mathematics

Article

Estimating Failure Probability with Neural Operator Hybrid Approach

Mujing Li [1], Yani Feng [1] and Guanjie Wang [2,*]

[1] School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China; limj@shanghaitech.edu.cn (M.L.); fengyn@shanghaitech.edu.cn (Y.F.)
[2] School of Statistics and Mathematics, Shanghai Lixin University of Accounting and Finance, Shanghai 201209, China
* Correspondence: guanjie@lixin.edu.cn

Abstract: Evaluating failure probability for complex engineering systems is a computationally intensive task. While the Monte Carlo method is easy to implement, it converges slowly and, hence, requires numerous repeated simulations of a complex system to generate sufficient samples. To improve the efficiency, methods based on surrogate models are proposed to approximate the limit state function. In this work, we reframe the approximation of the limit state function as an operator learning problem and utilize the DeepONet framework with a hybrid approach to estimate the failure probability. The numerical results show that our proposed method outperforms the prior neural hybrid method.

Keywords: failure probability; neural operator learning; DeepONet; approximation theory

MSC: 65C20; 65D15; 68T07

Citation: Li, M.; Feng, Y.; Wang, G. Estimating Failure Probability with Neural Operator Hybrid Approach. *Mathematics* **2023**, *11*, 2762. https://doi.org/10.3390/math11122762

Academic Editor: Theodore E. Simos

Received: 21 May 2023
Revised: 15 June 2023
Accepted: 16 June 2023
Published: 18 June 2023

1. Introduction

In practice, evaluating failure probability for systems that inherently contain uncertainty is a fundamental problem encountered in various fields, such as structural safety, risk management, reliability-based optimization, etc. Uncertainties in such systems are abstracted in terms of the failure mode, and failure probability estimation is essentially a problem of evaluating multivariate integrals in domains defined by certain failure modes. While the mathematical formulation of the problem is well defined, evaluating such integrals remains a challenging task in practice.

The most straightforward approach to evaluating the failure probability is to use the Monte Carlo sampling (MCS) method [1,2]. However, due to its slow convergence, MCS requires numerous samples, resulting in a heavy computational burden. This computational burden becomes even more pronounced when complex stochastic PDEs represent the failure modes, since MCS necessitates repeatedly solving the model to estimate the failure probability.

To address this issue, various approaches have been developed, including the first-order reliability method (FORM) [3], second-order reliability method (SORM) [4], and response surface method (RSM) [5,6]. These methods replace the limit state function with a surrogate model that is easy to evaluate, thereby greatly reducing the simulation time. Following this idea, a hybrid method was proposed by [7,8] to estimate the probability based on the surrogate model while re-evaluating samples in a given suspicious region. The design of the hybrid method significantly reduces the time complexity while ensuring accuracy.

There are various methodologies for constructing surrogate models, such as the stochastic Galerkin method [9,10], the reduced basis method [11,12], and deep learning. Deep learning has rapidly developed in recent decades, particularly in scientific and engineering applications. Physics-informed neural networks (PINNs), which build upon the widely known universal approximation capabilities of continuous functions for neural networks (NNs) [13,14], were introduced in [15] and have demonstrated their efficiency in numerous studies [16–18]. By utilizing established deep learning and machine learning techniques, NN models can be employed as surrogate models to approximate the limit state function, outperforming traditional surrogate models in certain problems, such as those of high-dimensional systems [19–21].

As a significant area within the domain of deep learning, operator learning has emerged in recent years. The underlying principle of operator learning resides in the observation that nonlinear operators can be effectively approximated by employing single-layer neural networks [22,23] (Theorem 1). Operator learning aims to map infinite-dimensional functions to infinite-dimensional functions. Since it is more expressive and can break the curse of dimensionality in input space [24], operator learning has gained much attention in recent years [24–26]. Among the operator learning techniques, the DeepONet introduced in [24,26] has been demonstrated to be effective in numerous applications, including [27–29].

In this work, we present a novel approach to failure probability estimation by reframing the approximation problem of the limit state function as an operator learning problem and subsequently adapting the DeepONet framework to address it. The operator learning formulation provides a more effective and generalized approach to constructing a surrogate model for the limit state function, resulting in enhanced precision and reduced simulation numbers. To further ensure the precision, we employed a hybrid method [7] for estimating the failure probability. Our proposed neural operator hybrid (NOH) approach significantly reduces the time complexity while maintaining high accuracy compared to earlier neural hybrid and Monte Carlo simulation approaches. We posit that the efficiency of our approach in estimating failure probability demonstrates the potential of operator learning in various tasks.

This paper is structured as follows: In Section 2, we present the problem setting and introduce a hybrid method for evaluating the failure probability. The neural operator learning and proposed algorithm are then fully described in Section 3. To demonstrate the effectiveness of our approach, we describe numerical experiments in Section 4 that cover a variety of scenarios, including ODEs, PDEs, and multivariate models. Finally, we offer concluding remarks and observations in Section 5.

2. Preliminaries

This section will provide an overview of the mathematical framework for failure probability and introduce a hybrid method for solving this problem.

2.1. Problem Setting

Let $Z = (Z_1, Z_2, \dots, Z_{n_z})$ be an n_z-dimensional random vector with the distribution function $F_Z(z) = \mathrm{Prob}(Z \leq z)$. The image of Z, i.e., the set of all possible values that Z can take, is denoted by Ω. It is our interest to evaluate the failure probability P_f defined by Equation (1):

$$P_f = \mathrm{Prob}\left(Z \in \Omega_f\right) = \int_{\Omega_f} \mathrm{d}F_Z(z) = \int \chi_{\Omega_f}(z)\mathrm{d}F_Z(z) = \mathbb{E}[\chi_{\Omega_f}(z)], \tag{1}$$

where the characteristic function $\chi_{\Omega_f}(z)$ is defined as:

$$\chi_{\Omega_f}(z) = \begin{cases} 1 & \text{if} \quad z \in \Omega_f, \\ 0 & \text{if} \quad z \notin \Omega_f, \end{cases} \tag{2}$$

and the failure domain Ω_f, where failure occurs, is defined as:

$$\Omega_f = \{Z : g(Z) < 0\}. \tag{3}$$

Here, $g(Z)$ is a scalar limit state function—also known as a performance function—that characterizes the failure domain. It should be emphasized that, in many real-world systems, $g(Z)$ does not have an analytical expression and is instead characterized by a complex system that requires expensive simulations to evaluate. Consequently, the evaluation of $g(Z)$ can be computationally expensive, leading to significant time complexity.

2.2. Hybrid Method

The most straightforward approach to estimating failure probability is the Monte Carlo sampling (MCS) method [1,2], which is given by:

$$P_f^{mc} = \frac{1}{M}\sum_{i=1}^{M} \chi_{\{g(z)<0\}}\left(z^{(i)}\right), \tag{4}$$

where $\{z^{(i)}\}_{i=1}^{M}$ is a set of sample points for the random vector Z. The characteristic function $\chi_{\{g(z)<0\}}(z^{(i)})$ takes a value of 1 if the limit state function $g(Z)$ evaluated at $z^{(i)}$ is less than zero and 0 otherwise. The failure probability P_f^{mc} is estimated as the average of the characteristic function over the M sample points.

However, evaluating the limit state function $g(Z)$ at numerous sample points can be a computationally intensive task, especially when dealing with complex stochastic systems, resulting in significant simulation time complexity. To address this issue, a surrogate model can be used to approximate the limit state function $g(Z)$ and avoid the need for direct evaluation at each sample point. Specifically, a surrogate model of $g(Z)$ is denoted by $\hat{g}(Z)$, which can be rapidly evaluated. The failure probability can then be estimated as:

$$\hat{P}_f^{mc} = \frac{1}{M}\sum_{i=1}^{M} \chi_{\{\hat{g}(z)<0\}}\left(z^{(i)}\right). \tag{5}$$

While surrogate models can significantly reduce computational costs in Monte Carlo methods, relying solely on them for estimating the failure probability may result in poor precision or even failure. To address this issue, a hybrid approach that combines the surrogate models $\hat{g}$ and the limit state function g was proposed in [7,8]. In the following, we give a brief review of the hybrid method.

Suppose that $(-\gamma, \gamma)$ is a suspicious region, where γ is a non-negative real number. In this case, we can approximate the failure domain Ω_f with $\widetilde{\Omega}_f$ as follows:

$$\widetilde{\Omega}_f = \{\hat{g}(Z) < -\gamma\} \cup \{\{|\hat{g}(Z)| \leq \gamma\} \cap \{g(Z) < 0\}\}, \tag{6}$$

where g is the limit state function, and $\hat{g}$ represents the surrogate model of g. Enhanced with the hybrid method, the failure probability can be estimable by MCS:

$$\begin{aligned} P_f^h &= \frac{1}{M}\sum_{i=1}^{M} \chi_{\hat{\Omega}_f}\left(z^{(i)}\right) \\ &= \frac{1}{M}\sum_{i=1}^{M}\left[\chi_{\{\hat{g}<-\gamma\}}\left(z^{(i)}\right) + \chi_{\{|\hat{g}|\leq\gamma\}}\left(z^{(i)}\right)\cdot\chi_{\{g<0\}}\left(z^{(i)}\right)\right]. \end{aligned} \tag{7}$$

The hybrid method can be considered as an approach for estimating P_f by using a surrogate $\hat{g}$, followed by a re-evaluation of the samples within the suspicious domain. While increasing the value of γ leads to higher time complexity, it also results in more accurate estimation. In Ref. [7], it was proved that for any surrogate $\hat{g}(Z)$ and for all $\epsilon > 0$, there exists a critical value $\gamma_N > 0$ such that for all $\gamma > \gamma_N$, the difference between the estimated P_f^h and the truth P_f is less than ϵ, i.e.,

$$\left|P_f - P_f^h\right| < \epsilon. \tag{8}$$

To be more precise,

$$\gamma_N = \frac{1}{\epsilon^{1/p}} \|g(Z) - \hat{g}(Z)\|_{L^p_\Omega}, \tag{9}$$

where the approximation is measured in the L^p-norm with $p \geq 1$.

$$\|g(Z) - \hat{g}(Z)\|_{L^p_\Omega} = \left(\int_\Omega |g(z) - \hat{g}(z)|^p \mathrm{d}F_Z(z) \right)^{1/p}. \tag{10}$$

Selecting an appropriate value of γ that balances accuracy and computational efficiency can be a challenging task. To address this challenge, an iterative algorithm, as demonstrated in Algorithm 1, is commonly employed in practice instead of directly selecting γ. In Algorithm 1, the surrogate $\hat{g}$ samples are gradually replaced with g samples in the iteration procedure until either the stopping criterion is reached or the iteration step reaches $\lceil M/\delta M \rceil$, which is equivalent to expanding the suspicious region at each iteration. When k reaches $\lceil M/\delta M \rceil$, the iterative hybrid algorithm degenerates to the Monte Carlo method (4), indicating that the convergence is achieved as $P_f^{(k)} \to P_f^{mc}, \quad k \to \lceil M/\delta M \rceil$. It is obvious that the time complexity of the iterative hybrid algorithm is heavily influenced by the accuracy of the surrogate model used in Algorithm 1.

Algorithm 1 Iterative Hybrid Method [7].

```
Input: surrogate model $\hat{g}$, $S = \{z^{(i)}\}_{i=1}^M$ samples from random variable $Z$, tolerance $\epsilon$, and
    sample size in each iteration $\delta M$.
1:  Initialization: $k = 0$.
2:  Compute $P_f^{(0)} = \frac{1}{M} \sum_{i=1}^M \chi_{\{\hat{g}(z)<0\}} \left(z^{(i)}\right)$.
3:  Sort $\left\{ \left| \hat{g}\left(z^{(i)}\right) \right| \right\}_{i=1}^M$ in ascending order; sort the correspond sample $S$ accordingly.
4:  for $k$ from 1 to $\lceil M/\delta M \rceil$ do
5:      $\delta S^k = \left\{ z^{(j)} \right\}_{j=(k-1)\delta M+1}^{k\delta M}$.
6:      $\delta P = \frac{1}{M} \sum_{z^{(j)} \in \delta S^k} \left[ -\chi_{\{\hat{g}<0\}} \left(z^{(j)}\right) + \chi_{\{g<0\}} \left(z^{(j)}\right) \right]$.
7:      $P_f^{(k)} = P_f^{(k-1)} + \delta P$.
8:      if $|\delta P| \leq \epsilon$ for several times then
9:          break
10:     end if
11: end for
Output: $P_f^{(k)}$
```

3. Neural Operator Hybrid Algorithm

In Section 2, we described the failure probability problem and the hybrid algorithm for solving it. As we have mentioned, the accuracy of the surrogate model greatly affects the performance of the iterative hybrid algorithm. In this section, we introduce neural operator learning and present the neural operator hybrid (NOH) algorithm, which reframes the approximation problem of the limit state function as an operator learning problem. Unlike prior studies that used neural networks as surrogate models for mappings (as in [19–21,30,31]), our algorithm constructs a surrogate model by using operator learning techniques. The benefits of our method are mainly in two aspects: First, it increases the generalization of the surrogate model. Second, it increases the precision of the surrogate model with more information involved, which results in a more effective and generalized approach to estimating failure probability.

3.1. Neural Operator Learning

Neural operator learning aims to accurately represent linear and nonlinear operators that map input functions into output functions. More specifically, let U be a vector space of functions on set K_1, and let V be a vector space of functions on set K_2; G is an operator map from U to V, i.e.,

$$G : u \mapsto G(u) \in V, \text{ for } u \in U, \tag{11}$$

where u is a function defined on the domain K_1, i.e.,

$$u : x \mapsto u(x) \in \mathbb{R}, \text{ for } x \in K_1, \tag{12}$$

and $G(u)$ is a function defined on the domain K_2, i.e.,

$$G(u) : y \mapsto G(u)(y) \in \mathbb{R}, \text{ for } y \in K_2. \tag{13}$$

In the context of this paper, U is referred to as the input function space, and V is referred to as the output function space. It is of interest to design neural networks that can approximate the mapping of the operator G from the input function space to the output function space.

In this work, we employ the DeepONet framework [26], an ascendingoperator learning approach based on the following theorem, to construct the surrogate of the operator G.

Theorem 1 (Universal Approximation Theorem for the Operator [23])**.** *Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X, K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, U is a compact set in $C(K_1)$, and G is a nonlinear continuous operator that maps U into $C(K_2)$. Then, for any $\epsilon > 0$, there are positive integers n, p, m and constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}, w_k \in \mathbb{R}^d, x_j \in K_1, i = 1, \ldots, n, k = 1, \ldots, p, j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon. \tag{14}$$

holds for all $u \in U$ and $y \in K_2$.

In DeepONet, the operator G is approximated by taking the inner product of two components, which can be expressed as follows:

$$G(u)(y) \approx \mathcal{G}(u)(y) := \sum_{k=1}^{p} \underbrace{b_k(u)}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}}, \tag{15}$$

where $b_k(u)$ is the output of the **trunk network** for a given input function u in U, and $t_k(y)$ is the output of the **branch network** for a given y in K_2. Figure 1 illustrates this architecture.

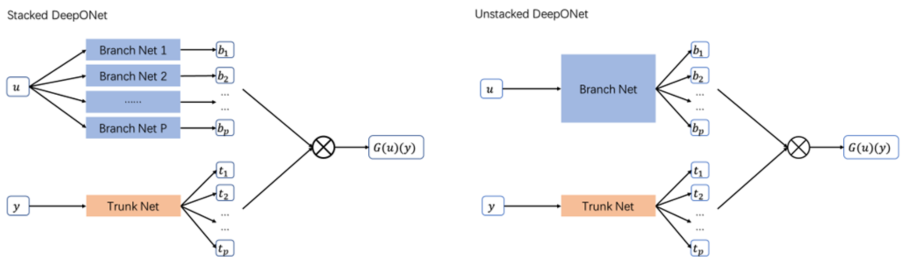


Figure 1. Illustrations of stacked and unstacked architectures of DeepONets [26]. DeepONets with unstacked structures replace branch networks with a single multi-layer neural network.

In practical applications, the input function u is vectorized as $[u(x_1), \ldots, u(x_m)]^T$, where the sample points $[x_1, x_2 \ldots x_m]^T$ are referred to as **sensors**. The input of the trunk network takes P specific values $y_1^{(i)}, \dots, y_P^{(i)}$ in K_2 for each $G(u^{(i)})$. The neural network can be trained by using the training data:

$$\mathcal{T} = \left\{ \left(u^{(1)}, G(u^{(1)})\right), \left(u^{(2)}, G(u^{(2)})\right), \cdots, \left(u^{(N)}, G(u^{(N)})\right) \right\}, \tag{16}$$

and by minimizing the following loss function:

$$\mathcal{L}(\Theta) = \frac{1}{NP} \sum_{l=1}^{P} \sum_{i=1}^{N} \| G_\Theta(u^{(i)})(y_l^{(i)}) - G(u^{(i)})(y_l^{(i)}) \|_2, \tag{17}$$

where the neural network G_Θ with parameter Θ approximates the operator G.

3.2. Neural Operator Hybrid Algorithm

Neural operator learning involves functions as both inputs and outputs, requiring us to reframe our problem accordingly. Let U be the collection of all functions defined on $[0,1]$ such that

$$u(x) = \overline{Z} - k\overline{\sigma} + 2k\overline{\sigma}x, \text{ for } x \in [0,1], \tag{18}$$

where k is a selective parameter, and $\overline{Z}$ and $\overline{\sigma}$ are defined as:

$$\overline{Z} = \frac{Z_1 + \ldots + Z_{n_z}}{n_z}, \ \overline{\sigma} = \frac{\sigma_1 + \ldots + \sigma_{n_z}}{n_z}. \tag{19}$$

Here, $\sigma_1, \dots, \sigma_{n_z}$ are the variances of random variables $Z_1, \dots, Z_{n_z}$, respectively. We can then define an operator G fromm U to V as follows:

$$G : u \mapsto G(u) \in V, \text{ for } u \in U, \tag{20}$$

where $V = \text{span}\{g(x)\}$, and $G(u)$ is defined by

$$G(u) : y \mapsto g(y) \in \mathbb{R}, \text{ for } y \in \mathbb{R}^{n_z}. \tag{21}$$

Here, g is the limit state function discussed in Section 2.1. Then, designing a surrogate model for G is a standard operator learning problem.

The inspiration for this reframing is the establishment of a relationship between the input function $u(x)$ and the random variable Z. It is important to note that the prior distribution of $u(x)$ is entirely known, which enables us to generate training data by using the following process: Firstly, we randomly sample $z^{(i)}$ according to the random distribution Z, where $i = 1, \dots, N$. Next, we define $u^{(i)}(x)$ by using the following equation:

$$u^{(i)}(x) = \overline{z}^{(i)} - k\overline{\sigma} + 2k\overline{\sigma}x, \ x \in [0,1], \tag{22}$$

where

$$\overline{z}^{(i)} = \frac{z_1^{(i)} + \ldots + z_{n_z}^{(i)}}{n_z}. \tag{23}$$

In practical implementation, we vectorize the input function $u^{(i)}(x)$ on a uniform grid of the interval $[0,1]$. Specifically, the vectorized input function is given by $[u^{(i)}(x_1), \dots, u^{(i)}(x_m)]$, where x_j denotes the j-th sensor and is defined by

$$x_j = \frac{j-1}{m-1}, \ j = 1, \dots, m. \tag{24}$$

Suppose that there are P observations $y_l^{(i)} \in \mathbb{R}^{n_z}$, $l = 1, \ldots, P$ for $G(u^{(i)})$; then, by Equation (21), we have $G(u^{(i)})(y_l^{(i)}) = g(y_l^{(i)})$. Once the dataset is generated, the model is trained by minimizing the following loss function:

$$\mathcal{L}(\Theta) = \frac{1}{NP} \sum_{l=1}^{P} \sum_{i=1}^{N} \| G_\Theta(u^{(i)})(y_l^{(i)}) - g(y_l^{(i)}) \|_2, \tag{25}$$

where G_Θ is a neural network with parameters Θ that approximates the operator G.

After constructing the surrogate for G by using the neural network G_Θ, we can integrate it into a hybrid algorithm to estimate the failure probability. This whole process is called the neural operator hybrid (NOH) method, and it is shown in Figure 2.

As discussed in Section 2.2, the convergence of Algorithm 1 depends on the norm measurement of the difference between the surrogate and the limit state function. Therefore, it is crucial to have an accurate approximation $\hat{g}(Z)$ for reliable estimation. The surrogate model constructed for the reformulated operator learning problem using DeepONet may achieve higher accuracy than that of the surrogate model constructed by using neural networks that do not incorporate information from random variables, as the former model utilizes additional information from random distribution functions. Moreover, DeepONet exhibits less generalization error than that of simple neural networks [26].

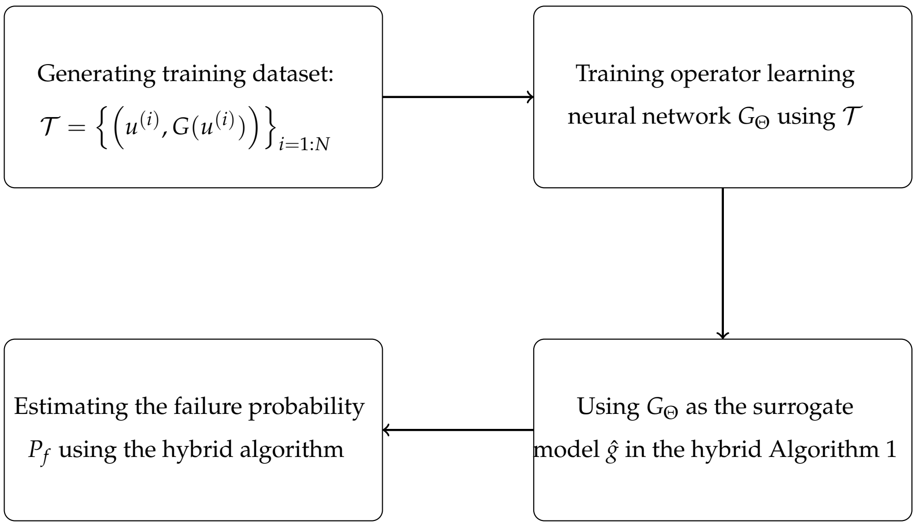

Generating training dataset:
$\mathcal{T} = \left\{ \left(u^{(i)}, G(u^{(i)}) \right) \right\}_{i=1:N}$
Training operator learning neural network G_Θ using $\mathcal{T}$
Using G_Θ as the surrogate model $\hat{g}$ in the hybrid Algorithm 1
Estimating the failure probability P_f using the hybrid algorithm

Figure 2. A descriptive flowchart for the NOH method.

4. Numerical Experiments

In this section, we present three numerical examples to demonstrate the efficiency and effectiveness of the proposed neural operator hybrid (NOH) method. Furthermore, we compare the NOH method with the neural hybrid (NH) method. For the purpose of clarity in presentation, we refer to the surrogate model constructed by using fully connected neural networks for g in the NH method as **the neural surrogate** and the surrogate constructed by using DeepONet for g in the NOH method as **the neural operator surrogate**. Both surrogate models are designed to approximate limit state function g, and their main difference is the structure of neural networks utilized.

For the NOH method, we use the simplest unstacked DeepONet to construct the neural operator surrogate for the operator G, with the branch and trunk networks implemented as fully connected neural networks (FNNs). The trunk network is employed with a depth of 2 and a width of 40 FNNs, while the branch network has a depth of 2 and a width of 40 FNNs. To facilitate a comparative analysis with the NH method, we built a neural surrogate for g by using a simple FNN with a parameter size comparable to that of the NOH method. Specifically, in the NH method, the FNN utilized for the neural surrogate has a depth of 3, and its width is adjusted to achieve a similar number of parameters to that in the DeepONet. Both models were optimized by using the Adam optimizer [32] with a learning rate of 0.001 on identical datasets.

The code was run by using PyTorch [33] and MATLAB 2019b on a workstation with an Nvidia GTX 1080Ti graphics card and an Intel Core i5-7500 processor with 16 GB of RAM. It is noteworthy that the evaluation of time complexity is based on the performance function (PF) calls N_{call}, which refers to the number of system simulations that need to be executed, rather than the running time of the programs, as program running speeds may vary significantly across different programming languages and platforms. The PF calls consist of the evaluation of the hybrid algorithm in line 6 of Algorithm 1 and simulations for generating training data in Equation (16). We do not evaluate the computational time required for the neural surrogate or the neural operator surrogate, as a model trained by using batch techniques can evaluate 10^5 samples in less than a second.

4.1. Ordinary Differential Equation

In this test problem, we consider a random ordinary differential equation (ODE) proposed in [7]. The ODE is given by:

$$\frac{\mathrm{d}s}{\mathrm{d}t} = -Zs, \quad s(0) = s_0, \tag{26}$$

where $s_0 = 1$, and $Z \sim \mathcal{N}(\mu, \sigma^2)$ is a Gaussian random variable with a mean of $\mu = -2$ and standard deviation of $\sigma = 1$. The limit state function is defined as $g(Z) = g(s(t,Z)) = s(t,Z) - s_d$, where $s_d = 0.5$ and $t = 1$. The exact failure probability $P_f = 0.003539$ is regarded as the reference solution, which can be computed by using the analytic solution $s(t,Z) = s_0\mathrm{e}^{-Zt}$.

To demonstrate the efficiency and effectiveness of the proposed NOH method, we compare it with a Monte Carlo simulation (MCS) and the NH method. We used DeepONet to train the neural operator surrogate in the NOH method and set the parameter k in Equation (22) to 4, the number of input functions for training N to 500, P to 500, and the number of sensors m to 100. In the NH method, we used the FNN as the neural surrogate.

Both surrogates in the NH and NOH methods were trained with identical datasets, epochs, and optimizers. Additionally, in the MCS, 10^6 samples were generated to estimate the failure probability.

Table 1 presents the performance of the MCS, the NOH method, and the NH method. As shown in the table, the NOH method outperformed MCS by achieving the same level of estimation precision with only approximately $O(N_{call}/1000)$ or 0.23% of the PF calls required by the MCS. The NH method failed to estimate the failure probability, as all of the outputs of the neural surrogate were greater than 0. In this special case, the hybrid iterative procedure always terminated too early, while the estimated failure probability remained at 0.

Table 1. Comparison of a Monte Carlo simulation (MCS), the neural hybrid (NH) method, and the neural operator hybrid (NOH) method. In the hybrid algorithm, we set δM to 25 and ϵ to 0, and we terminated the iterative procedure when $\delta P \leq \epsilon$ five times. The relative error is denoted by ϵ_e.

Method	P_f^h	N_{call}	ϵ_e
MCS	3.578×10^{-3}	10^6	0.11%
NOH	3.578×10^{-3}	500 (Training) + 1750 (Evaluating)	0.11%
NH	-	-	-

Figure 3 illustrates the convergence of the NOH method and the NH method. In order to compare the two methods, the iterative procedure in the hybrid algorithm was not terminated until the limit state function g was recomputed for at least 10^5 samples. The figure shows that the estimate of the failure probability by the NH method remained at 0 until around 100 iterations, and it converged after approximately 7000 iterations. In contrast, the NOH method converged after only 70 iterations, demonstrating its superior efficiency compared to that of the NH method.

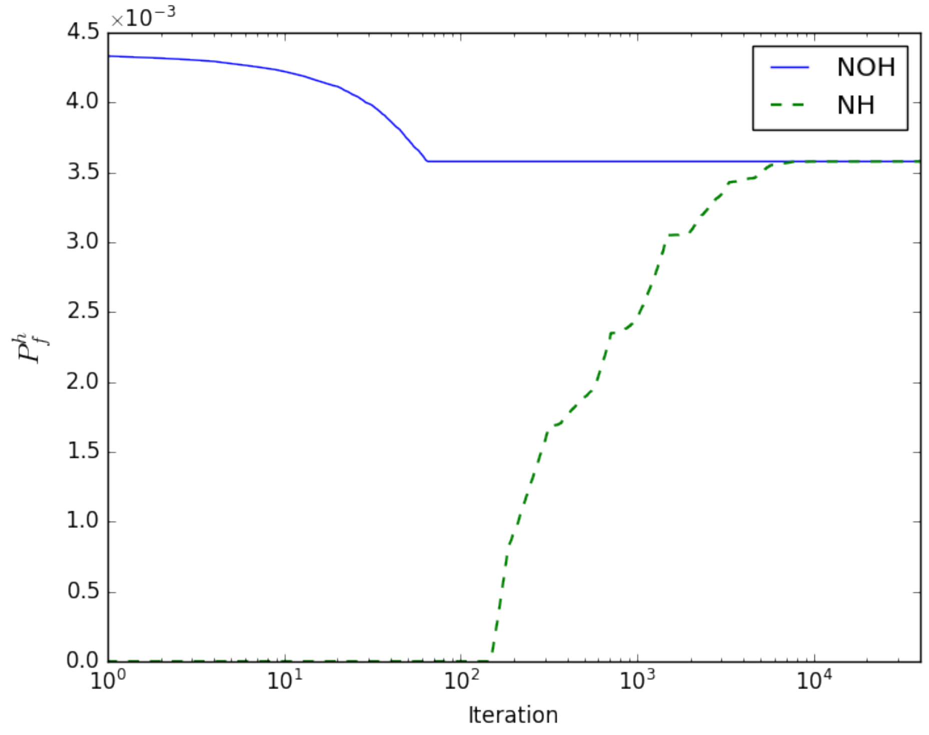


Figure 3. Convergence of the NOH method and the NH method. In the hybrid algorithm, we set δM to 25, and the iterative procedure was not terminated until the limit state function g was recomputed for 4×10^4 iterations.

In Figure 4, we compare the performance of the neural surrogate and the neural operator surrogate in predicting the limit state function g. We observed that all of the outputs of the neural surrogate were greater than 0, which led to the failure of the NH method in estimating the failure probability. It is evident that the neural operator surrogate outperformed the neural surrogate in predicting the limit state function g.

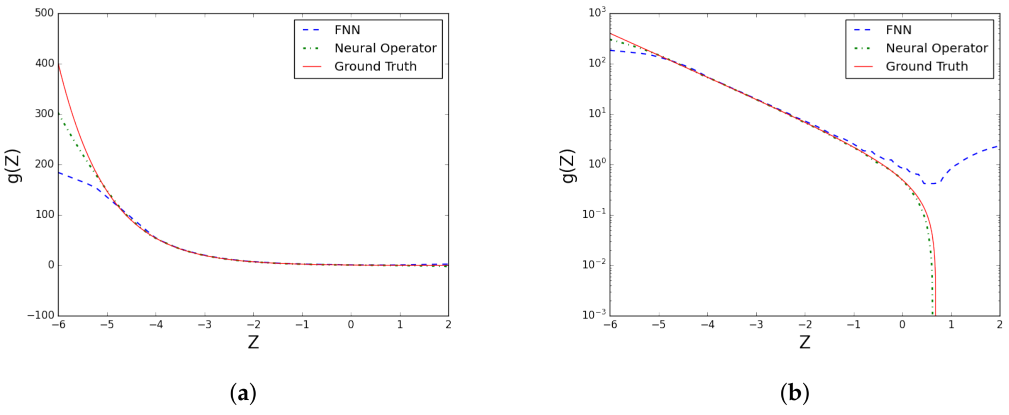


Figure 4. Comparison of the neural surrogate and the neural operator surrogate. Both models are compared to the ground truth $g(Z)$ from the analytic solution. The right figure is the left figure on the log scale. (**a**) Comparison of different surrogates. (**b**) Comparison on the log scale.

4.2. Multivariate Benchmark

Next, we consider a high-dimensional multivariate benchmark problem (dimensionality: $n = 50$) in the field of structural safety in [19,34]:

$$g(Z) = \beta n^{\frac{1}{2}} - \sum_{i=1}^{n} Z_i, \tag{27}$$

where $\beta = 3.5$ and each random variable $Z_i \sim \mathcal{N}(0,1)$, $i = 1, \ldots, n$. $g(Z)$ is the limit state function. In this test problem, the reference failure probability is $P_f^{mc} = 2.218 \times 10^{-4}$, which was obtained by using MCS with 5×10^6 samples.

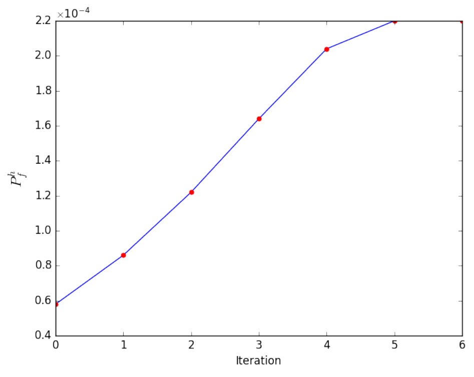
The proposed NOH method is compared with the NH method in terms of accuracy and efficiency. For the NOH method, we set the parameter k in Equation (22) to 4, the number of input functions N to 1000, P to 1000, and the number of sensors m to 100. In comparison, a naive neural surrogate employing an FNN with a similar number of parameters was also constructed and trained under conditions identical to those for the NH method.

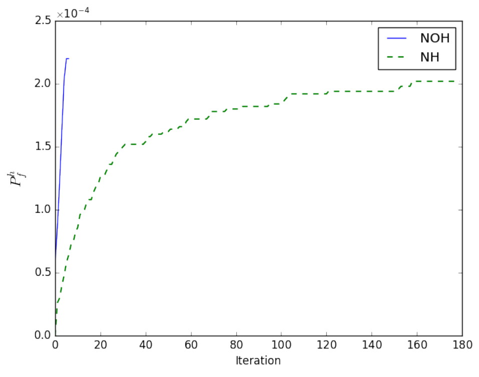
The performance of the MCS, the NOH method, and the NH method are illustrated in Table 2. Both the NOH method and the NH method demonstrated a substantial reduction in the number of samples required—approximately 0.1% of the computational cost of MCS. Notably, the NOH method outperformed the NH method by evaluating only 3% of the N_{call} while achieving a superior relative error 0.81% compared to the NH method's relative error of 8.92%. This indicated that the NOH method achieved higher accuracy with significantly fewer samples, making it a more efficient and effective approach for the given task.

Table 2. A performance evaluation was conducted to compare the MCS, the NH, and the NOH approaches. The NOH method was found to outperform the other methods, and it only evaluated 3% of N_{call} compared to NH method. The NOH method also achieved the best relative error of 0.81% while requiring the lowest number of samples with respect to the other methods.

Method	P_f^h	N_{call}	ϵ_e
MCS	2.22×10^{-4}	5×10^6	-
NOH	2.20×10^{-4}	1000 (Training) + 150 (Evaluating)	0.81%
NH	2.02×10^{-4}	1000 (Training) + 4175 (Evaluating)	8.92 %

In Figure 5, the convergence of the NOH method is depicted and compared with that of the NH method. Figure 5a demonstrates that the NOH method achieved an estimation of $P_f^h = 2.2 \times 10^{-4}$ with a relative error of $\epsilon_e = 0.81\%$ in less than six iterations. Figure 5b provides a comparison of convergent behaviors between the NH and NOH methods, clearly demonstrating that the NOH method converged in significantly fewer iterations. As a consequence, the NOH method required only 3% of the total evaluations N_{call}. These findings strongly suggest that neural operator surrogates offer improved precision and ease of training when compared to FNNs. The reduced iteration times and lower number of evaluations highlight the superior efficiency and accuracy of neural operator surrogates in approximating complex functions or operators.





(**a**) (**b**)

Figure 5. Convergence of the NOH method and the NH method. Panel (**a**) demonstrates the convergence of the NOH method, where NOH terminates in six iterations, while in (**b**), a comparison of the convergence rates between the NOH and the NH methods is conducted. The NH method requires significantly more iterations than the NOH method to converge because its neural surrogate lacks precision. For both methods, we set $\delta M = 25$ in each iteration. For NOH, ϵ was set to 0, and the iterative procedure was terminated when $\delta P \leq \epsilon$ for the first time. (**a**) Convergence of the NOH method. (**b**) Comparison between the NOH and NH methods.

Figure 6 depicts the superior accuracy of the neural operator surrogates. It compares the neural surrogate and the neural operator surrogate approximations from 50 randomly selected samples. As illustrated in the figure, the predictions of the limit function g by the neural operator surrogate are more accurate than those of the neural surrogate, as the former closely approximated the ground truth.

The experimental results provide evidence of the precision exhibited by the neural operator surrogate. The combination of reduced iteration times and a lower number of evaluations further accentuates the efficiency and accuracy of neural operator surrogates in approximating complex functions or operators. These findings emphasize that the NOH method achieved higher levels of accuracy while utilizing significantly fewer samples.

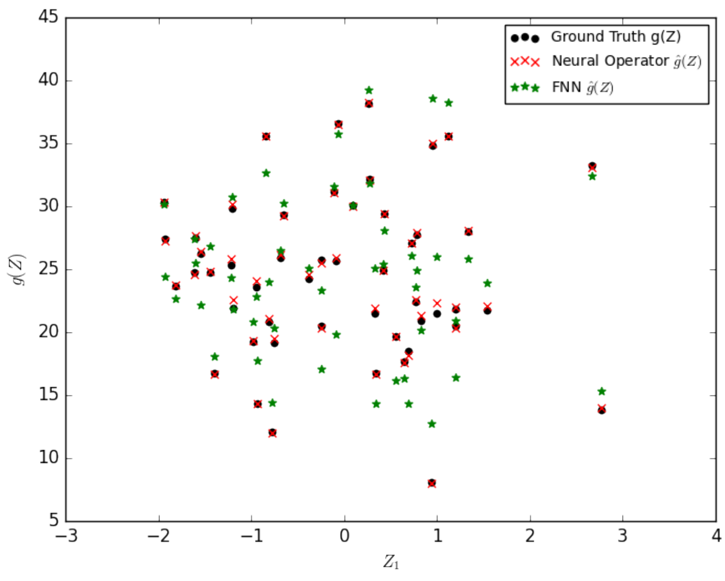


Figure 6. Comparison of the neural surrogate and the neural operator surrogate by using 50 randomly selected samples.

4.3. Helmholtz Equation

We consider the Helmholtz equation on a disk with a square hole [19]. The equation is given by:

$$-\Delta u - \kappa^2 u = 0, \tag{28}$$

where coefficient κ is a Gaussian random variable with a mean of $\mu = 60$ and variance of $\sigma = 1$, i.e, $\kappa \sim \mathcal{N}(60, 1)$. The system was set with a homogeneous term, and Dirichlet boundary conditions ($u = 0$) were applied on the edges of the square hole, while generalized Neumann conditions ($\vec{\xi} \cdot \nabla r - i\kappa r = 0$, and $\vec{\xi}$ is the radial distance from the object) were applied on the edge of the disk. The system was numerically solved by using the MATLAB PDE solver to obtain an accurate solution. A snapshot of the solution of Helmholtz is shown in Figure 7. A point sensor was placed at $x_p = [0.7264; 0.4912]$, and the failure probability was defined as $\text{Prob}(u(x_p, \kappa) > 1.00)$. The reference solution was $P_f^{mc} = 2.70 \times 10^{-4}$, which was obtained with a Monte Carlo simulation with 10^5 samples.

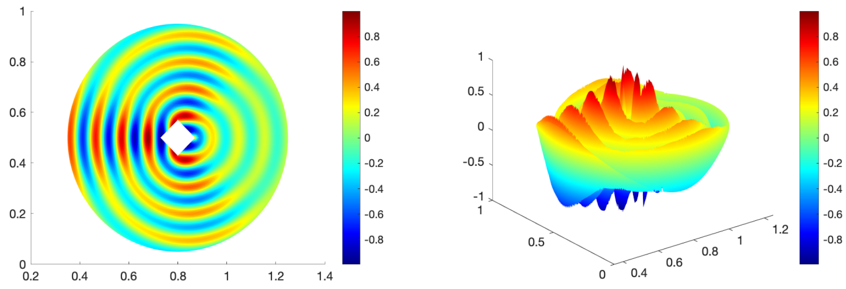


Figure 7. A snapshot of the solution of the Helmholtz equation.

With a setup similar to that in the previous experiment, we used DeepONet to train the neural operator surrogate in the NOH method and set the parameter k in Equation (22) to 4, the number of input functions N to 1000, P to 1000, and the number of sensors m to 100. We used a fully connected neural network (FNN) as the neural surrogate in the NH method and trained both models under identical conditions.

In Table 3, we present a performance analysis of the Monte Carlo simulation (MCS), the NOH method, and the NH method. Similarly to the previous experiments, the results indicate that the NOH method required fewer N_{call}—about 11%—than the NH method did while providing a more accurate estimation with 3.70% relative error versus 11.11%.

Table 3. A performance evaluation was conducted to compare the NH and the NOH approaches. The NOH method was found to outperform the NH method in both efficiency and accuracy. Specifically, the NOH method achieved a relative error of just 3.70% while requiring fewer N_{call} than the NH method did.

Method	P_f^h	N_{call}	ϵ_e
MCS	2.70×10^{-4}	10^5	-
NOH	2.80×10^{-4}	1000 (Training) + 100 (Evaluating)	3.70%
NH	3.00×10^{-4}	1000 (Training) + 875 (Evaluating)	11.11%

The convergence of the NOH method is illustrated and compared with that of the NH method in Figure 8. Although both methods converged quickly, NOH converged in noticably fewer iterations compared to the NH method. With only 100 evaluations, the NOH method could accurately estimate the failure probability as $P_f^h = 2.80 \times 10^{-4}$, with a relative error of 3.70%, while the NH method only achieved 11.11% while utilizing 875 evaluations. The faster convergence and lower relative error observed in the

NOH method signify the precision and capabilities of the neural operator surrogate when compared to the neural surrogate.

These results highlight the potential of neural operator surrogates to significantly enhance computational efficiency and accuracy in a variety of applications. Consequently, the NOH method emerges as a more efficient and effective approach for estimating failure probability.

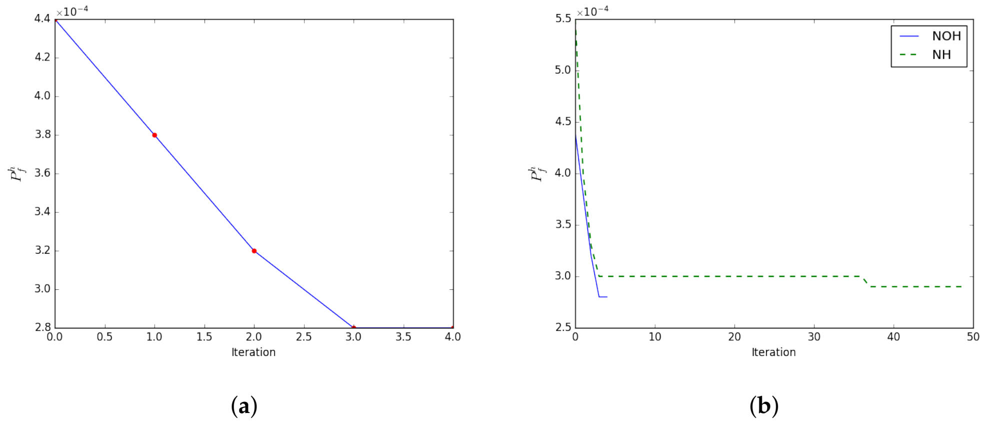


Figure 8. Convergence of the NOH method and the NH method. Panel (**a**) demonstrates the convergence of the NOH method, while in (**b**), a comparison of the convergence rates between the NOH and the NH methods is conducted. For both methods, we set $\delta M = 25$ in each iteration. For NOH, ϵ was set to 0, and the iterative procedure was terminated when $\delta P \leq \epsilon$ for the first time. (**a**) Convergence of the NOH method. (**b**) Comparison between the NOH and NH methods.

5. Conclusions

This paper introduced a neural operator hybrid method for the estimation of failure probability. Instead of approximating the limit state function directly, we reframe the problem as an operator learning task. This allows us to construct a highly efficient and precise surrogate operator model that can accurately estimate the limit state function. By integrating the surrogate operator model into the hybrid algorithm, we created the neural operator hybrid method. The numerical results demonstrate that the proposed method provides an efficient strategy for estimating failure probability, particularly in systems governed by ODEs, multivariate functions, and the Helmholtz equation. Our proposed method exhibited superior performance to that of the basic MCS approach, particularly in terms of efficiency. Furthermore, it surpassed the previous neural hybrid method in both efficiency and accuracy. Consequently, it is applicable and beneficial for addressing general failure probability estimation problems. The obtained results not only demonstrate the efficacy of neural operator learning frameworks in the context of failure probability estimation, but also imply their promising potential in other areas, such as Bayesian inverse problems and partial differential equations with random inputs. In our future work, techniques such as importance sampling (IS) [8,35] or adaptive learning [20,36] can further reduce the sample size required in the estimation process.

Author Contributions: Conceptualization, M.L. and G.W.; methodology, M.L.; software, M.L.; validation, M.L.; investigation, M.L., Y.F. and G.W.; data curation, M.L.; writing—original draft preparation, M.L., Y.F. and G.W.; writing—review and editing, M.L., Y.F. and G.W.; visualization, M.L.; supervision, M.L.; project administration, M.L.; funding acquisition, M.L., Y.F. and G.W. All authors have read and agreed to the published version of the manuscript.

Funding: M. Li and Y. Feng were supported by the National Natural Science Foundation of China (No. 12071291), the Science and Technology Commission of Shanghai Municipality (No. 20JC1414300), and the Natural Science Foundation of Shanghai (No. 20ZR1436200). G. Wang was supported by the Colleges and Universities Young Teachers' Training and Funding Program of Shanghai (No. ZZLX21039).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bjerager, P. Probability integration by directional simulation. *J. Eng. Mech.* **1988**, *114*, 1285–1302. [CrossRef]
2. Ditlevsen, O.; Bjerager, P. Methods of structural systems reliability. *Struct. Saf.* **1986**, *3*, 195–229. [CrossRef]
3. Der Kiureghian, A.; Dakessian, T. Multiple design points in first and second-order reliability. *Struct. Saf.* **1998**, *20*, 37–49. [CrossRef]
4. Hohenbichler, M.; Gollwitzer, S.; Kruse, W.; Rackwitz, R. New light on first-and second-order reliability methods. *Struct. Saf.* **1987**, *4*, 267–284. [CrossRef]
5. Rajashekhar, M.R.; Ellingwood, B.R. A new look at the response surface approach for reliability analysis. *Struct. Saf.* **1993**, *12*, 205–220. [CrossRef]
6. Khuri, A.I.; Mukhopadhyay, S. Response surface methodology. *Wiley Interdiscip. Rev. Comput. Stat.* **2010**, *2*, 128–149. [CrossRef]
7. Li, J.; Xiu, D. Evaluation of failure probability via surrogate models. *J. Comput. Phys.* **2010**, *229*, 8966–8980. [CrossRef]
8. Li, J.; Li, J.; Xiu, D. An efficient surrogate-based method for computing rare failure probability. *J. Comput. Phys.* **2011**, *230*, 8683–8697. [CrossRef]
9. Ghanem, R.G.; Spanos, P.D. *Stochastic Finite Elements: A Spectral Approach*; Courier Corporation: Chelmsford, MA, USA, 2003.
10. Xiu, D.; Karniadakis, G.E. The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput.* **2002**, *24*, 619–644. [CrossRef]
11. Boyaval, S.; Bris, C.L.; Lelièvre, T.; Maday, Y.; Nguyen, N.C.; Patera, A.T. Reduced basis techniques for stochastic problems. *Arch. Comput. Methods Eng.* **2010**, *4*, 435–454. [CrossRef]
12. Quarteroni, A.; Manzoni, A.; Negri, F. *Reduced Basis Methods for Partial Differential Equations: An Introduction*; Springer: Berlin/Heidelberg, Germany, 2016.
13. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.* **1989**, *2*, 303–314. [CrossRef]
14. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
15. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
16. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G.E. DeepXDE: A deep learning library for solving differential equations. *SIAM Rev.* **2021**, *63*, 208–228. [CrossRef]
17. Pang, G.; Lu, L.; Karniadakis, G.E. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **2019**, *41*, A2603–A2626. [CrossRef]
18. Zhang, D.; Guo, L.; Karniadakis, G.E. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *SIAM J. Sci. Comput.* **2020**, *42*, A639–A665. [CrossRef]
19. Li, K.; Tang, K.; Li, J.; Wu, T.; Liao, Q. A hierarchical neural hybrid method for failure probability estimation. *IEEE Access* **2019**, *7*, 112087–112096. [CrossRef]
20. Lieu, Q.X.; Nguyen, K.T.; Dang, K.D.; Lee, S.; Kang, J.; Lee, J. An adaptive surrogate model to structural reliability analysis using deep neural network. *Expert Syst. Appl.* **2022**, *189*, 116104. [CrossRef]
21. Yao, C.; Mei, J.; Li, K. A Mixed Residual Hybrid Method For Failure Probability Estimation. In Proceedings of the 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, 11–13 December 2022; IEEE: New York, NY, USA, 2022; pp. 119–124.
22. Chen, T.; Chen, H. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Trans. Neural Netw.* **1993**, *4*, 910–918. [CrossRef] [PubMed]
23. Chen, T.; Chen, H. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Trans. Neural Netw.* **1995**, *6*, 904–910. [CrossRef]
24. Lu, L.; Meng, X.; Cai, S.; Mao, Z.; Goswami, S.; Zhang, Z.; Karniadakis, G.E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *Comput. Methods Appl. Mech. Eng.* **2022**, *393*, 1–42. [CrossRef]
25. Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv* **2020**, arXiv:2010.08895.

26. Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; Karniadakis, G.E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat. Mach. Intell.* **2021**, *3*, 218–229. [CrossRef]
27. Lin, C.; Li, Z.; Lu, L.; Cai, S.; Maxey, M.; Karniadakis, G.E. Operator learning for predicting multiscale bubble growth dynamics. *J. Chem. Phys.* **2021**, *154*, 104118. [CrossRef]
28. Cai, S.; Wang, Z.; Lu, L.; Zaki, T.A.; Karniadakis, G.E. DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *J. Comput. Phys.* **2021**, *436*, 110296.
29. Mao, Z.; Lu, L.; Marxen, O.; Zaki, T.A.; Karniadakis, G.E. DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *J. Comput. Phys.* **2021**, *447*, 110698.
30. Papadrakakis, M.; Lagaros, N.D. Reliability-based structural optimization using neural networks and Monte Carlo simulation. *Comput. Methods Appl. Mech. Eng.* **2002**, *191*, 3491–3507. [CrossRef]
31. Kutyłowska, M. Neural network approach for failure rate prediction. *Eng. Fail. Anal.* **2015**, *47*, 41–48. [CrossRef]
32. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
33. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
34. Engelund, S.; Rackwitz, R. A benchmark study on importance sampling techniques in structural reliability. *Struct. Saf.* **1993**, *12*, 255–276. [CrossRef]
35. Tabandeh, A.; Jia, G.; Gardoni, P. A review and assessment of importance sampling methods for reliability analysis. *Struct. Saf.* **2022**, *97*, 102216. [CrossRef]
36. Teixeira, R.; Nogal, M.; O'Connor, A. Adaptive approaches in metamodel-based reliability analysis: A review. *Struct. Saf.* **2021**, *89*, 102019. [CrossRef]