss median $\times 10^{6}$	40	63	100	$2.1\, 10^{-6}$	1.3	17	2.6	$4.2\, 10^{-8}$
Loss 90^{th} percentile $\times 10^{6}$	100	150	440	$3.3\, 10^{-6}$	3.4	25	4.8	$6.4\, 10^{-8}$

Table 2: **Solving specific AC power flow**– our trained DSS models are highly correlated with the Newton-Raphson solutions, while being 2 to 3 orders of magnitude faster.

More details about the conversion from classical power systems notations to this set of variables is provided in Appendix D. This loss is not quadratic, as demonstrated by the presence of sinusoidal terms. One can notice the use of binary variables B_i^3 and B_i^5.

Experimental conditions Experiments are conducted on two standard benchmarks from the *Learning to Run a Power Network* competition [33]: IEEE case14 ($n = 14$) [34], and IEEE case118 ($n = 118$) [35]. In order to increase the diversity in terms of grid topology (encoded in $\mathbf{A}$), for each example, one (resp. two) randomly chosen power lines are disconnected with probability 25% (resp. 25%). For case14 (resp. case118), the dataset is split into $16064/2008/2008$ (resp. $18432/2304/2304$).

Each NN block has a single hidden layer of dimension $d = 10$ and a leaky-ReLU non linearity. For case14 (resp. case118), $\overline{k}$ was set to 10 (resp. 30). The number of weights is $1,722$ for each of the $\overline{k}$ $(\mathbf{M}, \mathbf{D})$ blocks, hence $17,220$ (resp. $51,660$) in total. Training is done for $883,000$ (resp. 253000) iterations with batch size $1,000$ (resp. 500), and lasted $48h$.

State-of-the-art AC power flow computation uses the Newton-Raphson method, used as baseline here (PandaPower implementation [36], on an Intel i5 dual-core (2.3GHz)). To the best of our knowledge, no GPU implementation was available, although recent work [37, 38] investigate such an avenue.

Results In both cases, correlations between power flows output by the trained DSSs and the Newton-Raphson method are above 99.99% (both real P_{ij} and imaginary Q_{ij}). In the 14 nodes case, correlations for V_i and θ_i are also above 99.99%. In the 118 nodes case, those correlations are respectively around 99.9% and 95% (except for model #2, for which it is 99.7% and 83%). This might be caused by the presence of large but loosely interconnected communities in the graphs of case118, and will be the object of further investigations. Table 2 shows the huge acceleration of DSS (by two orders of magnitude) over Newton-Raphson, at the cost of an important decrease in accuracy, although both methods output very similar power flows (correlation higher than 99.99%).

6 Conclusions and Future Work

This paper proposes a novel paradigm that blends statistics and optimization, Statistical Solver Problems. In the SSP framework, a single solver is trained to solve a family of problem instances sampled from a given distribution of optimization problems, possibly arising from system simulations. Such training is performed by directly minimizing the loss of the optimization problems at hand. In particular, no existing solutions (obtained from costly simulations) are needed for training. The Deep Statistical Solvers proposed in this paper, as a particular embodiment of the new proposed framework, is a class of Graph Neural Network, well suited to solving SSPs for which the loss function is permutation-invariant, and for which we theoretically prove some universal approximation properties.

The effectiveness of DSSs are experimentally demonstrated, showing a good compromise between accuracy and speed in dimensions up to 500 on two sample problems: solving linear systems, and the non-linear AC power flow. The accuracy on power flow computations matches that of state-of-the-art approaches while speeding up calculations by 2 to 3 orders of magnitude.

Future work will focus on incorporating discrete variables in the state space. Other avenues for research concern further theoretical improvements to investigate convergence properties of the DSS approach, in comparison to other solvers, as well as investigations on the limitations of the approach.

References

[1] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. *ArXiv: 1607.03597*, 2016.

[2] M. F. Kasim, D. Watson-Parris, L. Deaconu, S. Oliver, P. Hatfield, D. H. Froula, G. Gregori, M. Jarvis, S. Khatiwala, J. Korenaga, J. Topp-Mugglestone, E. Viezzer, and S. M. Vinko. Up to two billion times acceleration of scientific simulations with deep neural architecture search, 2020.

[3] T. T. Nguyen. Neural network load-flow. *IEEE Proceedings - Generation, Transmission and Distribution*, 142:51–58(7), 1995.

[4] S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent sub-grid processes in climate models. *PNAS*, 2018.

[5] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4), Dec 2019.

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[7] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *ArXiv: 1907.04502*, 2019.

[8] V. Vemuri. Modeling of complex systems: An introduction. *New York: Academic Press.*, 1978.

[9] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–. Springer-Verlag, 1999.

[10] M. T. McCann, K. H. Jin, and M. Unser. Convolutional neural networks for inverse problems in imaging: A review. *IEEE Signal Processing Magazine*, 34(6):85–95, 2017.

[11] P. W. Battaglia and al. Relational inductive biases, deep learning, and graph networks. *ArXiv: 1806.01261*, 2018.

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2020.

[13] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[14] V. Bolz, J. Rueß, and A. Zell. Power flow approximation based on graph convolutional networks. In *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1679–1686, 2019.

[15] T. N. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. S. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

[16] C. Chi, Y. Weike, Z. Yunxing, Z. Chen, and O. S. Ping. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *ArXiv: 1704.01212*, 2017.

[18] B. Donon, B. Donnot, I. Guyon, and A. Marot. Graph neural solver for power systems. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.

[19] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks Proceedings*, volume 2, pages 729–734, 2005.

[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[21] C. Gallicchio and A. Micheli. Graph echo state networks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010.

[22] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *ArXiv: 1812.08434*, 2018.

[23] N. Keriven and G. Peyré. Universal invariant and equivariant graph neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7092–7101. Curran Associates, Inc., 2019.

[24] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[25] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., USA, 2010.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[27] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[28] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Mathematics and its Applications. 1978.

[29] Dataset of linear systems built from the poisson equation. `http://doi.org/10.5281/zenodo.3878581`.

[30] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[31] N. Galoppo, N. K. Govindaraju, M. Henson, and D. Manocha. LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware. In *ACM/IEEE SC 2005 Conference*, pages 3–3. IEEE, 2005.

[32] P. S. Kundur. Power system stability. In *Power System Stability and Control, Third Edition*, pages 1–12. CRC Press, 2012.

[33] A. Marot, B. Donnot, C. Romero, L. Veyrin-Forrer, M. Lerousseau, B. Donon, and I. Guyon. Learning to run a power network challenge for training topology controllers. *ArXiv: 1912.04211*, 2019.

[34] Dataset of ac power flow systems on the ieee case14 power grid. `http://doi.org/10.5281/zenodo.3878677`.

[35] Dataset of ac power flow systems on the ieee case118 power grid. `http://doi.org/10.5281/zenodo.3878686`.

[36] L. Thurner, A. Scheidler, F. Schafer, J. H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun. pandapower - an open source python tool for convenient modeling, analysis and optimization of electric power systems. *IEEE Transactions on Power Systems*, 2018.

[37] X. Su, C. He, T. Liu, and L. Wu. Full parallel power flow solution: A gpu-cpu-based vectorization parallelization and sparse techniques for newton–raphson implementation. *IEEE Transactions on Smart Grid*, 11(3):1833–1844, 2020.

[38] D.-H. Yoon and Y. Han. Parallel power flow computation trends and applications: A review focusing on gpu. *Energies*, 13:2147, 05 2020.

A Underlying Graph Structure

We generalize the standard notion of neighborhood to the setting of Interaction Graphs and SSP $(\mathcal{G},\mathcal{D},\mathcal{U},\ell)$. The intuitive way of defining neighbors of a node i is to look for nodes j such that $A_{ij} \neq 0$ or $A_{ji} \neq 0$. However this intuitive definition does not perfectly suit the case of SSPs as the properties of the loss function ℓ do impact the interactions between nodes.

For instance, let the loss function ℓ be defined by $\ell(\mathbf{U},\mathbf{G}) = \sum_{i\in[n]} f(U_i)$, for some real-valued function f. In this case, there is no interaction between nodes when computing the state of the Interaction Graph, even though some coefficient A_{ij} may be non zero. We note in this case that:

$$\forall i \neq j, \forall \mathbf{U} \in \mathcal{U}, \frac{\partial^2 \ell}{\partial U_i \partial U_j}(\mathbf{U},\mathbf{G}) = 0 \tag{12}$$

We thus propose the following definition of the neighborhood of a node i with respect to Interaction Graph $\mathbf{G}$ and loss function ℓ:

$$\mathcal{N}(i;\mathbf{G},\ell) = \left\{ j \in [n] | \exists \mathbf{U}, \frac{\partial^2 \ell}{\partial U_i \partial U_j}(\mathbf{U},\mathbf{G}) \neq 0 \right\} \tag{13}$$

For a given class of SSP, the loss function ℓ does not change, so it will be omitted in the following, and we will write $\mathcal{N}(i;\mathbf{G})$.

One can observe that in the case of a quadratic optimization problem where $d_A = d_B = d_U = 1$ and $\ell(\mathbf{U},\mathbf{G}) = \mathbf{U}^T\mathbf{A}\mathbf{U} + \mathbf{B}^T\mathbf{U}$, this notion of neighborhood is exactly that given in Section 2.1, and $\widetilde{\mathbf{G}}$ is indeed the undirected graph defined by the non-zero entries of $\mathbf{A}$ (or more precisely those of $(\mathbf{A}+\mathbf{A}^T)/2$ when $\mathbf{A}$ is not symmetric).

B Mathematical proofs

In this section, we'll follow Keriven and Peyré [23] and use the notation $[\mathbf{G}]_i$ to denote the i^{th} component of any Interaction Graph or hyper-graph G. In the following, 'dense' means 'dense with respect to the uniform metric' by default. As a reminder, the uniform metric $\overline{d}$ on function spaces given two metric spaces (X, d_X) and (Y, d_Y) is defined by

$$\overline{d}(f, f') := \sup_{x\in X} d_Y(f(x), f'(x)). \tag{14}$$

B.1 Proof of equivariance of the proposed DSS architecture

The following is a proof of the equivariance of the architecture proposed in Section 3.

Proof. Because the loss function ℓ is permutation invariant, we only have to prove that eq. (8)-(9) satisfy the permutation-equivariance property.

Let us prove by induction on k that $\mathbf{H}^k$ is permutation-equivariant (by a slight abuse of notation in eq. (8), we consider the latent states $\mathbf{H}^k$ as functions of $\mathbf{G}$), i.e. that $\mathbf{H}^k(\sigma \star \mathbf{G}) = \sigma \star \mathbf{H}^k(\mathbf{G})$.

For $k = 0$, it is clear that $\sigma \star \mathbf{H}^0 = (0, ..., 0)_{i\in[n]} = \mathbf{H}^0$, which is independant of $\mathbf{G}$. Now suppose the equivariance property holds for $\mathbf{H}^{k-1}$, then from eq. (5) comes

$$\begin{aligned}
[\phi^k_{\rightarrow}(\sigma \star \mathbf{G})]_i &= \sum_{j\in\mathcal{N}^\star(i;\sigma\star\mathbf{G})} \Phi^k_{\rightarrow,\theta}(H^{k-1}_i(\sigma \star \mathbf{G}), (\sigma \star \mathbf{A})_{ij}, H^{k-1}_j(\sigma \star \mathbf{G})) \qquad (15)\\
&= \sum_{j\in\mathcal{N}^\star(i;\sigma\star\mathbf{G})} \Phi^k_{\rightarrow,\theta}([\sigma \star \mathbf{H}^{k-1}(\mathbf{G})]_i, (\sigma \star \mathbf{A})_{ij}, [\sigma \star \mathbf{H}^{k-1}(\mathbf{G})]_j) \qquad (16)\\
&= \sum_{j\in\mathcal{N}^\star(i;\sigma\star\mathbf{G})} \Phi^k_{\rightarrow,\theta}(H^{k-1}_{\sigma^{-1}(i)}(\mathbf{G}), A_{\sigma^{-1}(i)\sigma^{-1}(j)}, H^{k-1}_{\sigma^{-1}(j)}(\mathbf{G})) \qquad (17)\\
&= \sum_{\sigma^{-1}(j)\in\mathcal{N}^\star(\sigma^{-1}(i);\mathbf{G})} \Phi^k_{\rightarrow,\theta}(H^{k-1}_{\sigma^{-1}(i)}(\mathbf{G}), A_{\sigma^{-1}(i)\sigma^{-1}(j)}, H^{k-1}_{\sigma^{-1}(j)}(\mathbf{G})) \qquad (18)\\
&= \sum_{j\in\mathcal{N}^\star(\sigma^{-1}(i);\mathbf{G})} \Phi^k_{\rightarrow,\theta}(H^{k-1}_{\sigma^{-1}(i)}(\mathbf{G}), A_{\sigma^{-1}(i)j}, H^{k-1}_j(\mathbf{G})) \qquad (19)\\
&= [\phi^k_{\rightarrow}(\mathbf{G})]_{\sigma^{-1}(i)} \qquad (20)\\
&= [\sigma \star \phi^k_{\rightarrow}(\mathbf{G})]_i \qquad (21)
\end{aligned}$$

All the above equalities are straightforward, except maybe eq. (18), which comes from the equivariance property of the notion of neighborhood defined above by eq. (13). The same property follows for $\phi^k_{\leftarrow}$ by similar argument.

For $\phi^k_{\circlearrowleft}$, eq. (7) gives

$$\begin{aligned}
[\phi^k_{\circlearrowleft}(\sigma \star \mathbf{G})]_i &= \Phi^k_{\circlearrowleft,\theta}([\mathbf{H}^{k-1}(\sigma \star \mathbf{G})]_i, (\sigma \star \mathbf{A})_{ii}) \qquad (22)\\
&= \Phi^k_{\circlearrowleft,\theta}([\sigma \star \mathbf{H}^{k-1}(\mathbf{G})]_i, (\sigma \star \mathbf{A})_{ii}) \qquad (23)\\
&= \Phi^k_{\circlearrowleft,\theta}(H^{k-1}_{\sigma^{-1}(i)}(\mathbf{G}), A_{\sigma^{-1}(i)\sigma^{-1}(i)}) \qquad (24)\\
&= [\phi^k_{\circlearrowleft}(\mathbf{G})]_{\sigma^{-1}(i)} \qquad (25)\\
&= [\sigma \star \phi^k_{\circlearrowleft}(\mathbf{G})]_{(i)} \qquad (26)
\end{aligned}$$

This concludes the proof that $\mathbf{H}^k_i$ is permutation equivariant for all k, and from eq. (8) we conclude that $\mathbf{M}^k_\theta$ is permutation-equivariant. Similar proof holds for $\mathbf{D}^k_\theta$ and $\hat{\mathbf{U}}^k$, which in turn prove that

$$\widehat{\mathbf{U}}(\sigma \star \mathbf{G}) = \sigma \star \widehat{\mathbf{U}}(\mathbf{G}). \qquad (27)$$

This concludes the proof. □

B.2 Proof of Property 1

In Section 3, Property 1 states that if the loss function ℓ is permutation-invariant and if for any $\mathbf{G} \in \text{supp}(\mathcal{D})$ there exists a unique minimizer $\mathbf{U}^*(\mathbf{G})$ of problem (1), then $\mathbf{U}^*$ is permutation-equivariant.

Let ℓ be a permutation-invariant loss function and $\mathcal{D}$ a distribution such that for any $\mathbf{G} \in \text{supp}(\mathcal{D})$ there is a unique solution $\mathbf{U}^*$ of problem 1. Let $\mathbf{G} = (n, \mathbf{A}, \mathbf{B}) \in \text{supp}(\mathcal{D})$ and $\sigma \in \Sigma_n$ a permutation.

$$\begin{aligned}
\ell(\sigma \star \mathbf{U}^*(\mathbf{G}), \sigma \star \mathbf{G}) &= \ell(\mathbf{U}^*(\mathbf{G}), \mathbf{G}) && \text{by invariance of } \ell \qquad (28)\\
&= \min_{\mathbf{U}\in\mathcal{U}} \ell(\mathbf{U}, \mathbf{G}) && \text{by definition of } \mathbf{U}^* \qquad (29)\\
&= \min_{\mathbf{U}\in\mathcal{U}} \ell(\sigma \star \mathbf{U}, \sigma \star \mathbf{G}) && \text{by invariance of } \ell \qquad (30)\\
&= \min_{\mathbf{U}\in\mathcal{U}} \ell(\mathbf{U}, \sigma \star \mathbf{G}) && \text{by invariance of } \mathcal{U} \qquad (31)\\
&= \ell(\mathbf{U}^*(\sigma \star \mathbf{G}), \sigma \star \mathbf{G}) && \text{by definition of } \mathbf{U}^* \qquad (32)
\end{aligned}$$

Moreover the uniqueness of the solution ensures that $\mathbf{U}^*(\sigma \star \mathbf{G}) = \sigma \star \mathbf{U}^*(\mathbf{G})$, which concludes the proof.

B.3 Proof of Theorem 1

We will now prove Theorem 1, the main result on DSSs, by closely following the approach of [23]. We will first prove a modified version of the Stone-Weierstrass theorem, and then verify that the defining spaces for Interaction Graphs indeed verify the conditions of this theorem by proving several lemmas (most importantly Theorem 3 on separability).

Let $\mathcal{G}_{\text{eq.}} \subseteq \mathcal{G}$ be a set of compact, permutation-invariant Interaction Graphs. The compactness implies that there exist $\overline{n} \in \mathbb{N}$ such that all graphs in $\mathcal{G}_{\text{eq.}}$ have an amount of nodes lower than $\overline{n} \in \mathbb{N}$. Let $\mathcal{C}_{\text{eq.}}(\mathcal{G}_{\text{eq.}}, \mathcal{U})$ be the space of continuous functions from $\mathcal{G}_{\text{eq.}}$ on $\mathcal{U}$ that associate to any Interaction Graph $\mathbf{G} = (n, \mathbf{A}, \mathbf{B})$ one of its possible states $\mathbf{U} \in \mathbb{R}^n$. $(\mathcal{C}_{\text{eq.}}(\mathcal{G}_{\text{eq.}}, \mathcal{U}), +, \cdot, \odot)$ is a unital $\mathbb{R}$-algebra, where $(+, \cdot)$ are the usual addition and multiplication by a scalar, and $\odot$ is the Hadamard product defined by $[(f \odot g)(x)]_i = [f(x)]_i \cdot [g(x)]_i$. Its unit is the constant function $\mathbf{1} = (1, \ldots, 1)$.

Theorem 2 (Modified Stone-Weierstrass theorem for equivariant functions)**.**
Let $\mathcal{A}$ be a unital subalgebra of $\mathcal{C}_{eq.}(\mathcal{G}_{eq.}, \mathcal{U})$, (i.e., it contains the unit function $\mathbf{1}$) and assume both following properties hold:

- *(Separability) For all $\mathbf{G}, \mathbf{G}' \in \mathcal{G}_{eq.}$, with number of nodes n and n' such that $\mathbf{G}$ is not isomorphic to $\mathbf{G}'$, and for all $k \in [n], k' \in [n']$, there exists $f \in \mathcal{A}$ such that $[f(\mathbf{G})]_k \neq [f(\mathbf{G}')]_{k'}$;*
- *(Self-separability) For all $n \leq \overline{n}$, $I \subseteq [n]$, $\mathbf{G} \in \mathcal{G}_{eq.}$ with n nodes, such that no isomorphism of $\mathbf{G}$ exchanges at least one index between I and I^c, and for all $k \in I$, $l \in I^c$, there exists $f \in \mathcal{A}$ such that $[f(\mathbf{G})]_k \neq [f(\mathbf{G})]_l$.*

Then $\mathcal{A}$ is dense in $\mathcal{C}_{eq.}(\mathcal{G}_{eq.}, \mathcal{U})$ with respect to the uniform metric.

This proof of Theorem 2 is almost identical to that of Theorem 4 in [23], with the following differences.

1. For the input space, we consider Interaction Graphs of the form $(n, \mathbf{A}, \mathbf{B})$ with $\mathbf{A} \in (\mathbb{R}^{d_A})^{n^2}$ and $\mathbf{B} \in (\mathbb{R}^{d_B})^n$, instead of hyper-graphs of the form $\mathbb{R}^{n^d}$ for $d \in \mathbb{N}$. The corresponding metrics are naturally different, although the difference is not critical for the proof;
2. Similarly, we consider an output space with $\mathbf{U} \in (\mathbb{R}^{d_U})^n$ instead of $\mathbb{R}^n$;
3. We only assume $\mathcal{G}_{\text{eq.}} \subseteq \mathcal{G}$ to be compact and permutation-invariant instead of a $\mathcal{G}_{\text{eq.}}$ with an explicit form: $\mathcal{G}_{\text{eq.}} := \{G \in \mathbb{R}^{n^d} | n \leq n_{\max}, \|G\| \leq R\}$ (which makes this modified theorem more general).

We shall then indicate how to bypass these differences one by one and then reuse the proofs in [23].

For 1, the only properties of the input space involved in [23] are the number of nodes, action of permutation and the metric (with the corresponding topology). For the first two points, everything is still applicable in our setting. For the topology, the difference is not critical either since we are actually considering the product space of two of metric spaces defined in [23] and all corresponding properties follow.

For 2, we can always reduce to the case with $d_U = 1$ then stack the resulting function d_U times to have the expected shape. This works seamlessly with Hadamard product and all properties related to density.

For 3, there is actually no dependency on the explicit form of $\mathcal{G}_{\text{eq.}}$ or $\mathbf{G}$ in [23] (as for the case in 1). And the proof only relies on the upper bound on the number of nodes. So this generalization can be naturally obtained.

The detailed proof of Theorem 2 then follows the exact same procedure than that of Theorem 4 in [23], and we shall omit it here, refering the reader to [23] for all details.

Let $\overline{\overline{k}} \in \mathbb{N}$, and, as defined in Section 4, let $\mathcal{H}^{\overline{\overline{k}}}$ be the set of graph neural networks defined in Section 3 such that $\overline{k} \leq \overline{\overline{k}}$. Our goal is to prove that Theorem 2 can be applied to $\mathcal{H}^{\overline{\overline{k}}}$.

Because $\mathcal{H}^{\overline{\overline{k}}}$ is not an algebra, let us consider $\mathcal{H}^{\overline{\overline{k}}\odot}$, the algebra generated by $\mathcal{H}^{\overline{\overline{k}}}$ with respect to the Hadamard product. More formally:

$$\mathcal{H}^{\overline{\overline{k}}\odot} = \left\{ \sum_{s=1}^{S} \bigodot_{t=1}^{T_s} c_{st} f_{st} | S \in \mathbb{N}, T_s \in \mathbb{N}, c_{st} \in \mathbb{R}, f_{st} \in \mathcal{H}^{\overline{\overline{k}}} \right\}. \tag{33}$$

Note that the Hadamard product among f_{st}'s is well-defined since for a fixed input $\mathbf{G}$, all output values $f_{st}(\mathbf{G})$ take the same dimension - the size of $\mathbf{G}$.

$(\mathcal{H}^{\overline{\overline{k}}\odot}, +, \cdot, \odot)$ is obviously a unital sub-algebra of $(\mathcal{G}_{\text{eq.}}, +, \cdot, \odot)$ (the constant function $(1, \ldots, 1)$ trivially belongs to $\mathcal{H}^{\overline{\overline{k}}\odot}$). In order to apply Theorem 2 to $\mathcal{H}^{\overline{\overline{k}}\odot}$, one needs to prove that it satisfies both separability hypotheses.

Let us first notice that the self-separability property is a straightforward consequence of the hypothesis of separability of external inputs on supp($\mathcal{D}$). Hence we only need to prove the separability property:

Theorem 3. *$\mathcal{H}^{\overline{\overline{k}}\odot}$ satisfies the separability property of Theorem 2.*

The proof consists of 3 steps. In step 1, we prove that for all $\mathbf{G}$, $\mathbf{G}' \in$ supp($\mathcal{D}$) that are not isomorphic, there exists a sequence *(node, edge, node)* that only exists in $\mathbf{G}$. In Step 2, we build a continuous function $f^\dagger$ on $\mathcal{G}$ that returns an indicator of the presence of this sequence in the input graph. In Step 3, we prove that there exists a function $f_\theta \in \mathcal{H}^{\overline{\overline{k}}\odot}$ that approximates well enough $f^\dagger$.

For Step 1, we formally state it in the following lemma.

Lemma 1. *Let $\mathbf{G} = (n, \mathbf{A}, \mathbf{B})$ and $\mathbf{G}' = (n', \mathbf{A}', \mathbf{B}')$ be in supp($\mathcal{D}$) such that $\mathbf{G}$ and $\mathbf{G}'$ are not isomorphic and $n \geq n'$. Then there exist $i, j \in [n]$, $i \neq j$, such that, for all $i', j' \in [n']$, the following inequality holds:*

$$(B_i, A_{ij}, B_j) \neq (B'_{i'}, A'_{i'j'}, B'_{j'}) \tag{34}$$

Proof. This lemma relies on the separability hypothesis of supp($\mathcal{D}$) which states that there exists $\delta > 0$ such that for all $\mathbf{G} = (n, \mathbf{A}, \mathbf{B}) \in$ supp($\mathcal{D}$) and for all $i \neq j \in [n]$, $\|B_i - B_j\| \geq \delta$.

We shall use proof by contradiction: assume that for any $(i, j) \in [n]^2$ with $i \neq j$, there exists $\alpha(i, j) = (i', j') \in [n']^2$ such that $(B_i, A_{ij}, B_j) = (B'_{i'}, A'_{i'j'}, B_{j'})$. Two cases must be distinguished, depending on whether $n < n'$ or $n = n'$

If $n > n'$, then according to the pigeonhole principle, there exist two pairs $(i, j) \in [n]^2$ and $(l, m) \in [n]^2$ that have the same image by α, $(i', j') \in [n']^2$. Hence, $(B_i, A_{ij}, B_j) = (B'_{i'}, A'_{i'j'}, B'_{j'}) = (B_l, A_{lm}, B_m)$, which contradicts the separability hypothesis for $\mathbf{G}$.

If $n = n'$, according to the separability hypothesis of supp($\mathcal{D}$), there cannot exist $i \neq l \in [n]$ that are mapped to the same $i' \in [n']$ (i.e. $\alpha(i, j) = (i', j')$ for some j, j' and $\alpha(l, m) = (i', m')$ for some m, m'). Thus α actually defines an injective mapping $\chi : [n] \rightarrow [n]$ on the first component. Because $n = n'$, this mapping is also surjective and hence bijective. Due to the symmetry of i and j, we see that the mapping on the second component χ' defined by X is exactly χ. Hence we have found a permutation $\chi \in \Sigma_n$ such that

$$B_i = B'_{\chi(i)} \tag{35}$$

$$A_{ij} = A_{\chi(i)\chi(j)} \tag{36}$$

for any $(i, j) \in [n]^2$, which means that $\mathbf{G}$ and $\mathbf{G}'$ are isomorphic, contradicting the hypothesis, and thus completing the proof. □

Let us now proceed with Step 2. For convenience, we shall use a continuous kernel function defined by

$$K_\epsilon(x) = \max(0, 1 - |x|/\epsilon) \tag{37}$$

for $\epsilon > 0$. Then we have $K_\epsilon(0) = 1$ and $K_\epsilon(x) = 0$ for $|x| > \epsilon$.

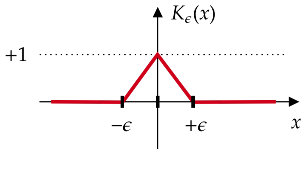

Figure 6: **Kernel function**

All intermediate functions of DSSs $\Phi^k_{\rightarrow}$, $\Phi^k_{\leftarrow}$, $\Phi^k_{\circlearrowleft}$, Ψ^k and Ξ^k (Section 3) live in function spaces that satisfy the Universal Approximation Property (UAP). So let us consider now a space of continuous functions that share the same architecture than DSS, but in which all spaces of parameterized neural networks have been replaced by corresponding continuous function space. We denote this space by $\mathcal{H}^{\overline{\overline{k}}\dagger}$ (by convention, a dagger(†) added to a Neural Network block from Section 3 will refer to the corresponding continuous function space (*e.g.* $\Phi^{k\dagger}_{\rightarrow}$). We are now in position to prove the following lemma.

Lemma 2. *For any $\mathbf{G}, \mathbf{G}' \in$ supp($\mathcal{D}$) that are not isomorphic, there exists a function $f^\dagger \in \mathcal{H}^{\overline{\overline{k}}\dagger}$ such that for any $k \in [n], k' \in [n']$, we have $[f^\dagger(\mathbf{G})]_k \neq [f^\dagger(\mathbf{G}')]_{k'}$.*

Proof. Without loss of generality, we suppose $n \geq n'$. According to Lemma 1, there exist $(i^\dagger, j^\dagger) \in [n]^2$, $i^\dagger \neq j^\dagger$, such that $\mathbf{G}$ contains a sequence $(B_{i^\dagger}, A_{i^\dagger j^\dagger}, B_{j^\dagger})$ that does not appear in $\mathbf{G}'$.

We are going to construct a continuous function $f^\dagger : \text{supp}(\mathcal{D}) \rightarrow \mathcal{U}$ that will be an indicator of the presence of the above sequence in the graph, and such that $f^\dagger(\mathbf{G}) = (1, \dots, 1) \in \mathbb{R}^n$ and $f^\dagger(\mathbf{G}') = (0, \dots, 0) \in \mathbb{R}^{n'}$ (thus proving Lemma 2).

Let us first recall the architecture of DSS, as defined by eq. (5)-(9), and let us choose *continuous* functions $\Phi^{1\dagger}_{\rightarrow}, \Phi^{1\dagger}_{\leftarrow}, \Phi^{1\dagger}_{\circlearrowleft}$ and $\Psi^{1\dagger}$ such that $\mathbf{H}^{1\dagger}$ is defined by, for any $\mathbf{G}'' \in \text{supp}(\mathcal{D})$,

$$[\mathbf{H}^{1\dagger}(\mathbf{G}'')]_i = 2K_\epsilon(\|B''_i - B_{i^\dagger}\|) - K_\epsilon(\|B''_i - B_{j^\dagger}\|) \tag{38}$$

where $\epsilon = \|A_{i^\dagger j^\dagger} - A'_{\sigma(i^\dagger)\sigma(j^\dagger)}\|$ if $\mathbf{B}$ and $\mathbf{B}'$ are isomorphic through permutation σ and $\epsilon = \min_\sigma \max_i \|B_i - B'_{\sigma(i)}\|$ otherwise. This function allows us to identify whether the external input B''_i is close to one of $B_{i^\dagger}$ or $B_{j^\dagger}$.

For $k = 2$, we define

$$\Phi^{2\dagger}_{\rightarrow}(h, a, h') = K_\epsilon(\|h - 2\| + \|a - A_{i^\dagger j^\dagger}\| + \|h' + 1\|) \tag{39}$$

and

$$\Phi^{2\dagger}_{\circlearrowleft}(h, a) = K_\epsilon(\|h - 1\| + \|a - A_{i^\dagger j^\dagger}\|). \tag{40}$$

Then we choose $\Psi^{2\dagger}$ such that

$$[\mathbf{H}^{2\dagger}]_i = \phi^{2\dagger}_{\circlearrowleft,i} + \phi^{2\dagger}_{\rightarrow,i} = \Phi^{2\dagger}_{\circlearrowleft}(H^{1\dagger}_i, A_{ii}) + \sum_{j \in \mathcal{N}^\star(i;\mathbf{G})} \Phi^{2\dagger}_{\rightarrow}(H^{1\dagger}_i, A_{ij}, H^{1\dagger}_j) \tag{41}$$

According to the construction of $\mathbf{H}^{1\dagger}$ and $\mathbf{H}^{2\dagger}$, we have $[\mathbf{H}^{2\dagger}(G)]_i = 1$ if $i = i^\dagger$ and 0 otherwise. And $\mathbf{H}^{2\dagger}(G') = (0, \dots, 0)$.

For $k \geq 3$, we let

$$[\mathbf{H}^{k+1\dagger}]_i = [\mathbf{H}^{k\dagger}]_i + \sum_{j \in \mathcal{N}^\star(i;\mathbf{G})} [\mathbf{H}^{k\dagger}]_j \tag{42}$$

Thus if $\overline{\overline{k}} \geq \Delta + 2$, we have $[\mathbf{H}^{\overline{\overline{k}}\dagger}(\mathbf{G})]_i \geq 1$ for any $i \in [n]$, due to the connectivity and the fact that the diameter of $\mathbf{G}$ is bounded by Δ, i.e. the propagation process described in eq. (42) reaches every node of $\mathbf{G}$. We have $[\mathbf{H}^{\overline{\overline{k}}\dagger}(G)]_i \geq 1$ for any $i \in [n]$, and $[\mathbf{H}^{\overline{\overline{k}}\dagger}(G')]_i = 0$ for any $i \in [n']$

Finally for the decoder, we let

$$\Xi^{\overline{\overline{k}}\dagger}(h) = \min(1, h) \tag{43}$$

and

$$[\hat{\mathbf{U}}^{\overline{\overline{k}}\dagger}]_i = \Xi^{\overline{\overline{k}}\dagger}(H^{\overline{\overline{k}}\dagger}_i). \tag{44}$$

We have thus constructed a function $f^\dagger$ such that $f^\dagger(\mathbf{G}) = (1, \dots, 1) \in \mathbb{R}^n$ and $f^\dagger(\mathbf{G}') = (0, \dots, 0) \in \mathbb{R}^{n'}$. Thus for any $k \in [n], k' \in [n']$, we have $[f^\dagger(\mathbf{G})]_k = 1 \neq 0 = [f^\dagger(\mathbf{G}')]_{k'}$, which concludes the proof. □

Lemma 3. *Let X, Y, Z be three metric spaces. Let $\mathcal{F} \subseteq \mathcal{C}(X, Y)$ and $\mathcal{G} \subseteq \mathcal{C}(Y, Z)$ be two sets of continuous functions. And let $\mathcal{F}^\ell \subseteq \mathcal{F}, \mathcal{G}^\ell \subseteq \mathcal{G}$ be two subsets of Lipschitz functions that are dense in $\mathcal{F}$ and $\mathcal{G}$ respectively. Then $\mathcal{G}^\ell \circ \mathcal{F}^\ell := \{g \circ f | g \in \mathcal{G}^\ell, f \in \mathcal{F}^\ell\}$ is dense in $\mathcal{G} \circ \mathcal{F}$.*

Proof. Let $g \circ f$ be a continuous function in $\mathcal{G} \circ \mathcal{F}$, $\epsilon > 0$. Due to the density of $\mathcal{G}^\ell$ in $\mathcal{G}$, there exists $g^\ell \in \mathcal{G}^\ell$ such that

$$\overline{d}(g, g^\ell) < \frac{\epsilon}{2}. \tag{45}$$

Let L_{g^ℓ} be the Lipschitz constant of g^ℓ, the density of $\mathcal{F}^\ell$ in $\mathcal{F}$ implies that there exists f^ℓ such that

$$\overline{d}(f, f^\ell) < \frac{\epsilon}{2L_{g^\ell}}. \tag{46}$$

Then we have

$$\begin{aligned} d_Z(g \circ f(x), g^\ell \circ f^\ell(x)) &\leq d_Z(g \circ f(x), g^\ell \circ f(x)) + d_Z(g^\ell \circ f(x), g^\ell \circ f^\ell(x)) && (47)\\ &< \frac{\epsilon}{2} + L_{g^\ell} d_Y(f(x), f^\ell(x)) && (48)\\ &< \frac{\epsilon}{2} + L_{g^\ell} \frac{\epsilon}{2L_{g^\ell}} = \epsilon && (49) \end{aligned}$$

for any $x \in X$. Thus $\overline{d}(g \circ f, g^\ell \circ f^\ell) < \epsilon$. Hence $\mathcal{G}^\ell \circ \mathcal{F}^\ell$ is dense in $\mathcal{G} \circ \mathcal{F}$.

□

Lemma 4. *$\mathcal{H}^{\overline{\overline{k}}}$ is dense in $\mathcal{H}^{\overline{\overline{k}}\dagger}$.*

Proof. As functions in $\mathcal{H}^{\overline{\overline{k}}}$ are composition of Lipschitz functions (neural network with linear transformation and Lipschitz activation as assumed), and all intermediate function spaces verify the Universal Approximation Property. We conclude immediately from using the definition of $\mathcal{H}^{\overline{\overline{k}}\dagger}$ and applying Lemma 3 consecutively. □

We are ready to prove Theorem 3, i.e., that $\mathcal{H}^{\overline{\overline{k}}\odot}$ satisfies the separability hypothesis of Theorem 2.

Proof. of Theorem 3

It suffices to show the separability for $\mathcal{H}^{\overline{\overline{k}}}$ since it is a subset of $\mathcal{H}^{\overline{\overline{k}}\odot}$.

Let $\mathbf{G}, \mathbf{G}' \in \text{supp}(\mathcal{D})$. According to Lemma 2, there exists $f^\dagger \in \mathcal{H}^{\overline{\overline{k}}\dagger}$ such that for any $k \in [n], k' \in [n']$, we have $[f^\dagger(\mathbf{G})]_k \neq [f^\dagger(\mathbf{G}')]_{k'}$. According to Lemma 4, there exists $f \in \mathcal{H}^{\overline{\overline{k}}}$ such that

$$\overline{d}(f^\dagger, f) < \frac{1}{3}. \tag{50}$$

Then for any $k \in [n], k' \in [n']$, we have $[f(\mathbf{G})]_k > \frac{2}{3}$ and $[f(\mathbf{G}')]_{k'} < \frac{1}{3}$. This proves the separability of $\mathcal{H}^{\overline{\overline{k}}}$ and furthermore, $\mathcal{H}^{\overline{\overline{k}}\odot}$. □

Before being able to prove Theorem 1, we need the last following lemma.

Lemma 5. *$\mathcal{H}^{\overline{\overline{k}}}$ is dense in $\mathcal{H}^{\overline{\overline{k}}\odot}$.*

Proof. We shall prove this result by explicitly constructing an approximation function in $\mathcal{H}^{\overline{\overline{k}}}$ for a given function in $\mathcal{H}^{\overline{\overline{k}}\odot}$.

Let $f^\odot \in \mathcal{H}^{\overline{\overline{k}}\odot}$, and $\epsilon > 0$. By definition of $\mathcal{H}^{\overline{\overline{k}}\odot}$ in eq. (33), there exists $S \in \mathbb{N}$, $\{T_s\}_{s \in \{1,\dots,S\}} \in \mathbb{N}^S$, as well as $\{c_{st}\} \in \mathbb{R}$ and $\{f_{st}\} \in \mathcal{H}^{\overline{\overline{k}}}$ for all (s,t) with $s \in [S], t \in [T_s]$, such that :

$$f^\odot = \sum_{s=1}^{S} \bigodot_{t=1}^{T_s} c_{st} f_{st} \tag{51}$$

Thus, for any (s,t), there exists $\overline{k}_{st} \leq \overline{\overline{k}}$, and $d_{st} \in \mathbb{N}$, such that f_{st} is composed of functions $\{\Phi^{k,s,t}_{\rightarrow,\theta}, \Phi^{k,s,t}_{\leftarrow,\theta}, \Phi^{k,s,t}_{\circlearrowleft,\theta}, \Psi^{k,s,t}_{\theta}, \Xi^{k,s,t}_{\theta}\}_{k \in [\overline{k}_{st}]}$, as defined by eq. (5)-(9) and Figure 3 in Section 3. d_{st} is the dimension of the latent states of *channel* f_{st}.

The different channels can have different number of propagation updates $\overline{k}_{st}$, but they are all bounded by $\overline{\overline{k}}$. Without loss of generality, we can assume that all $\overline{k}_{st}$ are equal to $\overline{\overline{k}}$ by padding, when needed, exactly $\overline{\overline{k}} - \overline{k}_{st}$ null operations $\Phi^k_{\rightarrow}$, $\Phi^k_{\leftarrow}$ and Ψ^k before the actual ones.

Let $d = \sum_{s=1}^{S} \sum_{t=1}^{T_s} d_{st}$ be the cumulated dimensions of the different channels.

For each (s,t), we introduce the matrix $W_{st} \in \{0,1\}^{d_{st} \times d}$ which is defined by:

$$[W_{st}]_{ij} = \begin{cases} 1, & \text{if } \sum_{s'=1}^{s} \sum_{t'=1}^{T_{s'}} d_{s't'} + \sum_{t'=1}^{t-1} d_{st'} + i = j \\ 0, & \text{otherwise.} \end{cases} \tag{52}$$

Thus $W_{st} = [0, \ldots, 0, I_{d_{st}}, 0, \ldots, 0]$. Basically, when given a vector of dimension d, W_{st} will be able to select exactly the component that corresponds to the channel (s,t), and will thus return a vector of dimension d_{st}.

Let us now define the functions $\{\Phi^k_{\rightarrow,\theta}, \Phi^k_{\leftarrow,\theta}, \Phi^k_{\circlearrowleft,\theta}, \Psi^k_\theta, \Xi^k_\theta\}_{k \in [\overline{\overline{k}}]}$ such that

$$\Phi^k_{\rightarrow,\theta}(H_i^{k-1}, A_{ij}, H_j^{k-1}) = \sum_{s=1}^{S} \sum_{t=1}^{T_s} W_{st}^\top . \Phi^{k,s,t}_{\rightarrow,\theta}(W_{st}.H_i^{k-1}, A_{ij}, W_{st}.H_j^{k-1}) \tag{53}$$

$$\Phi^k_{\leftarrow,\theta}(H_i^{k-1}, A_{ij}, H_j^{k-1}) = \sum_{s=1}^{S} \sum_{t=1}^{T_s} W_{st}^\top . \Phi^{k,s,t}_{\leftarrow,\theta}(W_{st}.H_i^{k-1}, A_{ij}, W_{st}.H_j^{k-1}) \tag{54}$$

$$\Phi^k_{\circlearrowleft,\theta}(H_i^{k-1}, A_{ij}) = \sum_{s=1}^{S} \sum_{t=1}^{T_s} W_{st}^\top . \Phi^{k,s,t}_{\circlearrowleft,\theta}(W_{st}.H_i^{k-1}, A_{ij}) \tag{55}$$

$$\Psi^k_\theta(H_i^{k-1}, B_i, \phi^k_{\rightarrow,i}, \phi^k_{\leftarrow,i}, \phi^k_{\circlearrowleft,i}) = \sum_{s=1}^{S} \sum_{t=1}^{T_s} W_{st}^\top . \Psi^{k,s,t}_\theta(W_{st}.H_i^{k-1}, B_i, W_{st}.\phi^k_{\rightarrow,i}, W_{st}.\phi^k_{\leftarrow,i}, W_{st}.\phi^k_{\circlearrowleft,i}) \tag{56}$$

These functions, using eq. (5)-(8), define a function acting on a latent space of dimension d. Moreover, for any channel (s,t) and any node $i \in [n]$, we have $W_{st}.H_i^{\overline{\overline{k}}} = H_i^{\overline{\overline{k}},s,t}$.

We have thus built a function of $\mathcal{H}^{\overline{\overline{k}}}$ that exactly replicates the steps performed on the different channels. Now, let us take a closer look at the decoding step.

Observing that the mapping from $\mathbb{R}^d$ to $\mathbb{R}^{d_U}$, $h \mapsto \sum_{s=1}^{S} \bigodot_{t=1}^{T_s} c_{st} \Xi_\theta^{\overline{\overline{k}},s,t}(W_{st}h)$ is indeed continuous, there exists a mapping $\Xi_\theta^{\overline{\overline{k}}} \in \mathcal{H}_d^{d_U}$ such that :

$$\|\Xi_\theta^{\overline{\overline{k}}}(h) - \sum_{s=1}^{S} \bigodot_{t=1}^{T_s} c_{st} \Xi_\theta^{\overline{\overline{k}},s,t}(W_{st}h)\| \leq \epsilon \tag{57}$$

for any h in a compact of $\mathbb{R}^d$. The resulting function $f \in \mathcal{H}^{\overline{\overline{k}}}$, composed of $\{\Phi^k_{\rightarrow,\theta}, \Phi^k_{\leftarrow,\theta}, \Phi^k_{\circlearrowleft,\theta}, \Psi^k_\theta, \Xi^k_\theta\}_{k \in [\overline{\overline{k}}]}$ using eq. (5)-(9), approximates $f^\odot$ with precision less than ϵ, which concludes the proof.

□

We now have all necessary ingredients to prove Theorem 1.

Proof. According to the hypotheses of compactness and permutation-invariance on $\text{supp}(\mathcal{D})$, both conditions of Theorem 2 are satisfied by $\text{supp}(\mathcal{D})$. Consider the subalgebra $\mathcal{H}^{\overline{\overline{k}}\odot}$ defined by eq. (33). According to the hypothesis of separability of external inputs, the hypothesis of connectivity and Theorem 3, $\mathcal{H}^{\overline{\overline{k}}\odot}$ satisfies the separability and self-separability conditions of Theorem 2. Applying Theorem 2, it comes that $\mathcal{H}^{\overline{\overline{k}}\odot}$ is dense in $\mathcal{C}_{\text{eq.}}(\text{supp}(\mathcal{D}))$. Then according to Lemma 5, $\mathcal{H}^{\overline{\overline{k}}}$ is dense in $\mathcal{H}^{\overline{\overline{k}}\odot}$. We conclude that $\mathcal{H}^{\overline{\overline{k}}}$ is dense in $\mathcal{C}_{\text{eq.}}(\text{supp}(\mathcal{D}))$ by the transitivity property of density. □

B.4 Proof of Corollary 1

Proof. Let $\epsilon > 0$. From Property 1, $\mathbf{U}^*$ is permutation-equivariant. Moreover, by hypothesis, $\mathbf{U}^*$ is continuous. Thus $\mathbf{U}^* \in \mathcal{C}_{\text{eq.}}(\text{supp}(\mathcal{D}))$.

And from Theorem 1, we know that there exists a function $Solver_\theta \in \mathcal{H}^{\Delta+2}$ such that

$$\forall \mathbf{G} \in \text{supp}(\mathcal{D}), \|Solver_\theta(\mathbf{G}) - \mathbf{U}^*(\mathbf{G})\| \leq \epsilon \tag{58}$$

□

C Linear Systems derived from the Poisson Equation

This appendix details the experiments of Section 5.1: it presents the data generation process, and also explains the change of variables that was made to help normalizing the data (not mentioned in the main paper for space reason, as it does not change the overall conclusions of the experiments). Finally, we also discuss an additional super generalization experiment briefly cited in the paper.

C.1 Data generation

Initial problem Consider a Poisson's equation with Dirichlet condition on its boundary $\partial\Omega$:

$$-\triangle u = f \; in \; \Omega$$

$$u|_{\partial\Omega} = g$$

where Ω a spatial domain in $\mathbb{R}^2$, and $\partial\Omega$ its boundaries. The right hand side f is defined on Ω, and the Dirichlet boundary condition g is defined on $\partial\Omega$. x and y will denote the classical 2D coordinates.

Random geometries Random 2D domains Ω are generated from 10 points, randomly sampled in the unit square. The Bézier curve that passes through these pints is created, and is further subsampled to obtain approximately 100 points in the unit square. These points defines a polygon, that is used as the boundary $\partial\Omega$. See the left part of Figure 7 to see four instances.

Random f and g Functions f and g are defined by the following equations:

$$f(x,y) = r_1(x-1)^2 + r_2y^2 + r_3, \qquad (x,y) \in \Omega \tag{59}$$

$$g(x,y) = r_4x^2 + r_5y^2 + r_6xy + r_7x + r_8y + r_9, \qquad (x,y) \in \partial\Omega \tag{60}$$

in which parameters r_i are uniformly sampled between -10 and 10.

Discretization The random 2D geometries are discretized using Fenics' standard mesh generation method (see Figure 7-right).

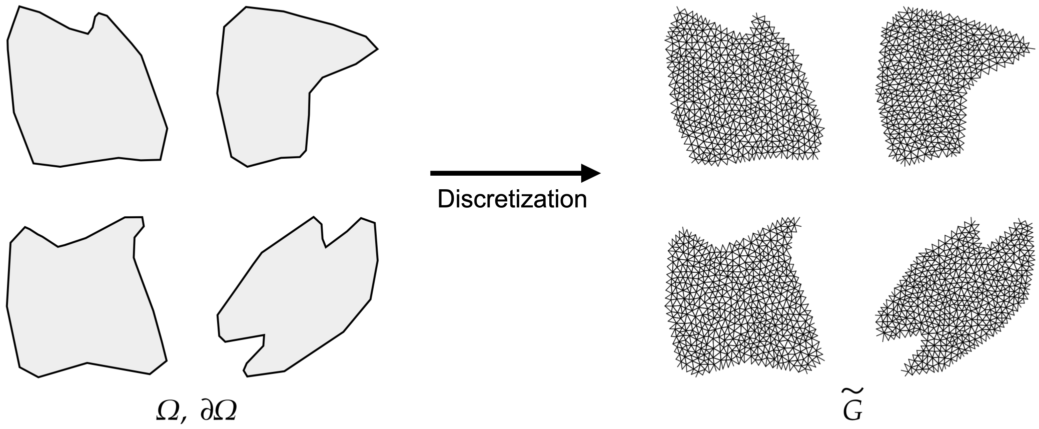

Figure 7: **Discretization of randomly generated domains**

Assembling The assembling step [28] consists in building a linear system from the partial differentiate equation and the discretized domain. The unknown are the values of the solution at the nodes of the mesh, and the equations are obtained by using the variational formulation of the PDE on basis functions with support in the neighbors of each node. This is also automatically performed using Fenics. The result of the assembling step is a square matrix $\mathbf{A}$ and a vector $\mathbf{B}$, and the solution is the vector $\mathbf{U}$ such that $\mathbf{AU} = \mathbf{B}$. Thus, as stated in Section 5, in the framework of SSPs, an Interaction Graph is defined from the number of nodes of the mesh, the matrix $\mathbf{A}$ and the vector $\mathbf{B}$, and the loss function is:

$$\ell(\mathbf{U}, \mathbf{G}) = \sum_{i \in [n]} (-B_i + \sum_{j \in [n]} A_{ij} U_j)^2 \tag{61}$$

C.2 Change of variables

Being able to properly normalize the input data of any neural network is a critical issue, and failing to do so can often lead to gradient explosions and other training failures (more details on data normalization in AppendixC.2). In the Poisson case study, the nodes at the boundary are constrained (i.e. $A_{ii} = 1$ and $A_{ij} = 0$ if $i \neq j$), and the interior nodes are not. Moreover, the coefficients of matrix $\mathbf{A}$ at these interior nodes satisfy a conservation equality (i.e. $A_{ii} = -\sum_{j \in [n] \setminus \{i\}} A_{ij}$). As a consequence, the distributions of their respective B_i are very different, sometimes even with different orders of magnitude. It is then almost impossible to properly normalize those multimodal distribution.

In order to tackle this issue, we consider the following change of variable, changing $\mathbf{A}, \mathbf{B}$ to $\mathbf{A}', \mathbf{B}'$, and modifying the loss function accordingly. For $\mathbf{B}$, we set the dimension $d_{B'}$ of $\mathbf{B}'$ to 3 as follows:

$$B'_i = \begin{cases} [B_i, 0, 0] \text{ if node } i \text{ is not constrained} \\ [0, 1, B_i] \text{ otherwise} \end{cases} \tag{62}$$

The B_i's for constrainted and unconstrainted nodes will hence be normalized independently.

Moreover, the information stored in the matrix $\mathbf{A}$ is rather redundant. As mentioned, for constrained nodes $A_{ii} = 1$ and $A_{ij} = 0$ if $i \neq j$, whereas for unconstrained nodes $A_{ii} = -\sum_{j \in [n] \setminus \{i\}} A_{ij}$. Hence the diagonal information can always be retrieved from $\mathbf{B}$ and the non diagonal elements of $\mathbf{A}$. We thus choose the following change of variable:

$$A'_{ij} = \begin{cases} A_{ij} \text{ if } i \neq j \\ 0 \text{ otherwise} \end{cases} \tag{63}$$

Finally, the loss function is transformed into the following function ℓ' (where $B_i'^p$ denotes the p^{th} component of vector B_i):

$$\ell'(\mathbf{U}, \mathbf{G}') = \sum_{i \in [n]} \left((1 - B_i'^2)(-B_i'^1) + B_i'^2(U_i - B_i'^3) + \sum_{j \in [n]} A'_{ij}(U_j - U_i) \right)^2 \tag{64}$$

One can easily check that this change of variables and of loss function defines the exact same optimization problem as in eq. (10), while allowing for an easier normalization, as well as a lighter sparse storage of $\mathbf{A}$.

C.3 Additional super generalization experiment

This appendix describes a second experiment regarding super-generalization. Figure 8 displays the results of the DSS model, learned without any noise, when increasing noise is added to the test examples, more and more diverging from the distribution of the training set (the graph size remains unchanged). Log-normal noise is applied to $\mathbf{A}$ ($A_{ij} \exp(\mathcal{N}(0, \tau))$, and normal noise to $\mathbf{B}$ ($B_i \mathcal{N}(1, \tau)$, for different values of noise variance τ. The correlation between the results of DSS and the 'ground truth', here given by the results of LU (solving the same noisy system). But although DSS results remain highly correlated with the ground truth for small values of

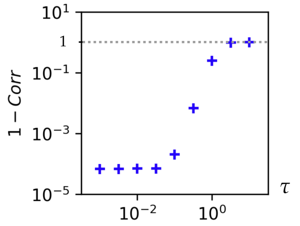

Figure 8: **Increasing noise variance** τ: Correlation (DSS, LU)

τ, they become totally uncorrelated for large values of τ (correlation close to 0): DSS has learned something specific to the distribution $\mathcal{D}$ of linear systems coming from the discretized Poisson EDP. Further work will extend these results, analyzing in depth the specifics of the learned models.

D Power systems

This appendix gives more details about the AC power flow problem, and how it is converted into the DSS framework.

The AC power flow equations model the steady-state behavior of transportation power grids. They are an essential part of both real-time operation and long-term planning. A thorough overview of the domain is provided in [32].

Let's consider a power grid with n nodes. The voltage at every electrical node is a sinusoid that oscillates at the same frequency. However, each node has a distinct module and phase angle. Thus, we define the complex voltage at node i, $V_i = |V_i|e^{\mathbf{j}\theta} \in \mathbb{C}$ (where $\mathbf{j}$ is the imaginary unit).

The admittance matrix $\mathbf{Y} = (Y_{ij})_{i,j\in[n]}; Y_{ij} \in \mathbb{C}$ defines the admittance of each power line of the network. The smaller $|Y_{ij}|$, the less nodes i and j are coupled. For $i, j \in [n]$, the coefficient Y_{ij} models the physical characteristics of the power line between nodes i and j (i.e. materials, length, etc.).

At each node i, there can be power consumption (houses, factories, etc.). The real part of the power consumed is denoted by $P_{d,i}$ and the imaginary part by $Q_{d,i}$. The subscript d stands for “demand“. Additionally, there can also be power production (coal or nuclear power plants, etc.). They are very different from consumers, because they constrain the local voltage module. They are defined by $P_{g,i}$ and $V_{g,i}$. The subscript g stands for “generation“. Nodes that have a producer attached to it are called “PV buses“ and are denoted by $I_{PV} \subset [n]$. The nodes that are not connected to a production are called “PQ buses“ and are denoted by $I_{PQ} \subset [n]$.

Moreover, one has to make sure that the global energy is conserved. There are losses at every power line that are caused by Joule's effect. The amount of power lost to Joule's effect being a function of the voltage at each node, it cannot be known before the voltage computation itself. Thus, to make sure that the production of energy equals the consumption plus the losses caused by Joule's effect, we need to be able to increase the power production accordingly. In this work we use the common “slack bus“approach which consists in increasing the production of a single producer so that global energy conservation holds. This node is chosen beforehand and we denote it by $i_s \in [n]$.

Thus the system of equations that govern the power grid is the following:

$$\forall i \in [n] \setminus \{i_s\}, \quad P_{g,i} - P_{d,i} = \sum_{j\in[n]} |V_i||V_j|(\text{Re}(Y_{ij})\cos(\theta_i - \theta_j) + \text{Im}(Y_{ij})\sin(\theta_i - \theta_j)) \tag{65}$$

$$\forall i \in I_{PQ}, \quad -Q_{d,i} = \sum_{j\in[n]} |V_i||V_j|(\text{Re}(Y_{ij})\sin(\theta_i - \theta_j) - \text{Im}(Y_{ij})\cos(\theta_i - \theta_j)) \tag{66}$$

$$\forall i \in I_{PV}, \quad |V_i| = V_{g,i} \tag{67}$$

The encoding into our framework requires a bit of work. For the coupling matrix we use $d_A = 2$ and $A_{ij} = [\text{Re}(Y_{ij}), \text{Im}(Y_{ij})]$. For the local input we take $d_B = 5$ and $B_i = [P_{g,i} - P_{d,i}, Q_{d,i}, 1(i \in I_{PQ}), V_{g,i}, 1(i = i_s)]$. Finally, for the state variable we use $d_U = 2$ and take $U_i = [|V_i|, \theta_i]$.

Taking the squared residual of eq. (65)-(67) and taking the sum over every node, we obtain the loss of eq. (11).

E Further implementation details

In this section we detail the implementation details that were made to robustify the training of the DSS. None of those changes alter the properties of the architecture.

Correction coefficient We introduce a parameter α that modifies eq. (42) in the following way:

$$H_i^k = H_i^{k-1} + \alpha \times \Psi_\theta^k(H_i^{k-1}, B_i, \phi_{\rightarrow,i}^k, \phi_{\leftarrow,i}^k, \phi_{\circlearrowleft,i}^k) \tag{68}$$

Choosing a sufficiently low value of α, helps to keep the successive $\overline{k}$ updates at reasonably low orders of magnitude.

Injecting existing solutions Depending on the problem at hand, it may be useful to initialize the predictions to some known value. This acts as an offset, that can help the training process to start not too far from the actual solutions. This offset is applied identically at every node, thus not breaking the permutation-equivariance of the architecture:

$$\widehat{U}_i^k = U_{offset} + \Xi_\theta^k(H_i^k) \tag{69}$$

For instance, in the power systems application, it is known that the voltage module is commonly around 1.0, while the voltage angle is around 0. Thus we used $U_{offset} = [1, 0]$ (keeping in mind that $d_U = 2$). On the other hand, in the linear systems application, there is no reason to use such an offset, so we used $U_{offset} = [0]$ (keeping in mind that here $d_U = 1$). But in several contexts, there exists some fast inaccurate method that can give an approximate solution closer to the final one than $(0, \ldots, 0)$.

Data normalization In addition to a potential change of variables (which helps disentangle multimodal distributions of the input data, see Appendix C.2), it is also critical to normalize the input Interaction Graphto help with the training of neural networks. Each function $\Phi_{\rightarrow,\theta}^k$, $\Phi_{\leftarrow,\theta}^k$ and $\Phi_{\circlearrowleft,\theta}^k$ take A_{ij} as input, and the functions Ψ_θ^k take b_i as input. We thus introduce hyperparameters $\mu_A, \sigma_A \in \mathbb{R}^{d_A}$ and $\mu_B, \sigma_B \in \mathbb{R}^{d_B}$ are used to create a normalized version of the data:

$$a_{ij} = \frac{A_{ij} - \mu_A}{\sigma_A} \tag{70}$$

$$b_i = \frac{B_i - \mu_B}{\sigma_B} \tag{71}$$

$\mathbf{g} = (\mathbf{a}, \mathbf{b})$ (with $\mathbf{a} = (a_{ij})_{i,j \in [n]}$ and $\mathbf{b} = (b_i)_{i \in [n]}$) is thus the normalized version of $\mathbf{G}$. We apply the DSS to this normalized $\mathbf{g}$ and consider the loss $\ell(Solver_\theta(\mathbf{g}), \mathbf{G})$ instead of $\ell(Solver_\theta(\mathbf{G}), \mathbf{G})$.

Gradient clipping We sometimes observed (e.g., in the power systems experiments) some gradient explosions. The solution we are currently using is to perform some gradient clipping. Further work should focus on facilitating this training process automatically.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have mentioned main claims in Section 1: Introduction, calling them "contributions".

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have indicated the limitations in the conclusion section, as future work.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We have provided clearly the hypotheses and a proof sketch in the main text, and detailed proofs in appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We have provided all details necessary to reproduce the results in the "Experimental conditions" section.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our code is made available in the supplementary materials, and links to the datasets can be found as references. In this version, to protect anonymization, we are not publishing the code on our GitHub repo, but this will be done for the final version.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The "Experimental conditions" section provides all details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes] .

Justification: We provide 10th and 90th percentiles.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute Resources are explained in the "Experimental conditions" section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We comply with the NeurIPS code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: This is a fundamental research paper. Discussing broader impact in beyond the scope of the papers.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our models do not present such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our data and code are well documented in supplemental material.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our research does not involve any human.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our research does not involve human subjects or ethical issues.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.