An Investigation of Physics Informed Neural Networks to solve the Poisson-Boltzmann Equation in Molecular Electrostatics

A. Author,[1, a)] B. Author,[1] and C. Author[2, b)]
[1)]*Authors' institution and/or address*
[2)]*Second institution and/or address*

(*Electronic mail: Second.Author@institution.edu.)

(Dated: 20 September 2024)

Abstract goes here

I. INTRODUCTION

Implicit solvation is a widely used approach in molecular modeling and considers the solvent as a continuum. This massively decreases the number of degrees of freedom of the system, and hence, the computational cost of mean-field calculations. In electrostatics, a widely used implicit method is the Poisson-Boltzmann equation (PBE)[1,2] solved on a multi-region domain, where the dielectric constant and salt concentration have a sharp variation along interfaces. The PBE has been solved numerically for decades using finite differences (FDM)[3–7], finite elements (FEM)[8–11], boundary elements (BEM)[12–19], (semi) analytical[20–23], and hybrid approaches[24–26], proving useful for a large community.

Scientific machine learning (SciML) has seen tremendous recent interest in utilizing tools from computer science, mathematics and statistics to solve complex scientific and engineering problems. SciML encompasses a wide array of methodologies such as digital twins[27], data assimilation[28], Bayesian inverse analysis[29], model reduction[30], Physics-informed machine learning[31] etc. In particular, a prominent and widely used physics-informed machine learning approach to solve Partial Differential Equations (PDEs) is the physics-informed neural networks (PINN).[32–34] PINN represents the approximate solution of the differential equation with a neural network, where the network's parameters are optimized using a loss function that contains the PDE residual. PINN has attracted a lot of interest from the computational mathematics community, and has been used in various applications, such as fluid mechanics,[35] heat transfer,[36] electromagnetism,[37] acoustics,[38] Lie-Poisson systems,[39] among others.

The definition of the residual gives rise to different variations of PINN, for example, by writing it in variational[40,41] or boundary integral[42,43] form. There are also special forms of PINN that are designed for domain decomposition, for example, extended PINN (XPINN),[44] conservative PINN (cPINN),[45] and distributed PINN (dPINN),[46] among others. These usually use one neural network per region, and they differ in the definition of the loss functions and the treatment of the interface conditions. These methods have been adapted to solve linear elliptic partial differential equations with interfaces[47–53], where domain decomposition can enhance precision by accounting for particularities of the solution in each subdomain. Additionally, there are theoretical studies that analyze their convergence[54] and error bounds[52] for elliptic interface problems. Although these studies vary in their choice of optimization algorithms, parameters, interface conditions, and neural network architectures, they all use a domain decomposition PINN strategy, and we employ a similar strategy in devising a PINN for the PBE.

The Poisson-Boltzmann model considered in this work is an example of an elliptic interface problem. Domain decomposition PINN methods (and others) have been recently applied to the PBE to compute electrostatic potentials and energies in molecular settings,[49,50,55,56] similar to the present work. For example, Li *et al.*,[55] proposed a PINN approach with variational principles to solve the PBE, through a multi-scale deep neural network. In that case, there is only a single neural network, whereas our work explicitly captures the interface by using a multi-domain approach. Also, the methodology in the work by Wu and Lu[49] is based on using an extended multiple-gradient descent (MGD)[57] algorithm in a multi-domain setting. This may be a difficuly, as optimizers like MGD are not readily available in common software packages like Tensor Flow[58] or Torch[59]. In this work, we devel-

[a)]Also at Physics Department, XYZ University.
[b)]http://www.Second.institution.edu/~Charlie.Author.

oped a PINN method for the PBE utilizing the common Adam optimizer[60]. Moreover, Wu and Lu define the solute-solvent interface with the van der Waals surface,[61] as opposed to the solvent-excluded surface (SES),[?] which prevails in PBE calculations, and is used in the present work. An alternative approach to solve the PBE based is based on IONet[56], which is a neural network that learns the differential operator with an interface. However, this technique requires training with previously computed numerical solutions. On the other hand, Ying and co-workers[50] presented the multi-scale fusion network (MSFN) approach, that relies on sub-networks that approximate the solution at different frequencies (or scales), and use a least-square type loss function.

There are other physics informed machine learning techniques that have been applied to molecular electrostatics and involve the PBE, but are not designed to solve it. For example, PBML[62] is a neural network model that was trained with a large dataset of PB solutions, and provides the solvation free energy from the molecular structure only. Similarly, pyPKa[63] uses PB calculations to feed a machine learning model that estimates pKas.

Regardless of the important progress of PINN for the PBE, further efforts are needed to understand the impact of different variants offered by PINN towards its applicability in practical cases. The present work intends to fill this gap by extensively testing different sampling strategies, architecture improvements, loss function balancing techniques, etc., to suggest good practices of PINN for the PB equation, and identify pressing challenges moving forward. Along with this analysis, we provide a TensorFlow-based Python code called XPPBE[64] that computes electrostatic potentials and energies from a PDB structure using an easy interface, for interested application scientists.

We note that there are numerous traditional methods for solving the PB equation (FDM, FEM, BEM, etc.), whereas the aim of this work is to explore the recently developed ML techniques. As such, we leave a direct comparison with the traditional solvers for future work. Currently, PINN is usually slower than standard numerical schemes in most applications,[65,66] however, they have great potential to improve their performance, as it has happened in complex problems like weather modeling[67,68] where they see a 10,000× speedup.

In this work we focus on solving the standard PBE with PINN, however, we believe this approach can pave the way for future advancements of molecular electrostatics simulation, distinguishing itself from traditional methods. For instance, PINN can be used to solve inverse problems[69], where physical properties, like permittivity, are learneable parameters. Also, PINN gives the possibility to easily incorporate experimental measurements or information from molecular dynamics simulations through a loss function, opening new doors in model development.

Probably we can adopt a reference standard, we repeat a lot of refs. Could be better to use the google scholar standard?

II. METHODS

A. The Poisson-Boltzmann equation

When a solute is immersed in a continuum solvent, it can be considered as a low-dielectric cavity (with relative permitivity? ε_m) with delta-like partial charges inside an infinite domain with the solvent's physical properties, as sketched by Fig. **??**. In biological settings, the solvent is usually water ($\varepsilon_w \approx$80) with salt ions that move in response to an external electric field. In equilibrium, the free ions arrange according to Boltzmann statistics, giving rise to the Poisson-Boltzmann equation. Following Fig. **??**, the electrostatic potential (ϕ) is modeled from the following coupled system of PDEs: eps=80 correspond to 80*eps0 (or probably refer to the non-dimensional eps or something like that?). Also, what do you refer for partial charges? That means factoring the charge of the electron? (in that depends if e is squared or not in the next equation) (same for including or not eps0 in the equation)

$$\begin{aligned}
-\varepsilon_m \nabla^2 \phi^{(m)}(\mathbf{x}) &= \frac{e}{k_B T} \sum_{i=1}^{n_c} q_i \delta\left(\mathbf{x}-\mathbf{x}_i\right), \quad \mathbf{x} \in \Omega_m \\
-\nabla^2 \phi^{(w)}(\mathbf{x}) + \kappa_w^2 \sinh\left(\phi^{(w)}\right)(\mathbf{x}) &= 0, \quad \mathbf{x} \in \Omega_w \\
\phi^{(m)}(\mathbf{x}) &= \phi^{(w)}(\mathbf{x}), \quad \mathbf{x} \in \Gamma \\
\varepsilon_m \partial_n \phi^{(m)}(\mathbf{x}) &= \varepsilon_w \partial_n \phi^{(w)}(\mathbf{x}), \quad \mathbf{x} \in \Gamma
\end{aligned} \tag{1}$$

where q_i are the partial charges in the solute molecule, κ_w is the inverse of the Debye length, and the electrostatic potential ϕ what is ϕ? In the eqs we hve ϕ^m etc. is nondimensionalized by $\frac{k_B T}{e}$ (Boltzmann constant times temperature, divided by the elementary charge). The interface Γ is usually defined either as the solvent accessible,[?] solvent excluded,[?]

van der Waals? , or Gaussian? surfaces. In this work, we use the solvent-excluded surface (SES).

Eq. (1) can be difficult to solve numerically because of the singularities at the charge's locations. There are several regularization techniques of the PB equation[9,70? ,71] that alleviate this issue, for example, by solving for a so-called *regular* or *reaction* potential which is a difference between the electrostatic potential and the Coulomb potential in some region of the domain. The Coulomb potential corresponding to the point charges to the total potential (nondimensionalized) is,

$$g(\mathbf{x}) = \frac{k_B T}{e} \sum_{i}^{n_c} \frac{q_i}{4\pi\varepsilon_m |\mathbf{x}-\mathbf{x}_i|} \tag{2}$$

In our case, we use a regularized version of the PB equation that only decomposes the electrostatic potential into singular and non-singular components in Ω_m. That is, we set $\psi^{(m)} = \phi^{(m)} - g$ in Ω_m. Moreover, in many practical applications, such as proteins, linearising the the hyperbolic sine in the PB equation yields acceptable results. leading to the linear Regularized Poisson-Boltzmann (RPB) Equation, probably maintain also the C potential nondimensionalized, so avoid confusing. Here we can discuss if we maintain everything with dimension or not, but i think that we must be consistent in the usage of *phi* with or with not dimension

$$\begin{aligned}
\nabla^2 \psi^{(m)}(\mathbf{x}) = 0 \ \mathbf{x} \ &\in \ \Omega_m \\
-\nabla^2 \phi^{(w)}(\mathbf{x}) + \kappa_w^2 \phi^{(w)}(\mathbf{x}) = 0 \quad \mathbf{x} \ &\in \ \Omega_w \\
\psi^{(m)}(\mathbf{x}) + g(\mathbf{x}) = \phi^{(w)}(\mathbf{x}) \ \mathbf{x} \ &\in \ \Gamma \\
\varepsilon_m \left(\partial_n \psi^{(m)}(\mathbf{x}) + \partial_n g(\mathbf{x})\right) = \varepsilon_w \partial_n \phi^{(w)}(\mathbf{x}) \ \mathbf{x} \ &\in \ \Gamma
\end{aligned} \tag{3}$$

The linear PB formulation is widely used, however, it may lead to large errors in highly charged systems, such as nucleic acids.

Boundary condition in here? or in the loss function

Talk about free energy? Same as before. ϕ will denote the dimensionless potential? so the free energy will have to be stated in terms of dimensionless potential? I dont know how to be consistent on it

Analytical solutions?

B. Neural Networks and PINNs

A Neural Network in the context of a PINN may be considered a function $\mathcal{N}$, parameterized by parameters θ from inputs $\mathbf{x} \in \mathbb{R}^m$ to outputs $\mathbb{R}^n$, where m and n denote the input and output dimensions. There are numerous neural network architectures e.g. Multi-layer Perception (MLP), Convolutional Networks, Recurrent Networks etc[72]. Additionally, there are numerous options for both defining the solution of the PDE using a neural network and also for defining the loss function. One simple setting of using PINN to solve a PDE is to employ a MLP with a mean-squared-error (MSE) loss function and representing the PDE solution by the output of the neural network. This simple setting often forms the basic building block for solving a PDE using PINN and we describe it next.

Given a so called activation function $\sigma : \mathbb{R} \to \mathbb{R}$, we define $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ by defining its ith component as $[\sigma(\mathbf{x})]_i = \sigma(\mathbf{x}_i)$ for $\mathbf{x} \in \mathbb{R}^d$. That is, the activation function is defined component-wise on the input vector. Common choices of the activation function are ReLU, tanh, sigmoid etc. Given a set of $L+1$ integers $d_i, i = 0, \ldots, L$, an L-layer MLP or Feed-Forward Neural network is then defined as $\mathcal{N}(\mathbf{x};\theta) = \mathbf{x}^{(L)}$, where the output of the ith layer $\mathbf{x}^{(i)}$ is defined recursively as

$$\mathbf{x}^{(i)} = \begin{cases} \mathbf{x}, & i = 0 \\ \sigma(W^{(i)}\mathbf{x}^{(i-1)} + b^{(i)}), & i = 1, \ldots, L-1 \\ W^{(L)}\mathbf{x}^{(L-1)} + b^{(L)}, & i = L \end{cases} \tag{4}$$

Here the weights $W^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and the biases $b^{(i)} \in \mathbb{R}^{d_i}$ form the set of parameters θ for the MLP.

Let $L(u) = f$ denote a generic PDE operator on a domain Ω, with boundary conditions $\mathscr{B}(u) = g$, for appropriate functions u,f and g. Then to solve a PDE using a neural network like the MLP is to form a loss function of the form,

$$\frac{1}{N_\Omega} \sum_{\mathbf{x}_i \in S_\Omega} |L(\mathcal{N}(\mathbf{x}_i;\theta)) - f(\mathbf{x}_i)|^2 + \frac{1}{N_{\partial\Omega}} \sum_{\mathbf{x}_i \in S_{\partial\Omega}} |B(\mathcal{N}(\mathbf{x}_i;\theta)) - g(\mathbf{x}_i)|^2. \tag{5}$$

Here the $S_\Omega \subset \Omega$ and $S_{\partial\Omega} \subset \partial\Omega$ are sets containing $N_\Omega$domain and $N_{\partial\Omega}$ boundary collocation points respectively, whereas $|\cdot|$ refers to the Euclidean norm. Such a loss function is often referred to as the MSE loss. This loss function is then minimized, usually by employing a variant of a gradient descent method e.g. SGD, Adams etc[72]. The convergence properties of PINN in this framework is studied in[73].

C. A Basic PINN for RPB

In this section we outline the neural network architecture, the loss function and details on the construction of collocation points to develop a basic PINN for the RPB. Later on in Section II D we improve on the basic PINN with a series of improvements.

1. A multidomain neural network architecture

Several recent works have applied Physics-Informed Neural Networks (PINN) to solve elliptic partial differential equations (PDEs) with interfaces[47–53]. These studies commonly suggest using two independent neural networks, each approximating the solution in a specific subdomain. The interface between the subdomains is handled by adding an additional term in the loss function, ensuring consistency across the boundary. Moreover, convergence and error bounds for such approximations have been explored[52,54].

In this work, we follow a similar approach but with certain modifications tailored to the regularized Poisson-Boltzmann Equation given in (3). Specifically, we design a single neural network architecture with two independent branches[74]. Each branch is responsible for approximating the electrostatic potential within its respective subdomain. One branch computes the reaction potential within the solute domain ($\mathcal{N}_m$), while the other estimates the total potential in the solvent domain ($\mathcal{N}_w$). At the simplest level, these branches are a simple MLP, however, we outline many improvements to this basic architecture in section II D. It is important to note that these branches output different types of potentials due to the specific form of the regularized PBE used in this work.

$$\begin{aligned}\psi_\theta^{(m)} &= \mathcal{N}_m(\mathbf{x};\boldsymbol{\theta}_m)\\ \phi_\theta^{(w)} &= \mathcal{N}_w(\mathbf{x};\boldsymbol{\theta}_w)\end{aligned} \tag{6}$$

Discuss equality in above Here, $\mathcal{N}_m(\mathbf{x};\boldsymbol{\theta}_m)$ and $\mathcal{N}_w(\mathbf{x};\boldsymbol{\theta}_w)$ represent the outputs of the solute and solvent branches, respectively, and $\boldsymbol{\theta}_m$ and $\boldsymbol{\theta}_w$ are their corresponding trainable parameters. The trainable parameters for the neural network is thus $\boldsymbol{\theta} = \boldsymbol{\theta}_m \cup \boldsymbol{\theta}_w$. This setup allows for the approximation of the total potential at any point within the domain as a combination of these two solutions,

$$\phi_\theta(\mathbf{x}) = \begin{cases} \phi_\theta^{(m)} = g(\mathbf{x}) + \mathcal{N}_m(\mathbf{x};\boldsymbol{\theta}_m) & \mathbf{x}\in\Omega_m \\ \phi_\theta^{(w)} = \mathcal{N}_w(\mathbf{x};\boldsymbol{\theta}_w) & \mathbf{x}\in\Omega_w \\ \dfrac{\phi_\theta^{(m)} + \phi_\theta^{(w)}}{2} & \mathbf{x}\in\Gamma \end{cases} \tag{7}$$

An alternative to a single neural network is to employ two (independent) neural networks having parameters $\boldsymbol{\theta}_m$ and $\boldsymbol{\theta}_w$. The rationale behind using a single neural network with a unified set of parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_m \cup \boldsymbol{\theta}_w$ (instead of two independent neural networks) is that this allows us to employ only one optimizer to minimize the loss function. We have observed that this approach leads to better convergence when solving this type of problem.

2. Loss function

The loss function $L(\boldsymbol{\theta};S)$ depends on the parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_m \cup \boldsymbol{\theta}_w$ and the set of collocation points S. The set S is divided into subsets corresponding to specific regions: $S = S_{\Omega_m} \cup S_{\Omega_w} \cup S_\Gamma \cup S_{\partial\Omega}$. Here, S_{Ω_m} and S_{Ω_w} represent the collocation points in the solute and solvent domains, respectively, while S_Γ refers to the points at the interface, and $S_{\partial\Omega}$ corresponds to the points on the boundary of the solvent domain. Let N_j denote the count of each subset S_j. The loss function is,

$$\begin{aligned}L(\boldsymbol{\theta};S) = {} & w_{\Omega_m} L_{\Omega_m}(\boldsymbol{\theta}_m;S_{\Omega_m}) + w_{\Omega_w} L_{\Omega_w}(\boldsymbol{\theta}_w;S_{\Omega_w})\\ & + w_{\partial\Omega} L_{\partial\Omega}(\boldsymbol{\theta}_w;S_{\partial\Omega}) + w_{\Gamma_D} L_{\Gamma_D}(\boldsymbol{\theta}_m,\boldsymbol{\theta}_w;S_\Gamma)\\ & + w_{\Gamma_N} L_{\Gamma_N}(\boldsymbol{\theta}_m,\boldsymbol{\theta}_w;S_\Gamma)\end{aligned} \tag{8}$$

where

$$L_{\Omega_m}(\boldsymbol{\theta}_m; S_{\Omega_m}) = \frac{1}{N_{\Omega_m}} \sum_{\mathbf{x}_i \in S_{\Omega_m}} \left[\nabla^2 \psi_\theta^{(m)}(\mathbf{x}_i)\right]^2, \tag{9}$$

$$L_{\Omega_w}(\boldsymbol{\theta}_w; S_{\Omega_w}) = \frac{1}{N_{\Omega_w}} \sum_{\mathbf{x}_i \in S_{\Omega_w}} \left[\nabla^2 \phi_\theta^{(w)}(\mathbf{x}_i) - \kappa^2 \phi_\theta^{(w)}(\mathbf{x}_i)\right]^2, \tag{10}$$

$$L_{\partial\Omega}(\boldsymbol{\theta}_w; S_{\partial\Omega}) = \frac{1}{N_{\partial\Omega}} \sum_{\mathbf{x}_i \in S_{\partial\Omega}} \left[\phi_\theta^{(w)}(\mathbf{x}_i) - \frac{1}{4\pi\varepsilon_w} \sum_k \frac{q_k e^{-\kappa|\mathbf{x}_i - \mathbf{x}_k|}}{|\mathbf{x}_i - \mathbf{x}_k|}\right]^2, \tag{11}$$

$$L_{\Gamma_D}(\boldsymbol{\theta}_m, \boldsymbol{\theta}_w; S_\Gamma) = \frac{1}{N_\Gamma} \sum_{\mathbf{x}_i \in S_\Gamma} \left[\phi_\theta^{(w)}(\mathbf{x}_i) - \psi_\theta^{(m)}(\mathbf{x}_i) - g(\mathbf{x}_i)\right]^2, \tag{12}$$

$$L_{\Gamma_N}(\boldsymbol{\theta}_m, \boldsymbol{\theta}_w; S_\Gamma) = \frac{1}{N_\Gamma} \sum_{\mathbf{x}_i \in S_\Gamma} \left[\varepsilon_w \frac{\partial}{\partial n}\left(\phi_\theta^{(w)}(\mathbf{x}_i)\right) - \varepsilon_m \frac{\partial}{\partial n}\left(\psi_\theta^{(m)}(\mathbf{x}_i) + g(\mathbf{x}_i)\right)\right]^2 \tag{13}$$

Each term in the loss function corresponds to a specific region of the domain: Ω_m (solute domain), Ω_w (solvent domain), $\partial\Omega$ (boundary of the solvent domain), and Γ (interface between the domains). Each loss term is weighted by a factor w_j (where the index j varies over $\{\Omega_m, \Omega_w, \partial\Omega, \Gamma_D, \Gamma_N\}$) to balance its contribution, ensuring that all components influence the optimization of the parameter set $\boldsymbol{\theta}$. Notably, the term L_{Ω_m} depends only on the parameters of the branch associated with the solute domain, while L_{Ω_w} and $L_{\partial\Omega}$ depend on the solvent domain. The two interface terms L_{Γ_D} and L_{Γ_N}, which enforce continuity of the potential and the electric displacement respectively, incorporate contributions from both neural networks.

In addition to the primary loss terms, additional terms can be introduced to incorporate known data, experimental results, or other relevant information. For example, a loss term based on known data can be defined as:

$$L_{data}(\boldsymbol{\theta}; S_{data}) = \frac{1}{N_{data}} \sum_{\mathbf{x}_i \in S_{data}} \left[\phi_\theta(\mathbf{x}_i) - \phi^\dagger(\mathbf{x}_i)\right]^2 \tag{14}$$

This term compares the predicted potential $\phi_\theta(\mathbf{x}_i)$ with known values $\phi^\dagger(\mathbf{x}_i)$ at the collocation points S_{data}.

3. Construction of collocation points

The interface Γ, modeled by a solvent-excluded surface (SES), often has complex geometry and hence it is not straightforward to construct the set of collocation points $S = S_{\Omega_m} \cup S_{\Omega_w} \cup S_\Gamma \cup S_{\partial\Omega}$. To accomplish this we first create a surface triangular mesh of the SES, and then create a volumetric tetrahedral mesh of the domain Ω which respects the SES (that is, the only intersection between a tetrahedron and the SES is a triangular face of the tetrahedron). Then we consider four sub-meshes: two tetrahedral sub-meshes corresponding to Ω_m and Ω_w, and two triangular sub-meshes corresponding to Γ and $\partial\Omega$. Then we select elements from each sub-mesh, and sample a point per each selected element randomly to generate $S_{\Omega_m}, S_{\Omega_w}, S_\Gamma$ and $S_{\partial\Omega}$.

This approach allows us to construct random samples without parametrizing the domains, while only requiring the discretization of the molecular surface—a well-established process. The distribution of collocation points throughout the domains depends on the distribution of the mesh elements, giving us full control over point density in different regions. This allows us to concentrate more points in areas where residuals are typically higher, such as at the interface. Additionally, the process can be repeated for selected elements to increase the density of collocation points in specific regions. We tried to follow the guidelines of[52] which states that the number of collocation points in each subdomain (solute, solvent, molecular surface and border condition) must increase with their size.

This approach is consistent with current research on collocation points in PINNs and can be adapted to include importance sampling techniques[75] and residual-based adaptive sampling methods[76] by adjusting the probability distribution for sampling within each mesh element.

D. An improved PINN architecture

We discuss improvements to the PINN architecture in this section. These enhancements lead to significant increases in the accuracy of the computed solution as we demonstrate in Section III.

1. Loss balancing algorithm

To ensure that each term of the loss function contributes effectively to the modification of the trainable parameters $\boldsymbol{\theta}$, we implemented a weight-adapting algorithm[77]. Consider the loss function as a linear combination of several loss terms:

$$L(\boldsymbol{\theta};S) = \sum_j w_j L_j(\boldsymbol{\theta};S_j) \tag{15}$$

Note that in our case the function $L(\boldsymbol{\theta};S)$ is given in (8) and corresponds to the form given above in (15). The objective is to determine the weights w_j for each loss term L_j such that the gradient $w_j \left\|\nabla_\theta L_j\right\|$ remains constant across all terms j:

$$C = w_j \left\|\nabla_\theta L_j\right\| \quad \forall j \tag{16}$$

To achieve this, we compute an estimator $\hat{w}_j$ for each term:

$$\hat{w}_j = \frac{\sum_i \|\nabla_\theta L_i\|}{\left\|\nabla_\theta L_j\right\|} \tag{17}$$

Next, we update the old weight using a soft adjustment, where α is a hyperparameter controlling the update rate:

$$w_{j,\text{new}} = \alpha w_{j,\text{old}} + (1-\alpha)\hat{w}_j \tag{18}$$

In this work, we set $\alpha = 1000$ to balance the influence of old and new weights.

2. Scaling layers

Two scaling layers are employed to normalize the inputs and outputs of the neural network, improving its convergence during training. This ensures that the values entering and leaving the network are between -1 and 1.

The input scaling is performed before the hidden layers. Given an input $\mathbf{x}$, the input scaling is,

$$\mathbf{h} = 2\frac{(\mathbf{x}-\mathbf{x}_{min})}{(\mathbf{x}_{max}-\mathbf{x}_{min})} - 1, \tag{19}$$

where the values $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ correspond to hyperparameters for each network and depend exclusively on the problem domain.

The output scaling is performed after the hidden layers as follows,

$$\mathbf{y} = \frac{\mathbf{h}+1}{2}(\mathbf{y}_{max}-\mathbf{y}_{min}) + \mathbf{y}_{min}, \tag{20}$$

where $\mathbf{h}$ indicates the output of the final hidden layer, and the values $\mathbf{y}_{min}$ and $\mathbf{y}_{max}$ correspond to hyperparameters for each network and must be estimated based on approximations of the real solution in each domain.

Estimating $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ is easy because they depend on the geometry, but estimating $\mathbf{y}_{min}$ and $\mathbf{y}_{max}$ is not straightforward since the solution to the problem is not known a priori. To estimate the values of $\mathbf{y}_{min}$ and $\mathbf{y}_{max}$, an approximation of the potential is constructed by superimposing the known solution of the Born ion, assuming that each charge in the macromolecule is an independent Born ion. More precisely, let the BORN function be defined as:

$$\text{BORN}(q_i,R_i) = \frac{q_i}{4\pi}\left(\frac{1}{\varepsilon_2(1+\kappa R_i)R_i} - \frac{1}{\varepsilon_1 R_i}\right) \tag{21}$$

The minimum and maximum reaction potentials (which are then used to obtain $\mathbf{y}_{min}$ and $\mathbf{y}_{max}$) are estimated with the following equations, where $R'_{ji} = \left\|\mathbf{x}_i - \mathbf{x}_j\right\|$ with $\mathbf{x}_j$ being the position of the point charge q_j.

$$\begin{aligned}\psi_{max} &= \max_i \left(\text{BORN}(q_i,R_i)\,,\, \text{BORN}(q_i,R_i) + \sum_{j\neq i}\text{BORN}(q_j,R'_{ji})\right)\\ \psi_{min} &= \min_i \left(\text{BORN}(q_i,R_i)\,,\, \text{BORN}(q_i,R_i) + \sum_{j\neq i}\text{BORN}(q_j,R'_{ji})\right)\end{aligned} \tag{22}$$

This does not guarantee that the output scaling layer will scale the solution strictly between -1 and 1, but in practice the values are close enough.

3. Random Fourier features layer

MLPs often suffer from a phenomenon called spectral bias, which biases the solution towards low-frequency functions, preventing the network from learning higher-order functions necessary to target the desired solution. This phenomenon can be mitigated using a Random Fourier Feature layer[78], which maps the input signals to a high-frequency space before passing them through the neural network. The layer is defined as follows:

$$\mathbf{h} = \begin{bmatrix} \cos(\mathbf{Bx}) \\ \sin(\mathbf{Bx}) \end{bmatrix} \tag{23}$$

where $\mathbf{B}$ is a matrix of shape $m \times d$, with m being the number of Fourier features and d the input dimension. The matrix $\mathbf{B}$ is generated from a normal distribution $\mathbf{B} \sim \text{Normal}(0, \sigma^2)$ and is non-trainable. This simple layer improves the PINNs method's ability to learn sharp gradients and complex solutions[77]. In this work, we used 128 Fourier features and a standard deviation of $\sigma = 1$.

4. *Trainable activation function*

Motivated by the work in[79], we implemented trainable activation functions. In this approach, the activation functions in each perceptron of the neural network are associated with a trainable parameter. Specifically, in this work, we use the hyperbolic tangent function, where the trainable parameter modifies the activation function as follows:

$$\sigma(x) = \tanh(ax) \tag{24}$$

Here, a is the trainable parameter (initialized to 1), which will be incorporated into the overall set of trainable parameters $\boldsymbol{\theta}$. The full set of parameters is defined as:

$$\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{a}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(L)}, \mathbf{a}^{(L)}\} \tag{25}$$

I will add an image of our architecture structure, denoting the 2 NN, scaling, fourier layers, etc.

5. *Complete Algorithm*

The complete algorithm used for the PINN solution of the PBE is presented in algorithm 1.

Algorithm 1 Algorithm for solving the PBE using PINNs

```
Input: Molecular information (.pdb file) and input variables (.yaml file)
  Generate mesh from the molecular information                          ▷ §II C 3
  Create the neural network $\mathcal{N} = \mathcal{N}_m \cup \mathcal{N}_w$ using the defined hyperparameters?
  for $k = 1$ to $k = n$ do                                             ▷ Training loop
      Obtain a random sample of collocation points $S = \bigcup_j S_j$ by sampling a set $S_j$ of collocation points from each subdomain
      Compute $\phi_\theta$ from $\mathcal{N}(S; \boldsymbol{\theta}_k)$                        ▷ Eq. (7)
      Compute the loss function $L(\boldsymbol{\theta}_k; S)$                     ▷ Eqs. (8) and (15)
      Update the trainable parameters $\boldsymbol{\theta}_{k+1}$ using an optimization method
      if mod($k$,$r$)==0 then This r needs to be mentioned earlier
          Update the weights $w_j$ of each loss term $L_j$                    ▷ §II D 1
      end if
  end for
Output: Optimized parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_m \cup \boldsymbol{\theta}_w$ for postprocessing.
```

6. *Software description*

We implemented algorithm 1 in a software package and named it XPPBE. The XPPBE software is straightforward to use. Below is an example of its basic usage:

Código .1: Basic example of using the XPPBE library

```python
from xppbe import Simulation
simulation = Simulation(yaml_path,molecule_dir)
simulation.create_simulation()
simulation.adapt_model()
simulation.solve_model()
simulation.postprocessing(run_all=True)
```

Change spanish to english in above code block

Note that the solver requires two inputs: a `yaml_path` for the `.yaml` file, which contains all the solver parameters (such as architectures, mesh parameters, and optimization methods, among others), and the `molecule_dir`, where the molecular information in a `.pdb` or `.pqr` file is stored. A more detailed explanation of these files can be found in the tutorials available in the repository[80].

III. RESULTS AND DISCUSSION

The following results are intended to validate the algorithm design outlined earlier. They are organized to demonstrate the impact of each algorithm feature in the solution, therefore providing evidence of the importance of including all of them. The analysis starts with the simple Born ion, and scales up to more complicated structures, where the influence of each component is clearer. The physical parameters were set to a solvent permittivity of $\varepsilon_w = 80$, an inverse of the Debye length of κ=0.125 Å^{-1}, and a solute dielectric constant of ε_m=2, except the spherical cases, where ε_m=1.

We explore a variety of PINN architectures to gauge the effect of different features. The base case, termed *Basic*, considers a fully connected multi-layer perceptron (MLP) architecture with 4 layers and 200 neurons per layer, with a trainable hyperbolic tangent activation function. The enhancements we consider are adding the weight adjusting algorithm (WA) of section II D 1, layers including Fourier features (FF) of section II D 3, and scaling of the input (SI) and output (SO) of the neural network of section II D 2. A combination of these features is indicated by the '+' sign e.g., $WA + SO$ indicates the usage of weight adjusting algorithm and output scaling. All trainable parameters were initialized using the Glorot normal distribution. We employed the Adam optimization algorithm with an exponentially decaying learning rate starting from 0.001.

The collocation points are created from surface and volume meshes. As the molecular surface definition, we used the solvent-excluded surface (SES)[?] , which we meshed with `msms`[?] or `Nanoshaper`[81]. From those surface definitions, we generated finite-element meshes with `lee2020open`[82]. We may cite Tetgen, because (as far as I know, lee2020open is an interface for tetgen in python)

All runs were performed on a workstation with two Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 12 cores each and 96 GB RAM memory, and a CUDA-enabled NVIDIA K40 GPU.

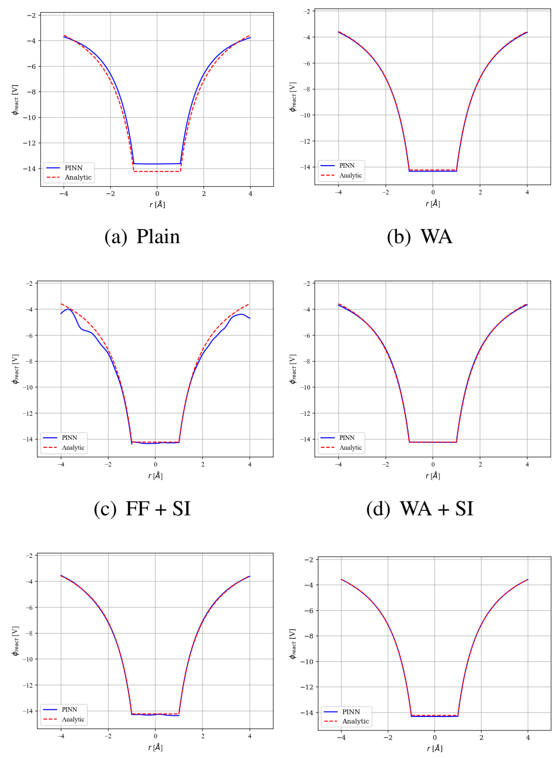
A. Born Ion

We started by modeling the electrostatic potential of a 1 Å radius Born ion with a centered $1e^-$ charge, using 6 different PINN architectures (See Figure 1). Starting from the base setup (*Basic*), we add systematically add other features to evaluate their effect on accuracy. In this test, the number of collocation nodes was N1 in the outer boundary, N2 in the SES, N3 in the solvent domain, and N4 in the molecular region, and the training was carried out for 20,000 iterations. The numbers N1 etc. were never used again. The training/-validation strategy needs to be explained.

Figure 1 shows the reaction potential (ψ) along the x axis using PINN (blue solid line) and an analytical expression[83] (red segmented line). We can see that…

Results fro Figure 1 are further supported by Table I, which shows ΔG_{solv}, errors in ΔG_{solv} and ψ what is meant by error in ψ, and the training and validation losses. The difference between them is the location where they are evaluated: the training loss is on the collocation points, whereas the validation loss is not. These results indicate that...

Figure 2 shows the evolution of the solvation energy and different losses throughout iterations. This figure shows that...

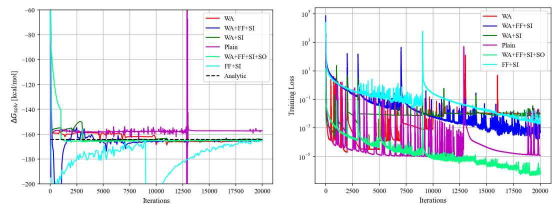


(a) Plain (b) WA

(c) FF + SI (d) WA + SI

(e) WA + FF + SI (f) WA + FF + SI + SO

FIG. 1. Reaction potential for the Born ion



(a) Solvation Energy (b) Training Loss

FIG. 2. Solvation energy and Training Loss for the six test cases

TABLE I. Born ion with six alternative architectures. Analytical solvation energy was XXX kcal/mol.

Case	Error $\mathscr{E}_{G_{solv}}$	Error $\mathscr{E}_{\psi}$	Training loss	Validation loss
Basic	4.38E-02	9.43E-01	1.165E-05	1.01E-05
WA	6.50E-03	6.62E-03	1.00E-02	9.85E-03
FF+SI	5.65E-03	5.24E-03	2.19E-03	1.02E-02
WA+SI	4.23E-05	1.89E-03	4.90E-03	4.72E-03
WA+FF+SI	3.24E-03	3.92E-03	4.70E-04	6.22E-04
WA+FF+SI+SO	5.28E-03	3.95E-03	6.81E-07	2.23E-06

B. Spherical molecule with off-centered charge

As a more challenging test, we moved on to model the electrostatic potential of a 1.2 Å sphere with an off-centered $+1e^{-}$ charge placed 0.45 Å away from the center. These simulations used the same number of collocation points and iterations as the Born ion. In this case, we only considered 4 alternative architectures: (i) considering weight and input scaling (WA + SI), (ii) case (i) plus a Fourier features layer (WA + FF + SI); (iii) case (i) adding an output scaling (WA + SI + SO), and (iv) turning all features on (WA + FF + SI + SO).

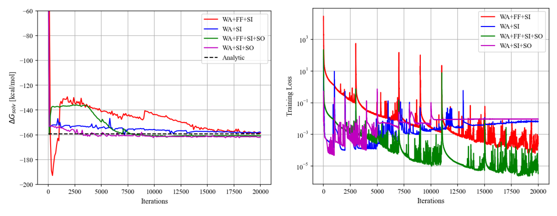
Figure 3 sketches ψ along the x axis. In this case, the importance of Fourier features and the output scaling on their own is more evident than for the Born ion, as Figure 4(a) is not capable of reproducing the curvature of the solution in the molecular region, whereas the other three alternatives can. Table II gives more quantitative evidence that including all features (WA + FF + SI + SO) yields the most accurate solution, as both errors and losses are the lowest. Also, Figure 3(b) shows that the losses for that case are not only the lowest, but are also decreasing at a higher rate that the other alternatives.

C. Calculations on arginine

Results with a spherical molecule in Tables I and II suggest that enabling all features of the algorithm (case WA+FF+SI+SO) yields the best results. Moving on to more realistic settings, we used this algorithm for a single arginine (XX atoms) to study the impact of (a) including a (approximated) known solution, and (b) the density of collocation nodes.

TABLE II. Results for the sphere with an off-centered charge. The analytical solution was XXX kcal/mol.

Case	Error $\mathscr{E}_{G_{solv}}$	Error $\mathscr{E}_{\psi}$	Training loss	Validation loss
WA+SI	6.82E-03	3.35E-02	6.55E-03	6.37E-03
WA+FF+SI	2.61E-03	3.37E-03	2.74E-04	7.26E-04
WA+SI+SO	1.69E-02	1.33E-02	9.36E-03	9.27E-03
WA+FF+SI+SO	9.98E-03	8.40E-03	3.30E-06	1.18E-05



(a) Solvation Energy (b) Training Loss

FIG. 3. Evolution of solvation energy and training loss with iterations for the sphere with an off-centered charge.

1. *Including a known solution*

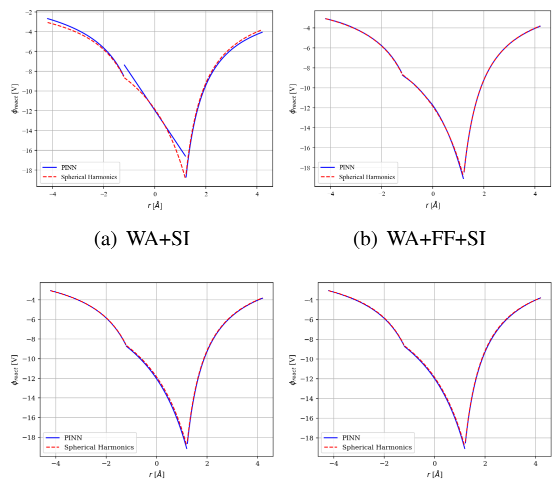
In many cases good approximations of the numerical solution are available, and could be used to help PINN to converge quicker. In molecular electrostatics, the potential away from the solute is very well modeled by the Yukawa potential:

$$g_Y(\mathbf{x}) = \sum_i^{n_c} q_i \frac{e^{-\kappa|\mathbf{x}-\mathbf{x}_i|}}{4\pi\varepsilon_w|\mathbf{x}-\mathbf{x}_i|} \tag{26}$$

which is equivalent to ignoring the low-dielectric region Ω_m, making it easy to compute. We then place N_Y nodes far from the solute, in Ω_w, where we enforce the following loss function (CHECK):

$$L_Y(\theta_w; S_{\Omega_w}) = \frac{1}{N_Y} \sum_{\mathbf{x}_i \in S_{\Omega_w}} \left[\phi_\theta^{(w)}(\mathbf{x}_i) - \frac{1}{4\pi\varepsilon_w} \sum_k \frac{q_k e^{-\kappa|\mathbf{x}_i-\mathbf{x}_k|}}{|\mathbf{x}_i-\mathbf{x}_k|} \right]^2 \tag{27}$$

The number of collocation points used in this study are detailed by Table **??**. Table IV shows the results after XXX iterations, which are further explored by the electrostatic potential distribution in Figure 5, along the the r axis for $\theta = 0, \phi =$



(a) WA+SI (b) WA+FF+SI

(c) WA+SI+SO (d) WA+FF+SI+SO

FIG. 4. Reaction potential (ψ) on the x axis for a sphere with an off-centered charge. PINN solution in blue, analytical solution in red.

$pi/2$. We are also including the history of ΔG_{solv} and losses in Figure 6. Although these results show a slight improvement of error metrics when considering known data (L_Y) (second and third column in Table III), both training and validation losses are worse by a small amount. Visual differences in Figures 5 and 6 are also minor. These results then indicate that including known data in computing the solution does not make a significant difference, and hence, we discarded the use of L_Y for the rest of our results.

TABLE III. Collocation nodes for arginine

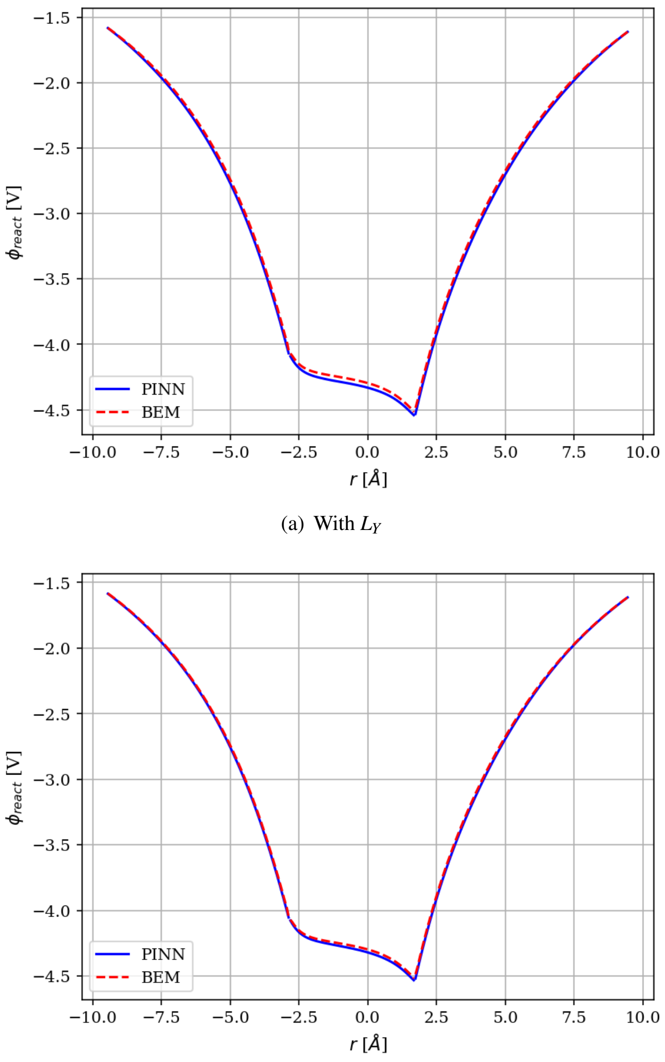
Ω_m	Ω_w	SES	$\partial\Omega$	N_Y
19582	59860	4536	8800	300

TABLE IV. Results for the argininee with known solution. Reference solution computed with BEM is XXX kcal/mol.

Case	Error $\mathscr{E}_{G_{solv}}$	Error $\mathscr{E}_{\psi}$	Training loss	Validation loss
With L_Y	6.92E-03	3.94E-03	1.99E-06	1.19E-06
Without L_Y	2.87E-03	4.85E-03	2.78E-06	1.81E-06

2. *Density of collocation nodes*

Similar to a mesh refinement study, this section intends to understand how the density of collocation nodes affects the quality of the solution. Table V describes the number of collocation nodes in 4 cases (*Coarse*, *Medium*, *Fine*, and *Finest*).



(a) With L_Y

(b) Without L_Y

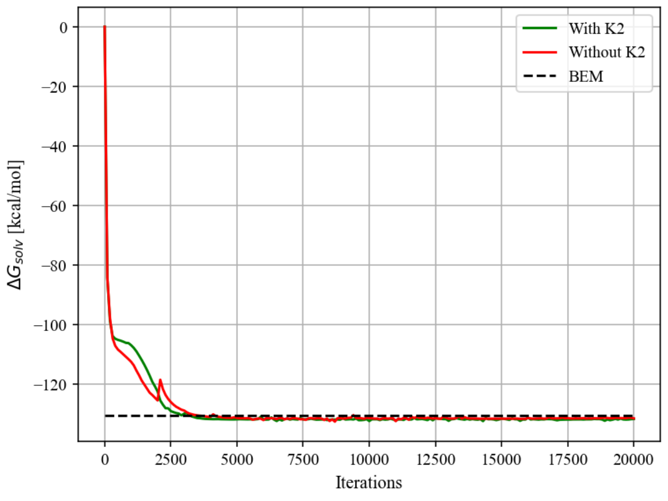
FIG. 5. Reaction potential for arginine along r for $\theta = 0$ $\phi = \pi/2$

The collocation node distributions were obtained from a surface mesh with W, X, Y, and Z vertices per Å^2 on the SES, respectively, and a volume mesh that started from the SES with a growth factor of XX. In all cases, the surface mesh of the surrounding sphere was left at YY vertices per Å^2.

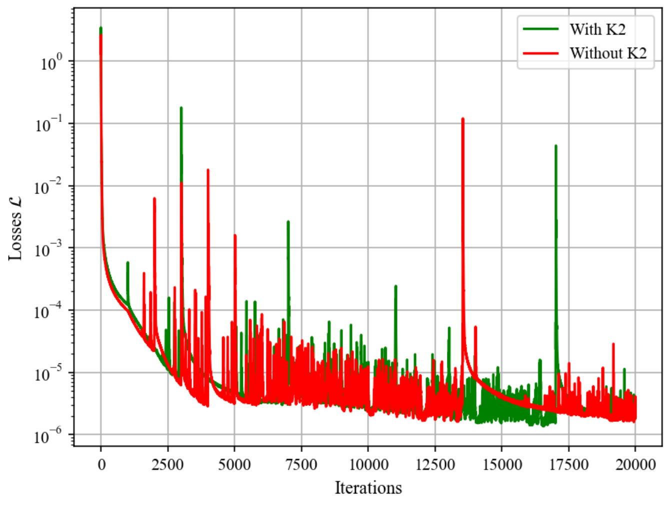
TABLE V. Number of collocation nodes in density study

Density	Ω_m	Ω_w	SES	$\partial\Omega$
Coarse	3383	10413	282	1238
Medium	4418	10440	372	1238
Fine	6017	11596	624	1238
Finest	7568	15089	1318	1238

Table VI shows the results for the collocation point density from Table V. As could be inferred from experience with nu-



(a) Solvation energy



(b) Training loss

FIG. 6. Solvation energy and training loss for L_Y study.

merical methods, all indicators drop as the number of collocation nodes increases. Note that the error in the second column is computed against a BEM solution that uses the equivalent mesh for each case (ie. the reference solution is different in each case). Regardless, the PINN error decreases with the mesh size, indicating that it is converging to the numerical solution computed with BEM. The latter statement is more evident in Figure 7, where the black line converges to the red line, which corresponds to the BEM solution computed on the mesh that was used to generate the collocation points. As a reference, the blue line in Figure **??** is a BEM solution with a surface mesh that is XX times finer than the *Finest* case.

Figure 8 shows the reaction potential in the direction $\theta = \pi/2$, $\phi = pi/2$. From these results, it is evident that the coarsest PINN calculation struggles to correspond to the numerical solution, however, this improves substantially for the medium density.

Comment Figure 9.

TABLE VI. Results for collocation node density study of arginine.

Case	ΔG^{θ}_{solv} [kcal/mol]	Error $\mathscr{E}_{G_{solv}}$	Error $\mathscr{E}_{\psi}$	Training loss	Validation loss
Coarse	-174.90	1.66E-01	8.91E-02	8.15E-04	3.36E-04
Medium	-145.99	3.14E-02	5.47E-02	3.07E-03	1.27E-04
Fine	-140.74	1.68E-02	3.20E-02	1.91E-04	2.81E-04
Finest	-134.72	7.23E-03	2.59E-02	8.63E-05	2.91E-05

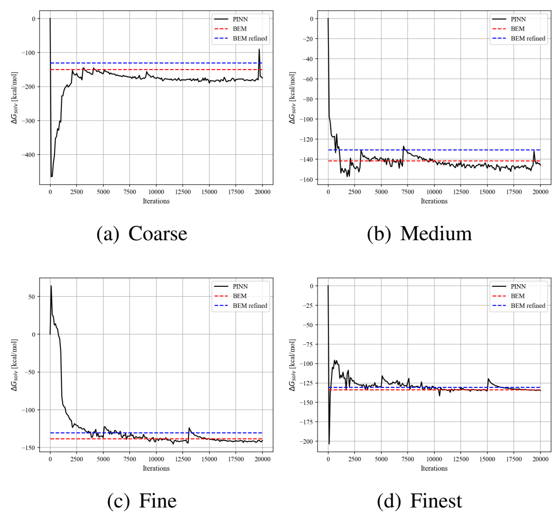


FIG. 7. Solvation energy history with iterations (black line). The red line is a BEM solution computed on the surface mesh that was used to generate the collocation nodes. The blue line corresponds to a fine-mesh BEM solution.

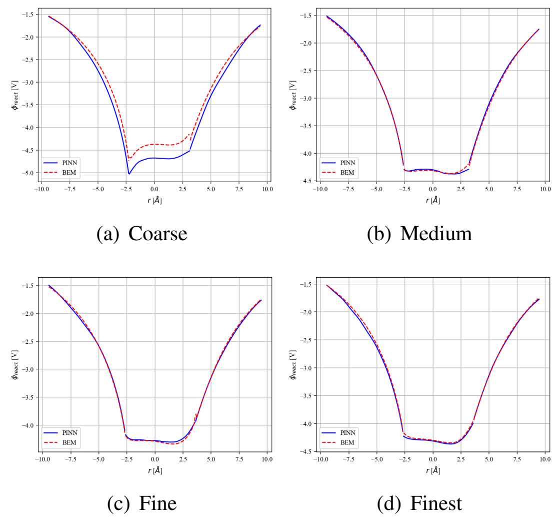


FIG. 8. Reaction potential for arginine with different node refinements, along the r axis for $\theta = \pi/2$, $\phi = \pi/2$. Red line: BEM solution. Blue line: PINN solution.

The volume and surface meshes used to generate the collocation points are related, as the volume mesh is generated starting from the surface mesh. However, there is no evident reason why the surface and volume collocation points should be coupled. To decouple them, we performed two extra calculations, where we used the collocation points from the *Finest* calculation, but sampled only a subset of the volume collocation points, while using all surface nodes. This is detailed in Table VII, where we samplex XX% and YY% of the volume

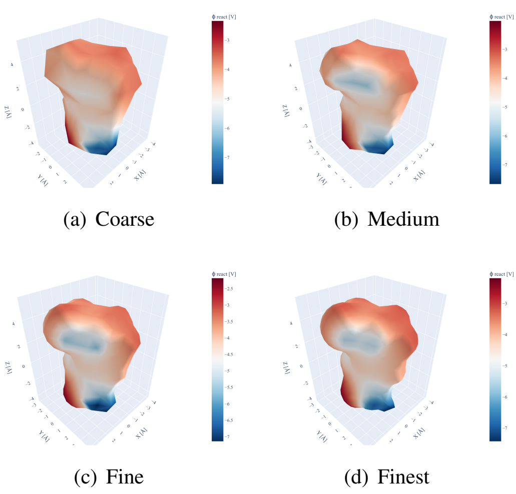


FIG. 9. Potencial de reacción en la interfaz de la arginina para las distintas mallas

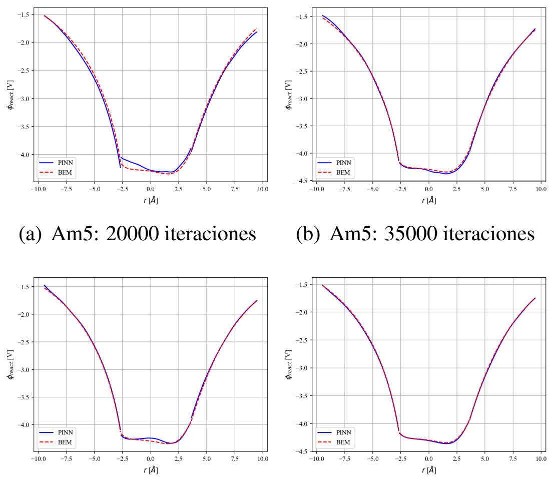
nodes. Results are presented in Table **??**

TABLE VII. Puntos de colocación utilizados para las nuevas mallas de la arginina

Simulación	Nodos R1	Nodos R2	Nodos I	Nodos D2
Finest XX	2282	4589	1318	1238
Finest YY	4494	9090	1318	1238

TABLE VIII. Resultados para estudio de malla de la arginina Solvation energy error must be changed with respect to the BEM mesh

Simulación	Iteraciones	ΔG^{θ}_{solv} [kcal/mol]	Error rel. $\mathscr{E}_{G_{solv}}$	Error L_2 $\mathscr{E}_{\psi}$	Loss $\mathscr{L}$ entrenamiento	I v
Am5	20000	-131.952	8.96E-03	2.76E-02	2.82E-04	2
Am5	35000	-133.641	8.37E-04	2.48E-02	1.68E-04	1
Am6	20000	-134.575	6.15E-03	2.74E-02	1.38E-04	5
Am6	35000	-135.220	1.10E-02	2.25E-02	2.39E-05	1


(a) Am5: 20000 iteraciones (b) Am5: 35000 iteraciones

(c) Am6: 20000 iteraciones (d) Am6: 35000 iteraciones

FIG. 10. Potencial de reacción para la arginina, dirección $\theta = \pi/2$, $\phi = \pi/2$

D. Bigger Molecules

Results for 1pgb[84] and 1ubq[85], protein with around 1000 point charges and complex interface geometry.

All results with WA+FF+SI+SO without K2.

```
mesh_properties:
    vol_max_interior: 0.6
    vol_max_exterior: 2.0
    percentage_vol_mesh: 0.7
    density_mol: 1.8
    density_border: 1.6
    mesh_generator: nanoshaper
    dR_exterior: 4.0
```

```
Collocation points for 1pgb:
    I_sample: 10092
    D2_sample: 23976
    R1_sample: 33252
    R2_sample: 110941
```

TABLE IX. Results for bigger molecules

Molecule	ΔG^{θ}_{solv} [kcal/mol]	Error $\mathscr{E}_{G_{solv}}$	Error $\mathscr{E}_{\psi}$	Training loss	Validation loss
1pgb 1ubq	-520.14	1.99E-02	3.41E-02	1.06E-06	2.50E-06

Non linear DNA

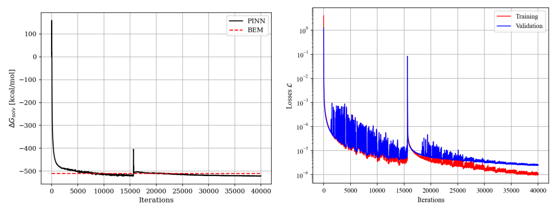

(a) G solv (b) Losses

FIG. 11. Solvation energy and losses evolution for 1pgb

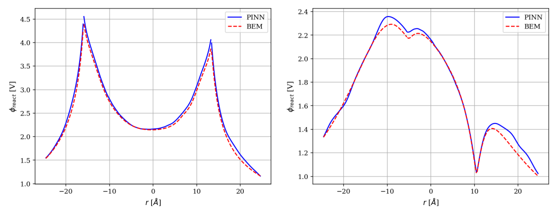

(a) x axis (b) y axis

FIG. 12. Reaction potential for the 1pgb in 2 directions for 1pgb

IV. CHALLENGES

- Experimental results
- Adam to BFGS
- nonlinear without approximation
- theoretical studies

Experimental results Testing repo

V. CONCLUSIONS

[1]B. Roux and T. Simonson, "Implicit solvent models," Biophysical chemistry **78**, 1–20 (1999).
[2]S. Decherchi, M. Masetti, I. Vyalov, and W. Rocchia, "Implicit solvent methods for free energy estimation," European Journal of Medicinal Chemistry **91**, 27–42 (2015).
[3]M. K. Gilson, A. Rashin, R. Fine, and B. Honig, "On the calculation of electrostatic interactions in proteins," Journal of Molecular Biology **184**, 503–516 (1985).
[4]N. A. Baker, D. Sept, M. J. Holst, and J. A. McCammon, "Electrostatics of nanoysystems: Application to microtubules and the ribosome," Proceedings of the National Academy of Sciences of the USA **98**, 10037–10041 (2001).
[5]W. Rocchia, E. Alexov, and B. Honig, "Extending the applicability of the nonlinear poisson- boltzmann equation: multiple dielectric constants and multivalent ions," The Journal of Physical Chemistry B **105**, 6507–6514 (2001).

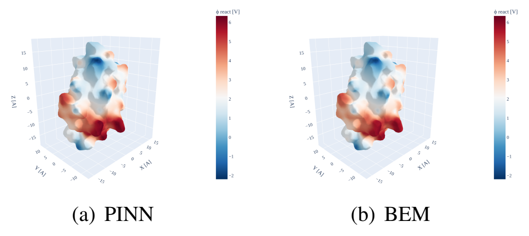

(a) PINN (b) BEM

FIG. 13. Reaction potential at interface for 1pgb

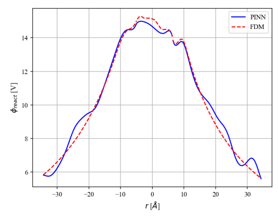


FIG. 14. Reaction potential DNA non linear

[6]A. H. Boschitsch and M. O. Fenley, "A fast and robust poisson–boltzmann solver based on adaptive cartesian grids," Journal of Chemical Theory and Computation **7**, 1524–1540 (2011).

[7]E. Jurrus, D. Engel, K. Star, K. Monson, J. Brandi, L. E. Felberg, D. H. Brookes, L. Wilson, J. Chen, K. Liles, *et al.*, "Improvements to the apbs biomolecular solvation software suite," Protein Science **27**, 112–128 (2018).

[8]C. M. Cortis and R. A. Friesner, "Numerical solution of the poisson–boltzmann equation using tetrahedral finite-element meshes," Journal of Computational Chemistry **18**, 1591–1608 (1997).

[9]L. Chen, M. J. Holst, and J. Xu, "The finite element approximation of the nonlinear poisson–boltzmann equation," SIAM journal on numerical analysis **45**, 2298–2320 (2007).

[10]D. Xie and S. Zhou, "A new minimization protocol for solving nonlinear poisson–boltzmann mortar finite element equation," BIT Numerical Mathematics **47**, 853–871 (2007).

[11]S. D. Bond, J. H. Chaudhry, E. C. Cyr, and L. N. Olson, "A first-order system least-squares finite element method for the poisson-boltzmann equation," Journal of Computational Chemistry **31**, 1625–1635 (2010).

[12]P. B. Shaw, "Theory of the Poisson Green's-function for discontinuous dielectric media with an application to protein biophysics," Physical Review A **32**, 2476–2487 (1985).

[13]B. J. Yoon and A. M. Lenhoff, "A boundary element method for molecular electrostatics with electrolyte effects," Journal of Computational Chemistry **11**, 1080–1086 (1990).

[14]A. Juffer, E. F. Botta, B. A. van Keulen, A. van der Ploeg, and H. J. Berendsen, "The electric potential of a macromolecule in a solvent: A fundamental approach," Journal of Computational Physics **97**, 144–171 (1991).

[15]A. H. Boschitsch, M. O. Fenley, and H.-X. Zhou, "Fast boundary element method for the linear poisson- boltzmann equation," The Journal of Physical Chemistry B **106**, 2741–2754 (2002).

[16]B. Lu, X. Cheng, J. Huang, and J. A. McCammon, "Order n algorithm for computation of electrostatic interactions in biomolecular systems," Proceedings of the National Academy of Sciences **103**, 19314–19319 (2006).

[17]W. Geng and R. Krasny, "A treecode-accelerated boundary integral poisson–boltzmann solver for electrostatics of solvated biomolecules," Journal of Computational Physics **247**, 62–78 (2013).

[18]C. D. Cooper, J. P. Bardhan, and L. A. Barba, "A biomolecular electrostatics solver using Python, GPUs and boundary elements that can handle solvent-filled cavities and Stern layers." Computer Physics Communications **185**, 720–729 (2014).

[19]S. D. Search, C. D. Cooper, and E. Van't Wout, "Towards optimal boundary integral formulations of the poisson–boltzmann equation for molecular electrostatics," Journal of Computational Chemistry **43**, 674–691 (2022).

[20]L. E. Felberg, D. H. Brookes, E.-H. Yap, E. Jurrus, N. A. Baker, and T. Head-Gordon, "Pb-am: An open-source, fully analytical linear poisson-boltzmann solver," Journal of computational chemistry **38**, 1275–1282 (2017).

[21]S. V. Siryk and W. Rocchia, "Arbitrary-shape dielectric particles interacting in the linearized poisson–boltzmann framework: An analytical treatment," The Journal of Physical Chemistry B **126**, 10400–10426 (2022).

[22]A. Jha, M. Nottoli, A. Mikhalev, C. Quan, and B. Stamm, "Linear scaling computation of forces for the domain-decomposition linear poisson–boltzmann method," The Journal of Chemical Physics **158** (2023).

[23]A. Jha and B. Stamm, "Domain decomposition method for poisson–boltzmann equations based on solvent excluded surface," arXiv preprint arXiv:2309.06862 (2023).

[24]A. H. Boschitsch and M. O. Fenley, "Hybrid boundary element and finite difference method for solving the nonlinear poisson–boltzmann equation," Journal of Computational Chemistry **25**, 935–955 (2004).

[25]J. Ying and D. Xie, "A hybrid solver of size modified poisson–boltzmann equation by domain decomposition, finite element, and finite difference," Applied Mathematical Modelling **58**, 166–180 (2018).

[26]M. Bosy, M. W. Scroggs, T. Betcke, E. Burman, and C. D. Cooper, "Coupling finite and boundary element methods to solve the poisson-boltzmann equation for electrostatics in molecular solvation," Journal of Computational Chemistry **45**, 787–797 (2024).

[27]M. Kapteyn, D. Knezevic, D. Huynh, M. Tran, and K. Willcox, "Data-driven physics-based digital twins via a library of component-based reduced-order models," International Journal for Numerical Methods in Engineering **123**, 2986–3003 (2022).

[28]M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat, "Deep learning and process understanding for data-driven earth system science," Nature **566**, 195–204 (2019).

[29]D. Calvetti and E. Somersalo, *Introduction to Bayesian Scientific Computing* (Springer New York, 2007).

[30]P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," SIAM Review **57**, 483–531 (2015).

[31]G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," Nature Reviews Physics **3**, 422–440 (2021).

[32]M. G. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," communications in Numerical Methods in Engineering **10**, 195–201 (1994).

[33]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 Compu-

tational physics **378**, 686–707 (2019).

[34]S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics–informed neural networks: Where we are and what's next," Journal of Scientific Computing **92**, 88 (2022).

[35]S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (pinns) for fluid mechanics: a review," Acta Mechanica Sinica **37**, 1727–1738 (2021).

[36]S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-Informed Neural Networks for Heat Transfer Problems," Journal of Heat Transfer **143**, 060801 (2021).

[37]M. Baldan, P. Di Barba, and D. A. Lowther, "Physics-informed neural networks for inverse electromagnetic problems," IEEE Transactions on Magnetics **59**, 1–5 (2023).

[38]J. D. Schmid, P. Bauerschmidt, C. Gurbuz, M. Eser, and S. Marburg, "Physics-informed neural networks for acoustic boundary admittance estimation," Mechanical Systems and Signal Processing **215**, 111405 (2024).

[39]C. Eldred, F. Gay-Balmaz, S. Huraka, and V. Putkaradze, "Lie–poisson neural networks (lpnets): Data-based computing of hamiltonian systems with symmetries," Neural Networks **173**, 106162 (2024).

[40]B. Yu *et al.*, "The deep ritz method: a deep learning-based numerical algorithm for solving variational problems," Communications in Mathematics and Statistics **6**, 1–12 (2018).

[41]E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "Variational physics-informed neural networks for solving partial differential equations," arXiv preprint arXiv:1912.00873 (2019).

[42]G. Lin, P. Hu, F. Chen, X. Chen, J. Chen, J. Wang, and Z. Shi, "Binet: Learning to solve partial differential equations with boundary integral networks," arXiv preprint arXiv:2110.00352 (2021).

[43]J. Sun, Y. Liu, Y. Wang, Z. Yao, and X. Zheng, "Binn: A deep learning approach for computational mechanics problems based on boundary integral equations," Computer Methods in Applied Mechanics and Engineering **410**, 116012 (2023).

[44]A. D. Jagtap and G. E. Karniadakis, "Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," Communications in Computational Physics **28** (2020).

[45]A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," Computer Methods in Applied Mechanics and Engineering **365**, 113028 (2020).

[46]V. Dwivedi, N. Parashar, and B. Srinivasan, "Distributed physics informed neural network for data-efficient solution to partial differential equations," arXiv preprint arXiv:1907.08967 (2019).

[47]W. Li, X. Xiang, and Y. Xu, "Deep domain decomposition method: Elliptic problems," in *Mathematical and Scientific Machine Learning* (PMLR, 2020) pp. 269–286.

[48]C. He, X. Hu, and L. Mu, "A mesh-free method using piecewise deep neural network for elliptic interface problems," Journal of Computational and Applied Mathematics **412**, 114358 (2022).

[49]S. Wu and B. Lu, "INN: Interfaced neural networks as an accessible meshless approach for solving interface PDE problems," Journal of Computational Physics **470**, 111588 (2022).

[50]J. Ying, J. Liu, J. Chen, S. Cao, M. Hou, and Y. Chen, "Multi-scale fusion network: A new deep learning structure for elliptic interface problems," Applied Mathematical Modelling **114**, 252–269 (2023).

[51]Y.-H. Tseng, T.-S. Lin, W.-F. Hu, and M.-C. Lai, "A cusp-capturing PINN for elliptic interface problems," Journal of Computational Physics **491**, 112359 (2023).

[52]X. Jiang, Z. Wang, W. Bao, and Y. Xu, "Generalization of PINNs for elliptic interface problems," Applied Mathematics Letters , 109175 (2024).

[53]A. K. Sarma, S. Roy, C. Annavarapu, P. Roy, and S. Jagannathan, "Interface pinns (i-pinns): A physics-informed neural networks framework for interface problems," Computer Methods in Applied Mechanics and Engineering **429**, 117135 (2024).

[54]S. Wu, A. Zhu, Y. Tang, and B. Lu, "Convergence of physics-informed neural networks applied to linear second-order elliptic interface problems," Communications in Computational Physics **33**, 596–627 (2023).

[55]Z. Liu, W. Cai, and Z.-Q. John Xu, "Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains," Communications in Computational Physics **28**, 1970–2001 (2020).

[56]S. Wu, A. Zhu, Y. Tang, and B. Lu, "Solving parametric elliptic interface problems via interfaced operator network," Journal of Computational Physics **514**, 113217 (2024).

[57]J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multi-objective optimization," Comptes Rendus Mathematique **350**, 313–318 (2012).

[58]`https://www.tensorflow.org/`.

[59]`https://pytorch.org/`.

[60]D. P. Kingma, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980 (2014).

[61]B. Lee and F. Richards, Journal of Molecular Biology **55**, 379–IN4 (1971).

[62]J. Chen, Y. Xu, X. Yang, Z. Cang, W. Geng, and G.-W. Wei, "Poisson-boltzmann-based machine learning model for electrostatic analysis," Biophysical Journal (2024).

[63]P. B. Reis, D.-A. Clevert, and M. Machuqueiro, "Pypka server: online p k a predictions and biomolecular structure preparation with precomputed data from pdb and alphafold db," Nucleic Acids Research , gkae255 (2024).

[64]`https://github.com/MartinAchondo/XPPBE`.

[65]P.-Y. Chuang and L. A. Barba, "Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration," arXiv preprint arXiv:2205.14249 (2022).

[66]T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb, "Can physics-informed neural networks beat the finite element method?" IMA Journal of Applied Mathematics , hxae011 (2024).

[67]T. Kurth, S. Subramanian, P. Harrington, J. Pathak, M. Mardani, D. Hall, A. Miele, K. Kashinath, and A. Anandkumar, "Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '23 (Association for Computing Machinery, New York, NY, USA, 2023).

[68]K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian, "Accurate medium-range global weather forecasting with 3d neural networks," Nature **619**, 533–538 (2023).

[69]A. D. Jagtap, Z. Mao, N. Adams, and G. E. Karniadakis, "Physics-informed neural networks for inverse problems in supersonic flows," Journal of Computational Physics **466**, 111402 (2022).

[70]Z. Zhou, P. Payne, M. Vasquez, N. Kuhn, and M. Levitt, "Finite-difference

solution of the Poisson-Boltzmann equation: Complete elimination of self-energy," **17**, 1344–1351.

[71]M. Holst, J. A. Mccammon, Z. Yu, Y. Zhou, and Y. Zhu, "Adaptive finite element modeling techniques for the poisson-boltzmann equation," Communications in computational physics **11**, 179–214 (2012).

[72]I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (MIT press, 2016).

[73]Y. Shin, J. Darbon, and G. E. Karniadakis, "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes," arXiv preprint arXiv:2004.01806 (2020).

[74]Throughout this paper, the term branches will also be referred to as independent neural networks or simply networks.

[75]M. A. Nabian, R. J. Gladstone, and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," Computer-Aided Civil and Infrastructure Engineering **36**, 962–977 (2021).

[76]C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," Computer Methods in Applied Mechanics and Engineering **403**, 115671 (2023).

[77]S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, "An Expert's Guide to Training Physics-informed Neural Networks," arXiv preprint arXiv:2308.08468 (2023).

[78]M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," Advances in neural information processing systems **33**, 7537–7547 (2020).

[79]A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," Journal of Computational Physics **404**, 109136 (2020).

[80]M. Achondo, "XPPBE: PINNs for PBE," `https://github.com/MartinAchondo/XPPBE` (2023).

[81]S. Decherchi and W. Rocchia, "A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale," PLOS one **8**, e59744 (2013).

[82]C. T. Lee, J. G. Laughlin, J. B. Moody, R. E. Amaro, J. A. McCammon, M. Holst, and P. Rangamani, "An open-source mesh generation platform for biophysical modeling using realistic cellular geometries," Biophysical Journal **118**, 1003–1008 (2020).

[83]J. G. Kirkwood, "Theory of solutions of molecules containing widely separated charges with special application to zwitterions," The Journal of Chemical Physics **2**, 351–361 (1934).

[84]T. Gallagher, P. Alexander, P. Bryan, and G. L. Gilliland, "Two crystal structures of the b1 immunoglobulin-binding domain of streptococcal protein g and comparison with nmr," Biochemistry **33**, 4721–4729 (1994).

[85]S. Vijay-Kumar, C. E. Bugg, and W. J. Cook, "Structure of ubiquitin refined at 1.8 åresolution," Journal of molecular biology **194**, 531–544 (1987).