Discontinuous Extreme Learning Machine for Interface and Free Boundary Problems

Anci Lin[a], Zhiwen Zhang[b,*], Weidong Zhao[a], Wenju Zhao[a,**]

[a]*School of Mathematics, Shandong University, Jinan, Shandong 250100, China*
[b]*Department of Mathematics, The University of Hong Kong, Pokfulam Road, Hong Kong SAR, P.R. China. Materials Innovation Institute for Life Sciences and Energy (MILES), HKU-SIRI, Shenzhen, P.R. China.*

Abstract

Interface and free boundary problems are fundamental in science and engineering, with interface non-smoothness posing significant challenges. While machine learning-based methods have shown promise for interface problems with predefined boundaries, their application to free boundary problems is hindered by computational demands and design limitations. Consequently, high-accuracy machine learning approaches for free boundary problems remain scarce. This paper presents a novel mesh-free approach based on the locELM framework, termed discontinuous extreme learning machine (DELM), a variant of physics-informed neural networks, to address the problems above. By utilizing the signed distance level set function and introducing an artificial discontinuity mechanism, DELM effectively handles interface non-smoothness with high accuracy and computational efficiency, using fewer parameters than traditional methods. Moreover, its independence from the specific forms of predefined interfaces enables seamless integration with front-tracking techniques, making it well-suited for free boundary problems. The proposed method achieves excellent performance in various interface and Stefan problems, providing an efficient and scalable solution for complex interface dynamics.

Keywords: Physics-informed neural networks, Artificial discontinuity, Interface problems, Free boundary problems, Extreme learning machine.

1. Introduction

Interface problems [1] play a crucial role in numerous scientific and engineering fields, spanning multiphase flows in fluid mechanics, and phase transitions in materials science. Broadly, interface problems encompass scenarios in which the interface may be either known or unknown. When the interface is unknown, these are known as *free boundary problems* [2], where determining the boundary is integral to finding the solution. In such problems, interfaces typically partition the space into multiple subdomains, each with distinct physical

*Co-corresponding author: zhangzw@hku.hk
**Co-corresponding author: zhaowj@sdu.edu.cn

properties, and the interface geometry can often be quite intricate. Partial differential equations (PDEs) with interfaces or moving boundaries are frequently employed to model the physical processes within these subdomains. However, the solution near interfaces, which is often non-smooth, presents significant challenges for numerical methods. Current approaches for solving interface problems generally fall into two categories: *mesh-based methods* and *mesh-free methods*. Each category has distinct advantages, making them valuable in different application contexts.

Traditional mesh-based methods for interface problems generally represent the interface either explicitly or implicitly [3]. In cases where the interface is unknown, explicit methods stand out, such as the front-tracking method [4, 5]. By explicitly tracking the movement of the interface as it evolves, this approach offers the advantage of precise localization and a clear representation of interface dynamics. However, front-tracking methods encounter significant challenges when the interface undergoes complex topological transformations, such as splitting or merging. In contrast, implicit methods handle the interface indirectly by introducing level set functions [6, 7, 8] or phase field functions [8, 9, 10, 11] on a fixed mesh. The main advantage of these methods is their ability to seamlessly manage complex topological changes, without requiring explicit tracking. Nonetheless, the accuracy of interface localization in implicit methods is typically lower compared to explicit approaches, as the interface is represented indirectly.

When constructing spatial meshes for interface problems, two commonly used approaches are body-fitted meshes and unfitted meshes. Body-fitted meshes are designed to align precisely with the interface boundaries, allowing for an accurate depiction of physical phenomena near the interface. This alignment offers high accuracy but comes with a significant drawback: it requires frequent mesh adjustments, particularly when the interface evolves. These adjustments can be time-intensive and often rely on computationally costly a posteriori error estimates. A notable example of this approach is the adaptive mesh finite element method [12, 13, 14]. In contrast, unfitted meshes do not require exact alignment with the interface. Instead, the interface is allowed to intersect freely with the background mesh, simplifying the mesh generation process and offering greater flexibility, especially for complex geometries or dynamic interfaces. Since the mesh does not align with the interface, it often becomes necessary to employ additional numerical techniques to address the discontinuities and irregularities in the vicinity of the interface. Examples of such techniques include the immersed interface method [15, 16, 17, 18, 19, 20], the extended finite element method (XFEM) [21, 22, 23], cutFEM [24], and the ghost fluid method [25]. Each of these methods addresses interface discontinuities in distinct ways, contributing to improved stability and accuracy in numerical calculations.

Recently, mesh-free methods, particularly those leveraging neural networks [26, 27], have gained significant attention for addressing interface problems. These methods achieve discretization through sampling at discrete points, which in turn avoids the complexities associated with mesh generation. Among these, piecewise neural networks [28, 29, 30] manage discontinuities by deploying independent neural networks across different subdomains, making them well-suited for both known and unknown interface problems. However, employing

multiple neural networks results in increased computational demands. For free boundary problems, an additional neural network is often needed to track the evolving interface, further amplifying the computational burden. The discontinuity capturing shallow neural network (DCSNN) [31] and cusp-capturing PINN (C-C PINN) [32], which address some of these challenges by introducing augmented variables and employing the Levenberg-Marquardt (LM) method, significantly alleviate computational challenges in elliptic interface problems. Nonetheless, these approaches still require a small set of additional parameters for the augmented variables and rely on a known interface, limiting its application to free boundary problems. Additionally, methods based on the random feature method (RFM) [33, 34, 35, 36] and extreme learning machine (ELM) [37] have been proposed for interface problems with known interfaces. These approaches typically reformulate the problem into a least-squares framework, achieving spectral accuracy in some cases. However, their reliance on known interfaces limits their generalization to free boundary problems

This paper introduces a novel mesh-free framework, termed DELM, designed to provide a unified solution approach for interface problems with both known and unknown interfaces. DELM is based on the ELM methodology and its variant, locELM [38], which incorporates domain decomposition techniques. The locELM method enhances the original ELM by enforcing specific smoothness conditions, yielding improved results in relatively simple problem settings. Building on this foundation, we advance the work of Lai et al. [31, 32] by enhancing the use of augmented variables (see Section 3.2). This is achieved through the effective incorporation of signed-distance functions, which capitalize on their superior quality features. By refining this integration, our approach seeks to further improve the speed and robustness of the method. As a result, our approach achieves notable performance gains for problems with known interfaces (see Section 4.2) and demonstrates compatibility with traditional front-tracking methods, thereby enabling the application to free boundary problems (see Section 4.3). Furthermore, our method introduces augmented variables without requiring additional parameters in neural networks, thus enhancing computational efficiency. Our contributions are summarized as follows:

- We propose a novel method, which is based on locELM, and efficiently addresses both known and unknown interface problems while maintaining high accuracy and fast computational speed.
- We present an enhanced augmented variable technique tailored for general interface problems. Building on the work of Lai et al. [31, 32], our approach further reduces the number of parameters, improves computational efficiency, and extends its applicability to free boundary problems.
- We integrate traditional front-tracking methods into our framework to achieve accurate and robust interface modeling.
- We demonstrate the effectiveness of our method through comprehensive validation on benchmark interface and free boundary problems.

This paper is organized as follows. Section 2 briefly introduces the mathematical formulation of interface problems. Section 3 details the proposed method, including the generalization of locELM for interface problems, the introduction of artificial discontinuities, and the extension to free boundary problems. Section 4 presents numerical results for both interface and free boundary problems. Finally, Section 5 summarizes the findings and discusses potential future research directions.

2. Mathematical formulation of interface and free boundary problems

Let $\Omega \subset \mathbb{R}^d$ (where $d = 1, 2, 3$) be a bounded domain that is partitioned into two disjoint subdomains, Ω_t^+ and Ω_t^-, separated by an interface Γ_t, which is generally not known a priori. The subscript t signifies that this partition is time-dependent and may evolve. This decomposition satisfies $\Omega = \Omega_t^+ \cup \Gamma_t \cup \Omega_t^-$, as illustrated in Fig. 1. The objective is to

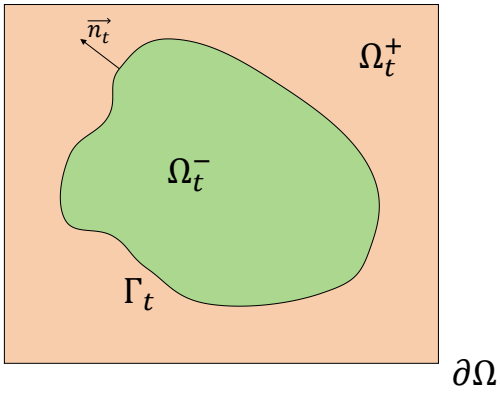


Figure 1: A sketch map for the domain and the interface.

determine a function $u(x, t)$ that fulfills distinct differential equations within each subdomain and meets specific conditions along the interface. Consider the abstract form of the interface or free boundary problem:

$$\begin{cases} (\partial_t + \mathcal{L}^{\pm})u(\mathbf{x}, t) = f^{\pm}(\mathbf{x}, t), & (\mathbf{x}, t) \in \Omega_t^{\pm} \times [0, T], \quad (2.1a) \\ [\![\mathcal{G}u(\mathbf{x}, t)]\!]_{\Gamma_t} = (\phi_1(\mathbf{x}, t), \phi_2(\mathbf{x}, t)), & (\mathbf{x}, t) \in \Gamma_t \times [0, T], \quad (2.1b) \\ [\![\mathcal{M}u(\mathbf{x}, t)]\!]_{\Gamma_t} = \langle \psi, \Gamma_t \rangle(\mathbf{x}, t), & (\mathbf{x}, t) \in \Gamma_t \times [0, T], \quad (2.1c) \\ \mathcal{B}u(\mathbf{x}, t) = g(\mathbf{x}, t), & (\mathbf{x}, t) \in \partial\Omega \times [0, T], \quad (2.1d) \\ \mathcal{I}u(\mathbf{x}, 0) = h(\mathbf{x}), & (\mathbf{x}, t) \in \Omega. \quad (2.1e) \end{cases}$$

In these equations, $\mathcal{L}^{\pm}$ represents spatial differential operators defined on subdomains $\Omega_t^{\pm}$, with possible variations in form or coefficients between the two regions. The notation ∂_t denotes the time derivative, reflecting that this problem can be dynamic or stationary. Boundary and initial conditions are specified through the operators $\mathcal{B}$ and $\mathcal{I}$, with $g(\mathbf{x}, t)$ and $h(\mathbf{x})$ as their respective values on $\partial\Omega$ and the initial time.

The jump condition across the interface Γ_t is defined as follows:

$$[\![u(\mathbf{x},t)]\!]_{\Gamma_t} = u^+(\mathbf{x},t) - u^-(\mathbf{x},t), \tag{2.2}$$

signifying the discontinuity across Γ_t, where $u^\pm$ denote the values of u approaching from $\Omega_t^\pm$. The operator $\mathcal{G}$ in Eq. (2.1b), mapping to a product space, enforces two interface conditions across Γ_t. For problems with dynamic interfaces, $\mathcal{G}$ takes the form:

$$\begin{aligned} \mathcal{G} &= (\mathcal{G}_1, \mathcal{G}_2)\,, \\ \mathcal{G}_1 = \mathbf{I}, &\quad \mathcal{G}_2(u) = \sigma(u)\cdot\mathbf{n}, \end{aligned} \tag{2.3}$$

where $\mathbf{I}$ is the identity operator, σ represents the flux or stress tensor, and $\mathbf{n}$ is the unit outer normal vector of Γ_t. For specific cases like the Stefan problem, $\mathcal{G}$ has the form:

$$\begin{aligned} \mathcal{G} &= (\mathcal{G}_1, \mathcal{G}_2)\,, \\ \mathcal{G}_1 = \mathbf{I}, &\quad \mathcal{G}_2(u) = u\chi_{\Omega_t^+}, \end{aligned} \tag{2.4}$$

where $\chi_{\Omega_t^+}$ is the characteristic function over Ω_t^+. The operator $\mathcal{M}$ in Eq. (2.1c) pertains to the motion of the interface and may be independent of other equations for known interfaces. For problems with known interfaces, $\mathcal{M}$ is expressed as

$$\mathcal{M}(u) = w\chi_{\Omega_t^+}, \tag{2.5}$$

where w represents the moving speed of the interface, typically serving as the coefficient of the first-order term in some equations. In the context of the Stefan problem, Eq. (2.1c) corresponds to the classical Stefan condition [39].

The symbols $f^\pm(\mathbf{x},t)$, $\phi_1(\mathbf{x},t)$, and $\phi_2(\mathbf{x},t)$ are known, smooth source terms specific to their respective subdomains and interface Γ_t. The symbol ψ, which represents a bounded linear operator on $(d-1)$-dimensional submanifolds, describes an unknown quantity of the free interface. Depending on the context, $\langle\psi,\Gamma_t\rangle$ may represent the normal velocity of the interface in Stefan problems [40, 41, 39], Hele-Shaw flows [42, 40], or the curvature in surface tension-driven fluid dynamics [43, 44, 45, 46, 47], among other applications.

This framework establishes conditions across the moving interface, which enable the interface to evolve as an integral part of the solution.

3. Mathemtical method

In this section, we begin by introducing a foundational neural network architecture, detailing both the network structure and the loss function. We then discuss a novel approach for incorporating discontinuities into the architecture, which leads to significant improvements in both computational speed and accuracy. This enhanced methodology is particularly effective for problems where the interface motion is independent of the governing equations. Finally, for scenarios where the interface motion is intrinsically tied to the equations, such as in the Stefan problem, a discrete-time scheme of the proposed framework is introduced to overcome this challenge.

3.1. Fundamental architecture of DELM

The physics-informed neural networks (PINNs) [48] methodology offers a general framework for solving PDEs by embedding physical laws into the neural network training process. We first primarily modify the variant called locELM [38], which is both efficient and accurate, to adapt interface problems. This adjustment is designed to set the foundation for further enhancements, which will be detailed in the following sections.

3.1.1. Full-connected neural network architectures

We begin by considering a full-connected neural network with L layers, which can be expressed in vector form as follows:

$$\mathbf{y} = W_o^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}_o^{(L+1)}, \tag{3.1}$$

where the activation vector $\mathbf{h}$ is computed through a series of hidden layers. Each hidden layer activation is given by:

$$\mathbf{h}^{(l)} = \sigma(W_h^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}_h^{(l)}), \quad 0 \le l \le L, \tag{3.2}$$

with the initial input vector $\mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^d$. Here, $W_h^{(l)} \in \mathbb{R}^{m_l \times m_{l-1}}$ denotes the weight matrix for the l-th layer, $\mathbf{b}_h^{(l)} \in \mathbb{R}^{m_l}$ is the corresponding bias vector, and $\sigma(\cdot)$ is the element-wise activation function. The output $\mathbf{y} \in \mathbb{R}^k$ is obtained through an output weight matrix $W_o^{(L+1)} \in \mathbb{R}^{k \times m_L}$ and an output bias $\mathbf{b}_o^{(L+1)} \in \mathbb{R}^k$. In this framework, d represents the input dimension, m_l denotes the number of neurons in the l-th hidden layer, and k specifies the output dimension. We define the final neural network as $\mathbf{y} = \mathbf{y}(\mathbf{x}; \theta)$, where $\theta = \{W_h^{(l)}, \mathbf{b}_h^{(l)}\}_{l=0}^{L} \cup \{W_o^{(L+1)}, \mathbf{b}_o^{(L+1)}\}$ encapsulates all the trainable parameters of the model.

The optimization target of PINNs is defined by a loss functional that integrates both data consistency and physical constraints:

$$\mathcal{L}(\mathbf{y}; \hat{\mathbf{y}}, \mathcal{N}) = \lambda_{\text{data}}\mathcal{L}_{\text{data}}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda_{\text{phy}}\mathcal{L}_{\text{phy}}(\mathcal{N}, \mathbf{y}), \tag{3.3}$$

where $\mathcal{L}_{\text{data}}$ denotes the data loss, typically calculated as the mean squared error between the predicted output $\mathbf{y}$ and the observed data $\hat{\mathbf{y}}$. The physics-based loss, $\mathcal{L}_{\text{phy}}$, enforces the governing PDE constraints by minimizing the residual of the differential operator, represented as $\mathcal{N}(\mathbf{y}) = 0$. The weighting parameters λ_{data} and λ_{phy} serve to balance the contributions from the data loss and the physics loss, respectively, allowing the model to learn both data fidelity and physical accuracy.

3.1.2. Domain decomposition and associated loss functional for interface problems

Next, we partition the domain into N subdomains $\{\Omega_i\}_{i=1}^N$, which may adjust in response to movements of the interface to capture localized properties effectively. The loss functional for each subdomain is defined as:

$$\mathcal{L}_{\text{sub}}^i = \lambda_{\text{data}}^i\mathcal{L}_{\text{data}}^i(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \lambda_{\text{phy}}^i\mathcal{L}_{\text{phy}}^i(\mathcal{N}, \mathbf{y}^i), \tag{3.4}$$

where $\mathcal{L}^i_{\text{sub}}$ represents the combined data and physics loss within the i-th subdomain. The overall loss functional is then given by:

$$\mathcal{L}_{\text{total}} = \sum_{i=1}^{N} \mathcal{L}^i_{\text{sub}} + \lambda_{\text{c-s}} \mathcal{L}_{\text{c-s}}, \tag{3.5}$$

where $\mathcal{L}_{\text{c-s}}$ denotes the cross-subdomain constraint loss, ensuring continuity and consistency of the solution across the junctions between subdomains. The weight $\lambda_{\text{c-s}}$ controls the influence of these cross-subdomain constraints within the total optimization process.

We now proceed to provide the specific formulation for each subdomain. For each subdomain Ω_i, which differs from the previously mentioned $\Omega_t^{\pm}$, a neural network is trained, enabling the final solution to be represented as:

$$\mathbf{y}(\mathbf{x};\theta) = \sum_{i=1}^{N} \mathbf{y}_i(\mathbf{x};\theta_i)\chi_i, \tag{3.6}$$

where χ_i is the indicator function associated with subdomain Ω_i, and $\mathbf{y}_i$ and θ_i denote the neural network and its parameters corresponding to subdomain Ω_i. The physics-based loss within each subdomain Ω_i is defined by Eqs. (2.1a)-(2.1e):

$$\begin{aligned}
\mathcal{L}^i_{\text{phy}}(\mathbf{y}) = &\|(\partial_t + \mathcal{L}^{\pm})(\mathbf{y})\chi_i - f^{\pm}\chi_i\|^2_{L^2(0,T;L^2(\Omega_i))} + \|\mathcal{B}(\mathbf{y})\chi_i - g\chi_i\|^2_{L^2(0,T;L^2(\partial\Omega))} \\
&+ \|\mathcal{I}(\mathbf{y})\chi_i - h\chi_i\|^2_{L^2(\Omega)} + \left\|[\![\mathcal{G}_1\mathbf{y}]\!]_{\Gamma_t}\chi_i - \phi_1\chi_i\right\|^2_{L^2(0,T;L^2(\Gamma_t))} \\
&+ \left\|[\![\mathcal{G}_2\mathbf{y}]\!]_{\Gamma_t}\chi_i - \phi_2\chi_i\right\|^2_{L^2(0,T;L^2(\Gamma_t))} + \left\|[\![\mathcal{M}\mathbf{y}]\!]_{\Gamma_t}\chi_i - \langle\psi,\Gamma_t\rangle\chi_i\right\|^2_{L^2(0,T;L^2(\Gamma_t))}.
\end{aligned} \tag{3.7}$$

The notation $L^2(0,T;L^2(D))$ denotes the space of functions that are square-integrable over both the temporal interval $(0,T)$ and the spatial manifold D. This means that for a function $f(t,\mathbf{x})$ in $L^2(0,T;L^2(D))$, where D may be the domain Ω or the submanifold Γ_t, we have:

$$||f||_{L^2(0,T;L^2(D))} = \left(\int_0^T \int_D |f(t,\mathbf{x})|^2 \, d\mathbf{x}\, dt\right)^{1/2}. \tag{3.8}$$

Let the set of junctions between subdomains be denoted by $\hat{\Gamma} = \bigcup_{i=1}^{N} \partial\Omega_i - \partial\Omega$. We enforce the continuity of function values and their gradients across these junctions with the following junction loss:

$$\mathcal{L}_{\text{c-s}}(\mathbf{y}) = \|[\![\mathbf{y}]\!]\|^2_{L^2(0,T;L^2(\hat{\Gamma}))} + \|[\![\nabla\mathbf{y}\cdot\mathbf{n}]\!]\|^2_{L^2(0,T;L^2(\hat{\Gamma}))}. \tag{3.9}$$

In this study, there is no data loss functional in Eq. (3.4), and thus the total loss functional is defined as follows:

$$\mathcal{L}_{\text{total}} = \sum_{i=1}^{N} \mathcal{L}^i_{\text{phy}} + \mathcal{L}_{\text{c-s}}. \tag{3.10}$$

3.1.3. Efficiency enhancement

To further simplify and enhance computational efficiency, we utilize a single hidden layer, with the weights in the first layer fixed to be uniformly distributed within the interval $[-R, R]$, where $R \in \mathbb{R}$ is a hyperparameter. The bias term $\mathbf{b}_o^{(L+1)}$ in the output layer is set to zero, so that the parameter set θ_i consists solely of the weights of the final layer, denoted by W_o^i. By minimizing the objective function defined in Eq. (3.10), we obtain the numerical solution to the equation. For additional details, please refer to [38].

By embedding physical laws directly into the learning process, locELM provides a flexible and efficient approach for solving complex PDEs. Furthermore, locELM offers notable advantages over conventional methods: it achieves higher accuracy and faster computation times when the solution is sufficiently smooth. Compared to standard PINNs, locELM also demonstrates improved accuracy and speed while using fewer parameters, making it an efficient choice for solving PDEs.

3.2. Artificial discontinuities in DELM

However, directly applying the method from Section 3.1 is inefficient for the types of interface problems considered in this paper. These problems, described by Eqs. (2.1a)-(2.1e), which often exhibit jumps in both function values and gradients at the interface, present a significant challenge. To address this, we introduce "artificial discontinuities" into the locELM, which allows our method to better adapt to the problem's inherent characteristics.

Traditional PINNs, when facing such challenges, typically require an increase in the number of parameters and collocation points [29, 49], adaptive sampling techniques [28, 50, 51, 52], or the construction of multiple networks [28]. Our work proposes a novel approach to address these challenges efficiently. To maintain the efficiency of our approach, we aim to incorporate these discontinuities into the locELM without adding to the computational burden, thereby enabling the efficient approximation of non-smooth solutions using smooth neural networks.

To address discontinuities in the function values, we introduce an augmented variable defined by

$$\hat{z} = \begin{cases} -1 & \text{if } \mathbf{x} \in \Omega^-, \\ 1 & \text{if } \mathbf{x} \in \Omega^+. \end{cases} \tag{3.11}$$

This augmented variable effectively transforms the original d-dimensional neural network into a $(d+1)$-dimensional network, with $\hat{z}$ serving as an additional coordinate near the interface. Since $\hat{z}$ is a piecewise constant variable, its weak derivative is zero, preserving the form of the governing equations. This approach enables us to introduce discontinuities in the function values.

Furthermore, to address gradient discontinuities, we begin by introducing a signed-distance level set function, denoted as ϕ, which characterizes the interface. The zero level set of this function corresponds to the location of the interface. This level set function satisfies

the Eikonal equation [53, 7, 54]:

$$\begin{cases} |\nabla\phi(\mathbf{x})| = 1, & \mathbf{x} \in \Omega, \\ \phi(\mathbf{x}) = 0, & \mathbf{x} \in \Gamma, \end{cases} \tag{3.12}$$

where Ω represents the domain, and Γ denotes the interface.

We then augment the absolute value of ϕ as an additional variable in the neural network. The numerical solution to Eqs. (2.1a)-(2.1e) is expressed as

$$u(\mathbf{x}, t) = U(\mathbf{x}, t, z = \phi_a(\mathbf{x}, t)), \tag{3.13}$$

where U represents the neural network and $\phi_a = |\phi|$. Using smooth activation functions in the neural network and the chain rule, the gradient of the solution can be written as

$$\nabla u = \nabla_{\mathbf{x}} U + \partial_z U \nabla_{\mathbf{x}} \phi_a, \tag{3.14}$$

where $\nabla\phi_a$ denotes the weak gradient of ϕ_a. Consequently, due to the smoothness properties of neural networks, the jump discontinuity in the gradient at the interface is given by

$$[\![\nabla u]\!](\mathbf{x}_{\mathbf{\Gamma}}, t) = 2\partial_z U \nabla_{\mathbf{x}} \phi(\mathbf{x}_{\mathbf{\Gamma}}, t). \tag{3.15}$$

Since ϕ is a signed-distance function, the unit normal vector at the interface is $\nabla\phi$. Together with the Eikonal equation (3.12), this yields

$$[\![\partial_{\mathbf{n}} u]\!](\mathbf{x}_{\mathbf{\Gamma}}) = 2\partial_z U. \tag{3.16}$$

Similarly, the time derivative of the numerical solution can be expressed as

$$\partial_t u = \partial_t U + \partial_z U \partial_t \phi_a. \tag{3.17}$$

This augmentation introduces gradient jump discontinuities within the neural network. Readers only need to substitute the corresponding terms into the equations.

We summarize our algorithm, DELM, in Algorithm 1, with a full architecture illustrated in Fig. 2. This approach enhances solution accuracy, making the method described in Section 3.1 well-suited for interface problems. While the basic concept of this method builds on the work of Lai et al. [31, 32], our improved version introduces three key differences and advantages over the original approach.

First, we freeze the weights in the first layer of the neural network (as discussed in Section 3.1), allowing the natural introduction of two new variables without increasing the total parameter count—further reducing parameters compared to Lai et al.'s methods. Additionally, by using different level set functions that satisfy Eikonal equations (3.12), we successfully extend the framework to a dynamic interface with a known interface and free boundary problems with an unknown interface (see Section 3.3), which enables efficient solutions across a wider range of challenges. Finally, this network configuration maintains the same accuracy level while significantly accelerating training times.

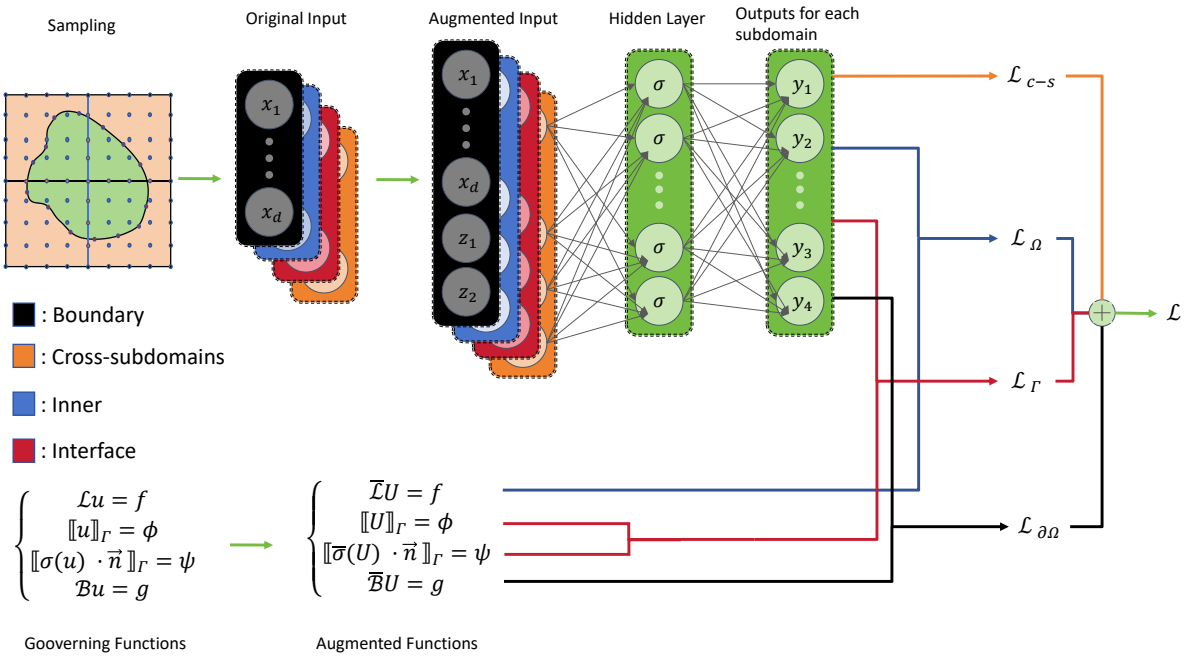


Figure 2: Architecture of DELM.

Algorithm 1 DELM (Discontinuous extreme learning machine)

Input: Training points $\{x_i^{\Omega_t^{\pm}}\}_{i=1}^{Q\times N_t}$, $\{x_i^{\text{c-s}}\}_{i=1}^{N_e\times N_b\times N_t}$, $\{x_j^{\Gamma_t}\}_{j=1}^{N_I\times N_t}$, $\{x_k^b\}_{k=1}^{2N_e\times N_b\times N_t}$.

0. Augment the input and equations through Eqs. (3.11), (3.14)-(3.17);

1. Construct a single-layer neural network with N_e outputs and assign weights and biases $\left\{W_h^{(0),j}, \mathbf{b}_h^{(0),j}\right\}_{j=1}^{N_e}$ with random values uniformly distributed over $[-R, R]$;

2. Compute the errors $\left\{\mathcal{L}_{\text{phy}}^i\right\}_{j=1}^{N_e}$ and $\mathcal{L}_{\text{c-s}}$ using augmented inputs and equations for each subdomain;

3. Minimize $\mathcal{L}_{\text{total}}$ using the least squares method to determine the optimal weight matrices $\{W_o^j\}_{j=1}^{N_e}$;

Output: Weights $\{W_o^j\}_{j=1}^{N_e}$ that define the DELM approximate solution.

3.3. Discrete DELM for free boundary problems

Sections 3.1 and 3.2 introduced a continuous solution method that is directly applicable when the interface movement is independent of the governing equations (for example, when flow terms are known and do not depend on the equations [36]). However, in cases where the interface motion is coupled with the governing equations, as in the Stefan problem, a time-discretized version of the method is required.

3.3.1. Time-discretization

First, given positive integer N_{steps}, we assume a uniform partition of the time interval $[0, T]$ as

$$0 = t_0 < t_1 < \cdots < t_n < \cdots < t_{N_{\text{steps}}-1} < t_{N_{\text{steps}}} = T \tag{3.18}$$

with a fixed time step size $\Delta t = T/N_{\text{steps}}$ and $t_n = n\Delta t$ for $0 \le n \le N_{\text{steps}}$. We then decouple the system and discretize it using an explicit-implicit Euler scheme:

$$\begin{cases} \dfrac{u^{n+1} - u^n}{\Delta t} + \mathcal{L}^{\pm} u^{n+1} = f^{\pm,n+1}, & \mathbf{x} \in \Omega^{\pm}_{n+1}, & \text{(3.19a)} \\ [\![\mathcal{G} u^{n+1}]\!]_{\Gamma^{n+1}} = \left(\phi_1^{n+1}, \phi_2^{n+1}\right), & \mathbf{x} \in \Gamma^{n+1}, & \text{(3.19b)} \\ [\![\mathcal{M} u^n]\!]_{\Gamma_n} = \langle \psi^{n+1}, \Gamma^{n+1} \rangle, & \mathbf{x} \in \Gamma^n, & \text{(3.19c)} \\ \mathcal{B} u^{n+1} = g^{n+1}, & \mathbf{x} \in \partial\Omega, & \text{(3.19d)} \\ \mathcal{I} u^0 = h, & \mathbf{x} \in \Omega. & \text{(3.19e)} \end{cases}$$

Here, the superscript n denotes the corresponding approximations at time instant t_n. We start by Eq. (3.19e) to update the interface through Eq. (3.19c), and then proceed to solve the remaining equations. This process is repeated iteratively until the temporal evolution is complete.

Taking the Stefan problem [55, 56] as an example, we present this discretized formulation, which models phase transitions such as the melting of ice into water. In the Stefan problem, the domain Ω is divided into two regions, Ω_t^+ and Ω_t^-, separated by a moving interface Γ_t, which represents the phase boundary between the solid and liquid phases. The position of the interface Γ_t is generally unknown and must be determined as part of the solution.

Example 1. A Stefan problem can be formulated as:

$$\begin{cases} \rho^{\pm} c^{\pm} \dfrac{\partial u^{\pm}}{\partial t} - \nabla \cdot (k^{\pm} \nabla u^{\pm}) = f^{\pm}(\mathbf{x}, t), & (\mathbf{x}, t) \in \Omega_t^{\pm} \times [0, T], & \text{(3.20a)} \\ u^+(\mathbf{x}, t) = u^-(\mathbf{x}, t) = u_m, & (\mathbf{x}, t) \in \Gamma_t \times [0, T], & \text{(3.20b)} \\ k^+ \nabla u^+ \cdot \mathbf{n} - k^- \nabla u^- \cdot \mathbf{n} = \rho L V_n, & (\mathbf{x}, t) \in \Gamma_t \times [0, T], & \text{(3.20c)} \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), & \mathbf{x} \in \Omega, & \text{(3.20d)} \\ u(\mathbf{x}, t) = g(\mathbf{x}, t), & (\mathbf{x}, t) \in \partial\Omega \times [0, T]. & \text{(3.20e)} \end{cases}$$

Here, $u^{\pm}(\mathbf{x}, t)$ represents the temperature in regions $\Omega_t^{\pm}$; $\rho^{\pm}$ is the density; $c^{\pm}$ is the specific heat capacity; $k^{\pm}$ is the thermal conductivity; u_m is the melting temperature; L is the latent heat of fusion; V_n is the normal velocity of the moving interface Γ_t; $\mathbf{n}$ is the unit normal vector pointing from Ω_t^- to Ω_t^+; $u_0(\mathbf{x})$ is the initial temperature distribution; and $g(\mathbf{x}, t)$ specifies the boundary temperature distribution.

The condition $u^+ = u^- = u_m$ on Γ_t ensures that the temperature remains continuous across the phase boundary at the melting temperature. The jump in heat flux across Γ_t is balanced by the latent heat released or absorbed during the phase change, as expressed in Eq. (3.20c), which involves the normal velocity V_n.

The Stefan problem is a classic example of a free boundary problem, where the interface Γ_t evolves over time and must be determined as part of the solution. It has significant applications in fields such as metallurgy [57], cryogenics [58], and geophysics [59], where understanding the dynamics of phase transitions is essential.

We now illustrate how to handle Example 1, starting by discretizing it in time. For Eq. (3.20a), we apply the implicit Euler method for time discretization and rearrange it as follows:

$$\rho^{\pm}c^{\pm}u_{\pm}^{(n+1)} - \Delta t\nabla\cdot\left(k^{\pm}\nabla u_{\pm}^{(n+1)}\right) = \Delta t f^{\pm}\left(x, t^{n+1}\right) + \rho^{\pm}c^{\pm}u_{\pm}^{n}, \quad x \in \Omega_{n+1}^{\pm}, \tag{3.21}$$

where $u_{\pm}^{n}$ represents the solution at the n-th time step within each respective region. For Eq. (3.20c), which expresses the Stefan condition, we discretize it using the explicit Euler method and rearrange it as follows:

$$\rho L S^{n+1} = \rho L S^{n} + \Delta t\left(k^{+}\nabla u_{+}^{n}\cdot\mathbf{n}^{n} - k^{-}\nabla u_{-}^{n}\cdot\mathbf{n}^{n}\right), \quad x \in \Gamma^{n}. \tag{3.22}$$

Here, S^n denotes the position of the interface at the n-th time step, and $\mathbf{n}^n$ is the normal vector of the interface at the n-th time step.

The steps to solve the problem are as follows: first, select discrete points for the initial interface position, denoted by $\{\mathbf{r}_i^0\}_{i=1}^N$, and update the interface position according to the initial conditions as described in Eq. (3.22). Next, update the solution at the subsequent time step using Eq. (3.21), following the procedure outlined in Sections 3.1 and 3.2 to obtain the solution.

3.3.2. Artificial discontinuities in discrete DELM

When solving the problem using the methods from Sections 3.1 and 3.2, an important step is to incorporate augmented variables. In addition to calculating the position of the interface at each time step, we also need to compute the signed level set function, along with its gradient and Laplacian. In one-dimensional cases, once the interface position is determined at each time step using Eq. (3.22), this information can be obtained with minimal effort.

In higher-dimensional cases, however, some additional steps are required. Here, we focus on the two-dimensional case for illustration, as the extension to higher dimensions follows a similar approach. In the two-dimensional scenario, the set of points $\{\mathbf{r}_i\}_{i=1}^N$ can be interpolated using an arc-length parameterized spline curve [60], denoted by $\mathbf{r}(s)$. For any point $\mathbf{x}$ in the plane, the closest parameter s_{near} is given by

$$s_{\text{near}} = \text{argmin}_{s\in[0,1]}\left|\mathbf{x} - \mathbf{r}(s)\right|. \tag{3.23}$$

We do not need to solve this minimization problem directly; instead, we can efficiently obtain the result by constructing a kd tree [61]. The level set function in Eq. (3.12) is then defined as

$$\phi(\mathbf{x}) = \text{sign}\left(\left(\mathbf{x} - \mathbf{r}(s_{\text{near}})\right)\cdot\mathbf{n}(s_{\text{near}})\right)\left|\mathbf{x} - \mathbf{r}(s_{\text{near}})\right|, \tag{3.24}$$

where $\mathbf{n}(s_{\text{near}})$ is the unit inner normal vector at the closest point on the interface. This approach allows us to efficiently compute the signed level set function for use in higher-dimensional cases.

Using the properties of the signed distance function, we can readily obtain the gradient of the level set function:

$$\nabla\phi(\mathbf{x}) = -\text{sign}(1 - \phi(\mathbf{x}) \cdot \kappa(s_{\text{near}}))\mathbf{n}(s_{\text{near}}), \tag{3.25}$$

where $\mathbf{n}$ and κ represent the unit inward normal vector and curvature at the corresponding point on the curve, respectively. With this expression, we can easily derive the Laplacian of the level set function as

$$\Delta\phi(\mathbf{x}) = \text{sign}(1 - \phi(\mathbf{x}) \cdot \kappa(s_{\text{near}}))\frac{\kappa(s_{\text{near}})}{|1 - \phi(\mathbf{x}) \cdot \kappa(s_{\text{near}})|}. \tag{3.26}$$

In fact, $\Delta\phi$ and $\nabla\phi$ represent, respectively, the curvature and the unit outward normal vector of the interface curve $\mathbf{r}$ as it is translated along the normal direction according to the given parameters. We describe the translated curve as

$$\mathbf{r}_{\text{new}}(s) = \mathbf{r}(s) + c\mathbf{n}(s), \tag{3.27}$$

where $c \in \mathbb{R}$ denotes the properly small translation distance. Using arc-length parameterization, we compute the tangent vectors of both the original and translated curves, as well as the curvature of the original curve:

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s} = \mathbf{t}, \quad \kappa = \mathbf{n} \cdot \frac{\mathrm{d}\mathbf{t}}{\mathrm{d}s}, \quad \frac{\mathrm{d}\mathbf{r}_{\text{new}}}{\mathrm{d}s} = \mathbf{t} + c\frac{\mathrm{d}\mathbf{n}}{\mathrm{d}s} = \mathbf{t} - c\kappa\mathbf{t} = (1 - c\kappa)\mathbf{t}. \tag{3.28}$$

where $\mathbf{t}$ is the unit tangent vector. By normalizing the tangent vector of the translated curve, we obtain

$$\mathbf{t}_{\text{new}} = \frac{\frac{\mathrm{d}\mathbf{r}_{\text{new}}}{\mathrm{d}s}}{\left\|\frac{\mathrm{d}\mathbf{r}_{\text{new}}}{\mathrm{d}s}\right\|} = \frac{(1 - c\kappa)\mathbf{t}}{|1 - c\kappa|}. \tag{3.29}$$

This expression corresponds to Eq. (3.25) after a clockwise rotation. Next, the relationship between the differential forms of the old and new curve parameters is given by

$$\mathrm{d}s_{\text{new}} = (1 - c\kappa)\mathrm{d}s. \tag{3.30}$$

Finally, we obtain the curvature of the new curve at the corresponding point as

$$\kappa_{\text{new}} = \mathbf{n}_{\text{new}} \cdot \frac{\mathrm{d}\mathbf{t}_{\text{new}}}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}s_{\text{new}}} = \frac{\kappa(1 - c\kappa)}{|1 - c\kappa|^2}. \tag{3.31}$$

This result corresponds to Eq. (3.26).

We summarize our discrete version in Algorithm 2. Ultimately, we have successfully extended our method to dynamic interface and free boundary problems, covering both discretized and continuous-time versions. When using the discretized time approach, this

method can easily be extended to higher-order schemes in time, such as Runge-Kutta [62] and BDF methods [63], if greater accuracy in time discretization is required. This enables us to retain the high-accuracy properties of traditional methods in the temporal direction while benefiting from neural networks in the spatial domain. Whether applied in the discrete or continuous form, our approach can efficiently and accurately handle interface and free boundary problems.

Algorithm 2 Discrete DELM

Input: The training points $\{x_i^{\Omega_t^{\pm}}\}_{i=1}^{Q}$, $\{x_i^{\text{c-s}}\}_{i=1}^{N_e\times N_b}$, $\{x_j^{\Gamma_t}\}_{j=1}^{N_I}$, $\{x_k^b\}_{k=1}^{2N_e\times N_b}$ and time steps N_{steps}.

1. Update the interface $\{x_j^{\Gamma_t}\}_{j=1}^{N_I}$ through Eqs. (3.19c), (3.19e);
2. Calculate gradient and Laplace information of the signed distance function ϕ through Eqs. (3.23)-(3.26);
3. Augment the input and equations through Eqs. (3.11), (3.14)-(3.17);
4. Conduct a single-layer neural networks with N_e outputs and assign the weights and biases $\left\{W_h^{(0),j}, \mathbf{b}_h^{(0),j}\right\}_{j=1}^{N_e}$ with uniform random values generated from the interval $[-R, R]$;
5. Define the errors $\left\{\mathcal{L}_{\text{phy}}^i\right\}_{j=1}^{N_e}$ and $\mathcal{L}_{\text{c-s}}$ through augmentation of Eqs. (3.19a), (3.19b), (3.19d) for each subdomains;
6. Minimize $\mathcal{L}_{\text{total}}$ by using the least square method to determine the optimal weight matrix $\{W_o^j\}_{j=1}^{N_e}$;
7. Repeat above steps N_{steps} times;

Output: Weights $\{W_o^j\}_{j=1}^{N_e}$ that fix the discrete DELM approximate solution.

4. Numerical results

4.1. Experimental setup

In this study, the entire domain is subdivided into uniformly distributed subdomains along each spatial direction, with the total number of subdomains denoted as N_e. For each subdomain, the number of neural network parameters is represented by M. Unless otherwise specified, point sampling follows a uniform distribution. The training points located on each boundary of a subdomain are indicated by N_b, and the training points on the interface are denoted by N_I. The total number of collocation points within each subdomain is represented by Q. The sampling method is illustrated in Fig. 3 and the total number of parameters can be calculated as $M_T = MN_e$. To ensure that junctions and interfaces do not conflict, it is necessary to maintain a certain distance between them during the setup of junctions. Specifically, if the distance between an interface and a junction is smaller than the distance between two adjacent junctions, the nearest junction will be removed. For practical purposes, the collocation points located on the interface are omitted to enhance computational efficiency. In the discretized version of the method, the number of time steps is denoted by N_{steps}.

The DELM or discrete DELM has a relatively small number of parameters. Therefore, for solving linear systems, a linear least-squares optimization method is used. For nonlinear

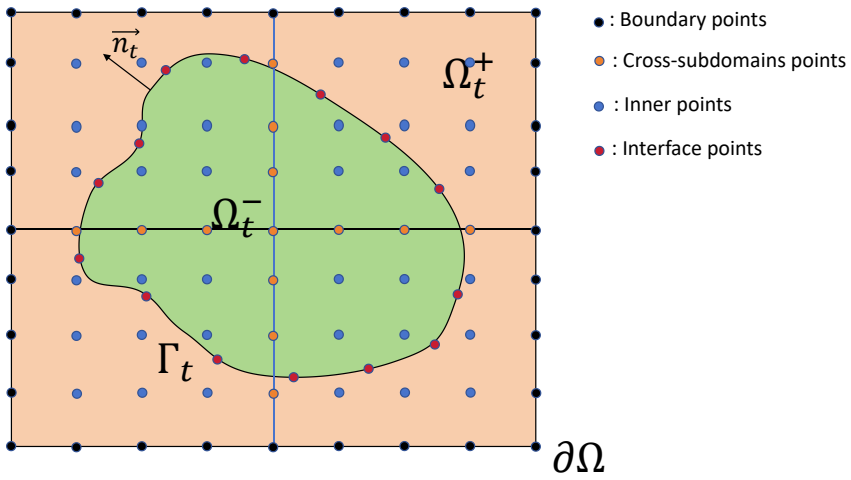


Figure 3: Points Sampling.

systems, we adopt a nonlinear least-squares optimization approach as described in [38]. These optimization methods, together with parametric spline interpolation and k-d tree construction, are implemented using the SciPy package in Python. The backpropagation process is handled by PyTorch, and for all experiments conducted in this study, random seeds are consistently set to 3407 to ensure reproducibility [64]. All programs are executed on an AMD R7-4800H CPU, providing a consistent computational environment for benchmarking and analysis.

4.2. Interface problems with known interfaces

4.2.1. Elliptic interface problem

To validate the effectiveness of the proposed method for addressing interface problems, we begin by considering a two-dimensional elliptic interface problem. The computational domain is defined as the square $\Omega = [0,1]\times[0,1]$, where a discontinuous coefficient is present, and the interface is represented by the zero level set of the function $\phi(x,y) = \frac{x^2}{0.5^2} + \frac{y^2}{0.5^2} - 1$. This level set divides Ω into two subdomains: Ω^- and Ω^+. The governing equation for the problem is given by:

$$\begin{cases} \nabla\cdot(\beta(\mathbf{x})\nabla u(\mathbf{x})) - \alpha(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x}\in\Omega^{\pm}, \\ [\![u]\!](\mathbf{x}_\Gamma) = 0, & \mathbf{x}_\Gamma\in\Gamma, \\ [\![\beta\partial_n u]\!](\mathbf{x}_\Gamma) = \rho(\mathbf{x}_\Gamma), & \mathbf{x}_\Gamma\in\Gamma, \\ u(\mathbf{x}_B) = g(\mathbf{x}_B), & \mathbf{x}_B\in\partial\Omega. \end{cases} \tag{4.1}$$

In this formulation, $u(\mathbf{x})$ is the solution function, $\rho(\mathbf{x}_\Gamma)$ and $g(\mathbf{x}_B)$ are given smooth functions, $\alpha(\mathbf{x}) \geq 0$, and $f(\mathbf{x})$ and $\beta(\mathbf{x}) > 0$ are defined in a piecewise smooth manner across the interface Γ. We select the exact solution $u(x,y)$ and the coefficient $\beta(x,y)$ as follows:

$$u(x,y) = \begin{cases} 1 - \exp\left(\frac{1}{\eta}\left(\frac{x^2}{0.5^2} + \frac{y^2}{0.5^2} - 1\right)\right), & (x,y) \in \Omega^-, \\ -\gamma \ln\left(\frac{x^2}{0.5^2} + \frac{y^2}{0.5^2}\right), & (x,y) \in \Omega^+, \end{cases} \tag{4.2}$$

$$\beta(x,y) = \begin{cases} \beta^-, & (x,y) \in \Omega^-, \\ \beta^+, & (x,y) \in \Omega^+. \end{cases} \tag{4.3}$$

Here, $\eta = \beta^-/\beta^+$ represents the contrast ratio between the coefficients in the two subdomains. In this example, we set $\beta^+ = 1$ and $\eta = 10$ to introduce a significant contrast that effectively challenges our method.

The parameters $\alpha = 1$ and $\gamma = 2$ are chosen to suit the exact solution form. For further details on the setup and solution, we refer to the work of Lai et al. [32]. The parameter settings for this test are summarized in Table 1. In this configuration, each spatial dimension is divided into two subdomains, with 8 boundary points and 36 interface points. Each subdomain has 8×8 collocation points. The number of neurons per subdomain ranges from 30 to 100, depending on the specific conditions. The parameter R is set to 0.2. The total number of training points used in this problem is 388, with the total number of parameters ranging between 120 and 400.

Table 1: Parameter settings for the elliptic interface problem.

N_e	N_b	N_I	Q	M	R	N_T	M_T
2×2	8	36	8×8	30-100	0.2	388	120-400

We employ the corresponding method from the work by Lai et al., known as cusp-capturing PINN (C-C PINN) [32], for comparison. The computational results are summarized in Table 2. Additionally, we plotted the error variation with respect to the number of parameters, as illustrated in Fig. 4. Our method can be consistently applied to all examples in this paper using the same random seed, while traditional neural network algorithms typically require different random seeds for each problem, or even for each combination of hyperparameters, to achieve optimal performance. Consequently, we conducted only one experiment for each parameter setting with our method. For the other methods, we present both the results from a single experiment and the average results obtained from five experiments, following the approach outlined in [32]. To ensure consistency and comparability, the random seed used in the single experiment was set to match the one utilized in our method.

We observe that, with a relatively small number of parameters, the model error remains significant; however, each region already reaches an L^2 error level of $O(10^{-3})$ with only 30 parameters per subdomain. As the parameters per subdomain increase from 30 to 100, the error decreases from $O(10^{-3})$ to $O(10^{-7})$, eventually slightly surpassing the accuracy of C-C PINN. At this stage, our method still retains advantages in terms of training speed, enabling rapid resolution of moving interface problems (see Sections 4.3.1 and 4.3.2). For readers

Table 2: Summary of computational results for the elliptic interface problem.

Method	$\frac{\|\hat{u}-u\|_2}{\|u\|_2}$	(M, M_T)	(Q, N_T)	# **Epochs**
DELM	6.0778e-03	(30, 120)	(64, 388)	1
	2.8932e-03	(40, 160)		
	3.7122e-04	(50, 200)		
	1.1516e-04	(60, 240)		
	1.4971e-05	(70, 280)		
	2.9372e-06	(80, 320)		
	2.1073e-06	(90, 360)		
	2.9523e-07	(100, 400)		

prioritizing training speed, our method significantly outperforms traditional approaches, particularly in fields such as real-time systems and high-frequency trading. Conversely, for readers focused on minimizing parameter count when solving elliptic interface problems, C-C PINN is more advantageous. This is due to our model's structure: it consists of only one hidden layer, and only the parameters in the final layer are trainable, which can limit nonlinearity when the parameter count is low.

The results in Fig. 4 highlight the impact of random seed selection on traditional neural network methods, where a single experiment often fails to reliably reflect convergence. In contrast, our method remains unaffected by random seed variability, a crucial advantage for dynamic problems, as it prevents unnecessary error accumulation and helps maintain high accuracy. Additionally, our method demonstrates a significantly faster neural scaling law compared to C-C PINN., achieving approximately $O(M^{-9})$, a substantial improvement over C-C PINN's $O(M^{-6})$.

After this example, we present error plots for $M = 100$, as shown in Fig. 5. The overall error is minimal, with L^∞ and L^2 errors on the order of 10^{-6} and 10^{-7}, respectively. These results were obtained using only 384 points and 100 parameters per subdomain while maintaining a very high solution speed. Notably, the majority of the error is concentrated near the interface, which is expected due to the gradient discontinuities normal to the interface. This concentration of error at the interface poses a particular challenge for dynamic systems, especially in Stefan-type problems, where it becomes even more pronounced in two-dimensional cases (see Section 4.3.2). However, as discussed earlier, the rapid neural scaling law of our method effectively mitigates this potential bottleneck.

In summary, our method achieves the desired accuracy for the elliptic interface problem, combining high precision with extremely fast training speed. The method's capability for rapid neural scaling laws lays a strong foundation for efficiently addressing moving interface problems in future applications.

4.2.2. Dynamic problem

To further demonstrate the applicability of discrete DELM, we present an example involving a dynamic interface problem governed by parabolic-type equations over a two-dimensional

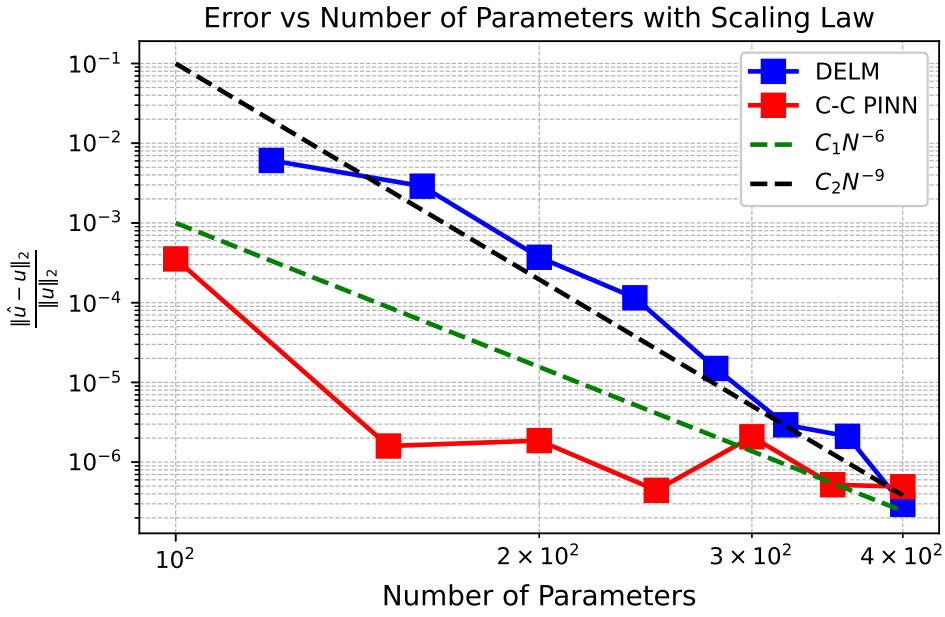


Figure 4: Error variation with respect to the number of parameters.

square domain $\Omega = [0,1]\times[0,1]$, containing discontinuous coefficients, which was also tested by [36]. The exact solution is defined as follows:

$$u(x,y,t) = \begin{cases} (t+1)\left(1-\exp\left(\frac{1}{\eta}\left(\frac{x^2}{(0.5t+0.35)^2}+\frac{y^2}{(0.5t+0.35)^2}-1\right)\right)\right), & (x,y)\in\Omega^-, \\ -\gamma(t+1)\ln\left(\frac{x^2}{(0.5t+0.35)^2}+\frac{y^2}{(0.5t+0.35)^2}\right), & (x,y)\in\Omega^+, \end{cases} \tag{4.4}$$

and the interface is given by the level set function

$$\phi(x,y,t) = \frac{x^2}{(0.5t+0.35)^2}+\frac{y^2}{(0.5t+0.35)^2}-1. \tag{4.5}$$

As in the previous example, the parameter β is defined as Eq. (4.3), a piecewise constant. In this example, we set the parameters as follows: $\beta^+ = 1$, $\eta = 10$ (indicating high contrast), $\alpha = 0$, and $\gamma = 0.875$. This configuration provides a challenging test case to effectively evaluate the accuracy of our method. The example itself is adapted from the elliptic problem discussed in the previous section, allowing us to explore the method's performance under similar conditions.

The neural network parameter settings for this dynamic problem are summarized in Table 3. In this configuration, each spatial dimension is divided into two subdomains, with 15 boundary points and 80 interface points. Each subdomain contains 15×15 collocation points. The number of neurons per subdomain ranges from 51 to 93, depending on the time step length. The parameter R is set to 0.65. The total number of training points used is 1160, with the total number of parameters ranging between 201 and 372.

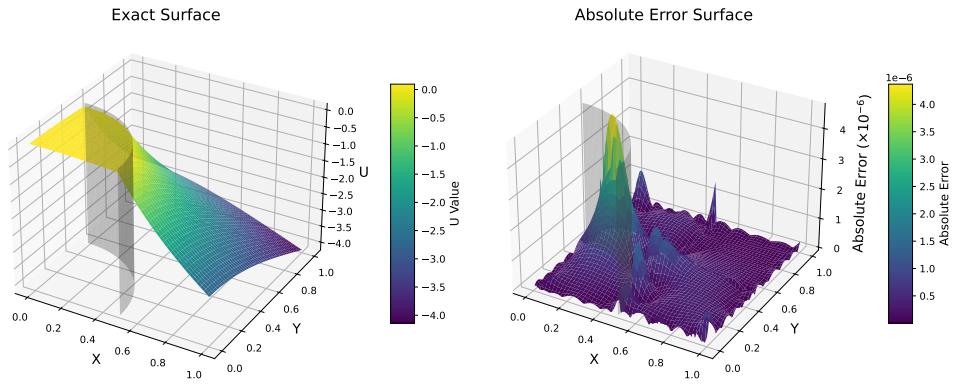


Figure 5: Exact surface and absolute error surface for the elliptic interface problem.

Table 3: Parameter Settings for the dynamic problem.

N_e	N_b	N_I	Q	M	R	N_T	M_T
2×2	15	80	15×15	51-118	0.65	1160	204-472

The optimal parameter settings at each time step, along with the corresponding errors and temporal convergence order, are presented in Table 4. The data includes spatial parameters

Table 4: Error and convergence order for various time steps and parameters in the dynamic problem.

(M, M_T)	(Q, N_T)	N_{steps}	$\|u - \hat{u}\|_{L^2(0,T;\Omega)}$	Order
(51, 204)	$(15 \times 15, 1160)$	2	0.001243	-
(57, 228)		4	0.0006104	1.03
(65, 260)		8	0.0003042	1.00
(71, 284)		16	0.0001505	1.02
(78, 312)		32	7.3874e-05	1.03
(88, 352)		64	3.6978e-05	1.00
(104, 416)		128	1.8514e-05	1.00
(118, 472)		256	9.3262e-06	0.99

(M, M_T), temporal parameters (Q, N_T), the number of time steps N_{steps}, the error norm $\|u - \hat{u}\|_{L^2(0,T;\Omega)}$, and the convergence order, providing a comprehensive overview of the method's performance across different time steps.

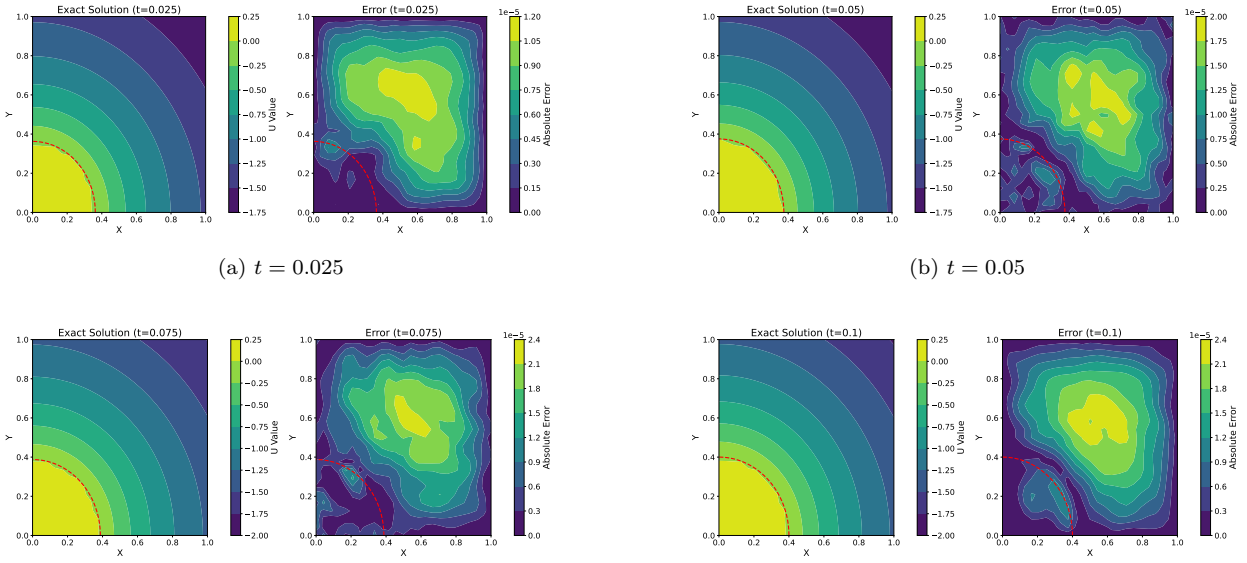
The data reveals that as the number of time steps N_{steps} increases, the error progressively decreases, indicating an optimal level of temporal convergence. For example, with an initial step count of $N_{\text{steps}} = 2$, the error is 0.001243, while increasing the steps to $N_{\text{steps}} = 256$

reduces the error to 9.3262×10^{-6}. This demonstrates that the method achieves progressively better accuracy as the time step count grows.

The table also reveals the optimal temporal convergence order, which stabilizes around 1 from $N_{\text{steps}} = 4$ onwards. For example, at step counts 4, 8, 16, 32, and 64, the convergence orders are 1.03, 1.00, 1.02, 1.03, and 1.00, respectively, indicating that the numerical method achieves the theoretically expected first-order convergence in time. This stable convergence order reflects the robustness and effectiveness of temporal discretization.

Furthermore, the spatial parameters $(Q, N_T) = (15 \times 15, 1160)$ demonstrate the method's effectiveness in achieving higher accuracy by increasing time steps under fixed spatial settings. In this configuration, the parameter count per subdomain increases from 51 to 118, resulting in 225 collocation points per subdomain, achieving spatial accuracy on the order of $O(M^{-6})$. This high-order accuracy validates the spatial NN's effectiveness in reducing numerical error.

Additionally, we plot the results for $N_{\text{steps}} = 256$, as shown in Fig. 6. In this example, the



(a) $t = 0.025$ (b) $t = 0.05$

(c) $t = 0.075$ (d) $t = 0.1$

Figure 6: Exact solution and absolute error for the dynamic problem.

largest error occurs in the outer region, followed by the area around the interface, while the error within the region's interior is minimal. As time progresses, the errors in the vicinity of the interface and within it remain at the $O(10^{-6})$ level, indicating that our discrete method effectively manages non-smoothness around the interface. The success of this example lays a strong foundation for tackling the Stefan problem in subsequent work.

4.3. Stefan Problems

4.3.1. 1D Stefan problem

We now present a one-dimensional two-phase Stefan problem as an additional example. Consider the domain $\Omega = [0, 2]$, where the interface $s(t)$ divides Ω into $\Omega^- = [0, s(t)]$ and

$\Omega^+ = [s(t), 2]$, with the time interval $t \in [0, 0.5]$. The specific formulation of the governing equations is as follows:

$$\begin{cases} \dfrac{\partial u^{\pm}}{\partial t} - \nabla \cdot (k^{\pm} \nabla u^{\pm}) = f^{\pm}(x,t), & x \in \Omega_t^{\pm}, \quad (4.6a) \\ u^+(x,t) = u^-(x,t) = 0, & x \in \Gamma_t, \quad (4.6b) \\ \alpha^+ \dfrac{\partial u^+}{\partial x} - \alpha^- \dfrac{\partial u^-}{\partial x} = \dfrac{\mathrm{d}s}{\mathrm{d}t}, & x \in \Gamma_t, \quad (4.6c) \\ u(x,0) = u_0(x), & x \in \Omega. \quad (4.6d) \end{cases}$$

The parameters for these equations are provided in Table 5. We select the exact solution as

Table 5: Parameters for the equations.

k^-	k^+	α^-	α^+
2	1	-2	1

follows:

$$u(x,t) = \begin{cases} 2\left(\exp\left(\frac{t+1/2-x}{2}\right) - 1\right), & (x,t) \in \Omega^- \times [0,T], \\ \exp(t + 1/2 - x) - 1, & (x,t) \in \Omega^+ \times [0,T], \end{cases} \quad (4.7)$$

and

$$s(t) = t + 1/2. \quad (4.8)$$

The initial and boundary conditions can be easily derived from these expressions. This example is inspired by the work of Wang et al.[29], who employed two independent neural networks to approximate $u(x,t)$ and $s(t)$. This approach avoids the complexity introduced by coupling systems, albeit at the cost of increased parameters and training time.

The parameter settings for this example are presented in Table 6. The spatial domain is divided into five subdomains, each with one boundary point on either side and one interface point. The number of collocation points and neurons per subdomain varies from 20 to 200 and 4 to 8, respectively, depending on the time step length. For all experiments in this example, the parameter R is set to 0.1. Consequently, the total number of training points and parameters ranges from 107 to 1007 and 20 to 40, respectively.

Table 6: Parameter settings for the 1D Stefan problem.

N_e	N_b	N_I	Q	M	R	N_T	M_T
5	1	1	20-200	4-8	0.1	107-1007	20-40

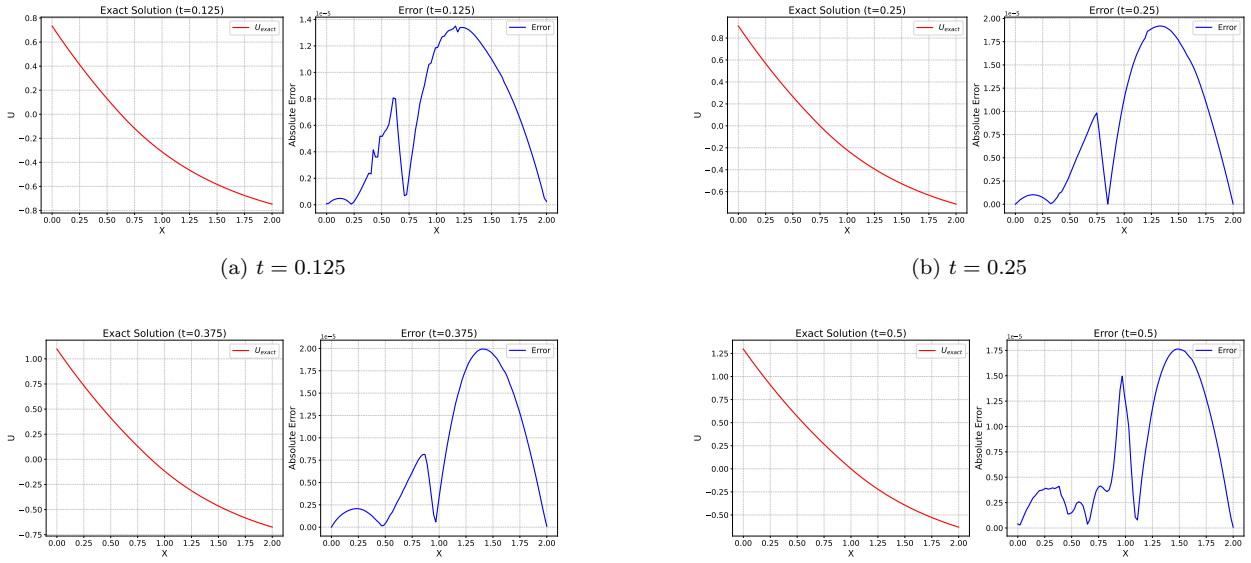
Next, we present the optimal parameter settings for each time step length, along with the computed errors and convergence order with respect to time, as shown in Table 7. As shown in Table 7, the time-related error achieves its optimal convergence order. In the spatial

Table 7: Error and convergence order for various time steps and parameters in the 1D Stefan problem.

(M, M_T)	(Q, N_T)	N_{steps}	$\|u - \hat{u}\|_{L^2(0,T;\Omega)}$	Order
(4, 20)	(20, 107)	2	0.006508	-
(5, 25)	(30, 157)	4	0.002622	1.50
(5, 25)	(30, 157)	8	0.001027	1.05
(5, 25)	(30, 157)	16	0.0005554	0.94
(5, 25)	(30, 157)	32	0.0002984	1.24
(6, 30)	(50, 257)	64	0.0001559	0.89
(6, 30)	(50, 257)	128	7.4409e-05	1.03
(7, 28)	(70, 357)	256	3.7546e-05	1.02
(8, 40)	(100, 507)	512	1.8790e-05	1.01
(8, 40)	(200, 1007)	1024	9.3975e-06	1.04

domain, the number of parameters and collocation points per subdomain increased from 4 and 20 to 8 and 200, respectively. Ultimately, by using only 8 parameters and 200 collocation points per subdomain, we achieve an accuracy of $O(10^{-6})$.

We also provide visualizations for $N_{\text{steps}} = 1024$, as shown in Fig. 7. Similar to the elliptic



(a) $t = 0.125$ (b) $t = 0.25$

(c) $t = 0.375$ (d) $t = 0.5$

Figure 7: Exact solutions and absolute errors for the 1D Stefan problem.

interface problem, a noticeable error peak appears near the interface, as discussed earlier in Section 4.2.1. From the figure, we observe that the method achieves very low errors, with L^∞ and L^2 errors on the order of $O(10^{-5})$ and $O(10^{-6})$, respectively.

In summary, our method demonstrates remarkable effectiveness in solving one-dimensional two-phase Stefan problems. Our approach not only uses fewer parameters and achieves faster

training speeds but also attains significantly higher accuracy. In the following sections, we will apply our method to address more complex Stefan problems.

4.3.2. 2D Stefan problem

To further demonstrate the effectiveness of our method, we present a two-dimensional example. Consider a square domain $\Omega = [0,1] \times [0,1]$ over the time interval $t \in [0, 0.1]$ with a circular interface defined by

$$\Gamma = \left\{ (x,y,t) \in \Omega \times [0,T] : \phi(x,y,t) = \frac{x^2}{(0.5t+0.35)^2} + \frac{y^2}{(0.5t+0.35)^2} - 1 = 0 \right\}. \quad (4.9)$$

This interface divides Ω into $\Omega^- = \{\phi < 0\} \cap \Omega$ and $\Omega^+ = \{\phi > 0\} \cap \Omega$. The governing equations for this problem are formulated as follows:

$$\begin{cases} \dfrac{\partial u^\pm}{\partial t} - \nabla \cdot (k^\pm \nabla u^\pm) = f^\pm(\mathbf{x},t), & \mathbf{x} \in \Omega_t^\pm, & (4.10a) \\ u^+(\mathbf{x},t) = u^-(\mathbf{x},t) = 0, & \mathbf{x} \in \Gamma_t, & (4.10b) \\ \alpha^+ \nabla u^+ \cdot \mathbf{n} - \alpha^- \nabla u^- \cdot \mathbf{n} = V_n, & \mathbf{x} \in \Gamma_t, & (4.10c) \\ u(\mathbf{x},0) = u_0(\mathbf{x}), & \mathbf{x} \in \Omega. & (4.10d) \end{cases}$$

The parameters for this example are identical to those used in the previous case, as outlined in Table 5. We select the exact solution as follows:

$$u(x,y,t) = \begin{cases} (t+1)\left(1 - \exp\left(\frac{1}{\eta}\left(\frac{x^2}{(0.5t+0.35)^2} + \frac{y^2}{(0.5t+0.35)^2} - 1\right)\right)\right), & (x,y) \in \Omega^-, \\ -\gamma(t+1)\ln\left(\frac{x^2}{(0.5t+0.35)^2} + \frac{y^2}{(0.5t+0.35)^2}\right), & (x,y) \in \Omega^+, \end{cases} \quad (4.11)$$

where we set $\eta = 2$ and $\gamma = \frac{7}{8}$. The initial and boundary conditions can be derived directly from the exact solution. This example is adapted from the elliptic interface problem discussed in Section 4.2.1 and presents greater challenges in terms of both computational speed and accuracy compared to the one-dimensional case. Simulating the interface and calculating the normal flux are inherently more complex in two dimensions. Nevertheless, as the results below demonstrate, our method achieves excellent performance in this challenging scenario.

We first present the parameter settings for this example, as shown in Table 8. The basic

Table 8: Parameter settings for 2D Stefan problem.

N_e	N_b	N_I	Q	M	R	N_T	M_T
2×2	15	80	15×15	51-93	0.65	1160	201-372

parameters for this example are similar to those in the elliptic interface problem. Each spatial dimension is divided into two subdomains, with 15 boundary points, 80 interface points, and 15×15 collocation points within each subdomain. The number of neurons per

subdomain ranges from 51 to 93, depending on the time step length. For all experiments in this example, the parameter R is set to 0.65. The total number of training points used is 1160, with the total number of parameters ranging between 201 and 372.

As with the one-dimensional Stefan problem, we now present the optimal parameter settings for each time step length, along with the computed errors and convergence order with respect to time, as shown in Table 9. In this example, the number of time steps increases

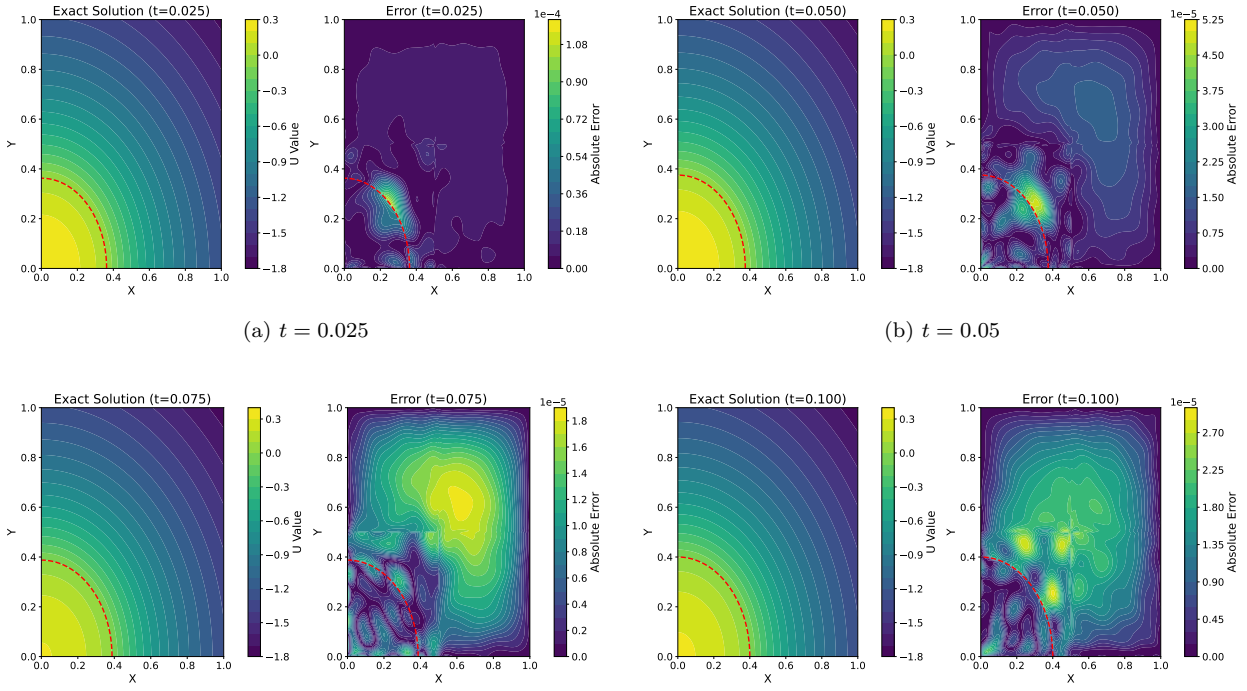
Table 9: Error and convergence order for various steps and parameters in the 2D Stefan problem.

(M, M_T)	(Q, N_T)	N_{steps}	$\|u - \hat{u}\|_{L^2(0,T;\Omega)}$	Order
(72, 288)	$(15 \times 15, 1160)$	2	0.001207	-
(89, 356)	$(15 \times 15, 1160)$	4	0.001404	-0.22
(66, 264)	$(15 \times 15, 1160)$	8	0.004678	-1.74
(51, 204)	$(15 \times 15, 1160)$	16	0.001513	1.63
(78, 312)	$(15 \times 15, 1160)$	32	0.0003836	1.98
(83, 332)	$(15 \times 15, 1160)$	64	4.5852e-05	3.06
(93, 372)	$(15 \times 15, 1160)$	128	2.2294e-05	1.04
(84, 336)	$(15 \times 15, 1160)$	256	1.0233e-05	1.12

from 2 to 256, while the number of neurons per subdomain varies from 51 to 93, showing an initial increase followed by a decrease. Unlike the one-dimensional case, the complexity of the interface simulation in two dimensions requires smaller time steps to keep the error in the decoupling system low. To further reduce the error, we must increase the number of neurons in the spatial domain. When the time step length falls below a certain threshold, error reduction primarily results from the Euler scheme.

For relatively small time steps, a higher-order time discretization scheme could be employed to enhance accuracy. The variations in convergence order further illustrate this point. At the initial time steps, the error decreases quickly, but not solely due to the Euler scheme; the primary factor is the error in the decoupling system. As the time step increases, our convergence order approaches the theoretical order of the Euler scheme, indicating that the error asymptotically aligns with that of the Euler method. Ultimately, with only 15×15 collocation points per subdomain and 84 neurons in total, we achieved an error as low as 1.02×10^{-5}.

Additionally, we present the results for $N_{\text{steps}} = 256$, as shown in Fig. 8. As shown, the error is primarily concentrated near the interface, especially in the initial stages of time evolution. This phenomenon has been discussed earlier in the text. However, the two-dimensional example presents some unique characteristics. First, the error tends to concentrate in regions further from the boundary, which is expected due to the limited Dirichlet information in these areas. As time progresses, the error near the interface gradually spreads into other regions, particularly in the exterior domain. This occurs because, over time, the decoupling system error becomes more pronounced in regions distant from the initial time, leading to larger errors. Overall, the figure shows that our method achieves very low errors, with the L^2 error ranging from $O(10^{-5})$ to $O(10^{-6})$.



(a) $t = 0.025$ (b) $t = 0.05$

(c) $t = 0.075$ (d) $t = 0.1$

Figure 8: Exact solution and absolute error for the 2D Stefan problem.

In summary, our method exhibits excellent performance in solving two-dimensional two-phase Stefan problems. Our approach not only requires fewer parameters and offers faster training times but also achieves significantly higher accuracy.

5. Conclusion

The proposed DELM framework effectively addresses the challenges of mesh generation in traditional methods and overcomes the limitations of conventional mesh-free approaches, such as high parameter counts and lengthy training times. By leveraging the strengths of locELM and incorporating an innovative augmented variable technique, DELM not only maintains high computational efficiency but also extends its applicability from known interface problems to free boundary problems.

A key contribution of our work is the enhanced augmented variable technique, which builds upon the foundation established by Lai et al. [31, 32]. This approach significantly reduces the number of parameters required, streamlines computational processes, and expands its utility to a broader class of interface problems, including free boundary scenarios. By integrating traditional front-tracking methods within our framework, DELM achieves precise and robust interface modeling, ensuring accurate handling of complex interface geometries and dynamics.

The effectiveness and versatility of our method have been rigorously validated through comprehensive benchmarking on a wide range of interface and free boundary problems. These

validations highlight its superior accuracy and efficiency compared to existing approaches. Furthermore, the ability to handle both known and unknown interface problems within a unified framework underscores the practicality and robustness of the proposed methodology.

While our current work demonstrates strong potential in solving interface problems, future research could explore several promising directions. These include extending the framework to address more complex and highly nonlinear interface or free boundary problems, and enhancing its adaptability to dynamic interfaces. With these advancements, DELM has the potential to become a versatile and powerful tool for a wide range of applications in science and engineering.

Acknowledgements Wejun Zhao was supported by the National Key R&D Program of China (No. 2023YFA1008903-3), Natural Science Foundation of Shandong Province (No. ZR2023ZD38), National Natural Science Foundation of China (No. 12131014). Weidong Zhao was supported by the National Natural Science Foundation of China (Nos. 12371398, 12071261). Zhiwen Zhang was supported by the National Natural Science Foundation of China (Projects 92470103 and 12171406), the Hong Kong RGC grant (Projects 17307921 and 17304324), the Seed Funding Programme for Basic Research (HKU), the Outstanding Young Researcher Award of HKU (2020-21), and Seed Funding for Strategic Interdisciplinary Research Scheme 2021/22 (HKU).

Data availability Data is available on request from the authors.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethics approval and consent to participate Not applicable.

References

[1] D. A. Edwards, H. Brenner, D. T. Wasan, Interfacial transport processes and rheology, Butterworths/Heinemann, London, 1991.

[2] A. Friedman, Free boundary problems in science and technology, Notices of the AMS 47 (8) (2000) 854–861.

[3] R. Furzeland, A comparative study of numerical methods for moving boundary problems, IMA Journal of Applied Mathematics 26 (4) (1980) 411–429.

[4] S. O. Unverdi, G. Tryggvason, A front-tracking method for viscous, incompressible, multi-fluid flows, Journal of computational physics 100 (1) (1992) 25–37.

[5] D. E. Womble, A front-tracking method for multiphase free boundary problems, SIAM journal on numerical analysis 26 (2) (1989) 380–396.

[6] S. Chen, B. Merriman, S. Osher, P. Smereka, A simple level set method for solving stefan problems, Journal of Computational Physics 135 (1) (1997) 8–29.

[7] S. Osher, J. A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations, Journal of computational physics 79 (1) (1988) 12–49.

[8] F. Bai, D. Han, X. He, X. Yang, Deformation and coalescence of ferrodroplets in rosensweig model using the phase field and modified level set approaches under uniform magnetic fields, Communications in Nonlinear Science and Numerical Simulation 85 (2020) 105213.

[9] G. J. Fix, Phase field methods for free boundary problems (1982).

[10] J. Mackenzie, M. Robertson, A moving mesh method for the solution of the one-dimensional phase-field equations, Journal of Computational Physics 181 (2) (2002) 526–544.

[11] Y. Gu, X. He, D. Han, On the phase-field modeling of rapid solidification, Computational Materials Science 199 (2021) 110812.

[12] L. Chen, H. Wei, M. Wen, An interface-fitted mesh generator and virtual element methods for elliptic interface problems, Journal of Computational Physics 334 (2017) 327–348.

[13] Z. Chen, S. Dai, On the efficiency of adaptive finite element methods for elliptic problems with discontinuous coefficients, SIAM Journal on Scientific Computing 24 (2) (2002) 443–462.

[14] Z. Chen, J. Feng, An adaptive finite element algorithm with reliable and efficient error control for linear parabolic problems, Mathematics of computation 73 (247) (2004) 1167–1193.

[15] Z. Li, M.-C. Lai, The immersed interface method for the navier–stokes equations with singular forces, Journal of Computational Physics 171 (2) (2001) 822–842.

[16] R. J. LeVeque, Z. Li, Immersed interface methods for stokes flow with elastic boundaries or surface tension, SIAM Journal on Scientific Computing 18 (3) (1997) 709–735.

[17] Z. Li, K. Ito, The immersed interface method: numerical solutions of PDEs involving interfaces and irregular domains, SIAM, 2006.

[18] R. J. LeVeque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM Journal on Numerical Analysis 31 (4) (1994) 1019–1044.

[19] X. He, T. Lin, Y. Lin, et al., Immersed finite element methods for elliptic interface problems with non-homogeneous jump conditions, International Journal of numerical analysis and modeling 8 (2) (2011) 284–301.

[20] W. Feng, X. He, Y. Lin, X. Zhang, Immersed finite element method for interface problems with algebraic multigrid solver, Communications in Computational Physics 15 (4) (2014) 1045–1067.

[21] I. Babuška, U. Banerjee, K. Kergrene, Strongly stable generalized finite element method: Application to interface problems, Computer Methods in Applied Mechanics and Engineering 327 (2017) 58–92.

[22] J. Chessa, H. Wang, T. Belytschko, On the construction of blending elements for local partition of unity enriched finite elements, International Journal for Numerical Methods in Engineering 57 (7) (2003) 1015–1038.

[23] Y. Xiao, J. Xu, F. Wang, High-order extended finite element methods for solving interface problems, Computer Methods in Applied Mechanics and Engineering 364 (2020) 112964.

[24] E. Burman, S. Claus, P. Hansbo, M. G. Larson, A. Massing, Cutfem: discretizing geometry and partial differential equations, International Journal for Numerical Methods in Engineering 104 (7) (2015) 472–501.

[25] X.-D. Liu, R. P. Fedkiw, M. Kang, A boundary condition capturing method for poisson's equation on irregular domains, Journal of computational Physics 160 (1) (2000) 151–178.

[26] I. Goodfellow, Deep learning (2016).

[27] Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach, Journal of Computational Physics 400 (2020) 108963.

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

[29] S. Wang, P. Perdikaris, Deep learning of free boundary and stefan problems, Journal of Computational Physics 428 (2021) 109914.

[30] S. Wu, B. Lu, Inn: Interfaced neural networks as an accessible meshless approach for solving interface pde problems, Journal of Computational Physics 470 (2022) 111588.

[31] W.-F. Hu, T.-S. Lin, M.-C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems, Journal of Computational Physics 469 (2022) 111576.

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

[33] J. Chen, X. Chi, Z. Yang, et al., Bridging traditional and machine learning-based algorithms for solving pdes: the random feature method, J Mach Learn 1 (2022) 268–98.

[34] J. Chen, Y. Luo, et al., The random feature method for time-dependent problems, arXiv preprint arXiv:2304.06913 (2023).

[35] J. Chen, L. Tan, High-precision randomized iterative methods for the random feature method, arXiv preprint arXiv:2409.15818 (2024).

[36] X. Chi, J. Chen, Z. Yang, The random feature method for solving interface problems, Computer Methods in Applied Mechanics and Engineering 420 (2024) 116719.

[37] Y. Liang, Q. Zhang, S. Zeng, A piecewise extreme learning machine for interface problems, Mathematics and Computers in Simulation 227 (2025) 303–321.

[38] S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, Computer Methods in Applied Mechanics and Engineering 387 (2021) 114129.

[39] L. I. Rubinšteĭn, The stefan problem, Vol. 8, American Mathematical Soc., 2000.

[40] S. Howison, Complex variable methods in hele–shaw moving boundary problems, European Journal of Applied Mathematics 3 (3) (1992) 209–224.

[41] J. F. Rodrigues, J. M. Urbano, On a darcy–stefan problem arising in freezing and thawing of saturated porous media, Continuum Mechanics and Thermodynamics 11 (1999) 181–191.

[42] T. Y. Hou, Z. Li, S. Osher, H. Zhao, A hybrid method for moving interface problems with application to the hele–shaw flow, Journal of Computational Physics 134 (2) (1997) 236–252.

[43] P. Howell, Surface-tension-driven flow on a moving curved surface, Journal of engineering mathematics 45 (2003) 283–308.

[44] J. B. Keller, M. J. Miksis, Surface tension driven flows, SIAM Journal on Applied Mathematics 43 (2) (1983) 268–277.

[45] S. Popinet, An accurate adaptive solver for surface-tension-driven interfacial flows, Journal of Computational Physics 228 (16) (2009) 5838–5866.

[46] L. Bronsard, B. T. Wetton, A numerical method for tracking curve networks moving with curvature motion, Journal of Computational Physics 120 (1) (1995) 66–87.

[47] S. J. Ruuth, B. T. Wetton, A simple scheme for volume-preserving motion by mean curvature, Journal of Scientific Computing 19 (2003) 373–384.

[48] M. Raissi, P. Perdikaris, 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 (2019) 686–707.

[49] Z. Shen, H. Yang, S. Zhang, Optimal approximation rate of relu networks in terms of width and depth, Journal de Mathématiques Pures et Appliquées 157 (2022) 101–135.

[50] K. Tang, X. Wan, C. Yang, Das-pinns: A deep adaptive sampling method for solving high-dimensional partial differential equations, Journal of Computational Physics 476 (2023) 111868.

[51] K. Tang, J. Zhai, X. Wan, C. Yang, Adversarial adaptive sampling: Unify pinn and optimal transport for the approximation of pdes, arXiv preprint arXiv:2305.18702 (2023).

[52] X. Wan, T. Zhou, Y. Zhou, Adaptive importance sampling for deep ritz, Communications on Applied Mathematics and Computation (2024) 1–25.

[53] A. V. Borovskikh, The two-dimensional eikonal equation, Siberian Mathematical Journal 47 (2006) 813–834.

[54] H. Zhao, A fast sweeping method for eikonal equations, Mathematics of computation 74 (250) (2005) 603–627.

[55] G. Lamé, B. Clapeyron, Mémoire sur la solidification par refroidissement d'un globe liquide, in: Annales Chimie Physique, Vol. 47, 1831, pp. 250–256.

[56] J. Stefan, Über einige probleme der theorie der wärmeleitung, Sitzungber., Wien, Akad. Mat. Natur 98 (1889) 473–484.

[57] D. M. Stefanescu, R. Ruxanda, Fundamentals of solidification (2004).

[58] H. S. Carslaw, J. C. Jaeger, H. Feshbach, Conduction of heat in solids, Physics Today 15 (11) (1962) 74–76.

[59] H. Blatter, Velocity and stress fields in grounded glaciers: a simple algorithm for including deviatoric stress gradients, Journal of Glaciology 41 (138) (1995) 333–344.

[60] J. W. Peterson, Arc length parameterization of spline curves, Journal of Compu ter Aided Design (2006).

[61] P. Ram, K. Sinha, Revisiting kd-tree for nearest neighbor search, in: Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining, 2019, pp. 1378–1388.

[62] J. C. Butcher, A history of runge-kutta methods, Applied numerical mathematics 20 (3) (1996) 247–260.

[63] J. C. Butcher, Numerical methods for ordinary differential equations, John Wiley & Sons, 2016.

[64] D. Picard, Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision, arXiv preprint arXiv:2109.08203 (2021).