Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.083227

ARTICLE

A Deep-Learning-Based Constitutive Method for Geomaterials Using a Neural Cutting Plane Algorithm

Qingxiang Meng[1,2,*], **Zijie He**[1,2], **Yajun Cao**[1,2] **and Weijiang Chu**[3]

[1]Key Laboratory of Ministry of Education for Geomechanics and Embankment Engineering, Hohai University, Nanjing, China
[2]Research Institute of Geotechnical Engineering, Hohai University, Nanjing, China
[3]Powerchina Huadong Engineering Corporation Limited, Hangzhou, China

*Corresponding Author: Qingxiang Meng. Email: mqx@hhu.edu.cn

Received: 31 March 2026; Accepted: 06 May 2026; Published: 30 June 2026

ABSTRACT: Constitutive modeling for geomaterials remains challenging because of limited data availability, strong nonlinearity, pressure sensitivity, and the non-smooth characteristics of commonly used yield surfaces. This study presents a deep-learning-based constitutive method for geomaterials that incorporates a neural stress-integration procedure based on the cutting plane algorithm (CPA). Two compact fully connected networks are trained to learn the yield function and its stress gradient from an augmented stress-state dataset. The trained networks are then incorporated into a cutting plane return-mapping procedure, in which only first-order information is required for the plastic stress return. This avoids explicit analytical yield expressions and second-derivative evaluations and is therefore more naturally compatible with non-smooth Mohr–Coulomb-type yield-surface representations in a first-order return-mapping sense. Numerical results show that the proposed method reproduces the reference Mohr–Coulomb response along the examined monotonic triaxial compression paths. Compared with the finite-difference closest-point projection method (CPPM) implementation considered in this study, the CPA-based neural stress-update procedure requires fewer network calls per update, indicating a more economical implementation for the present learned constitutive framework.

KEYWORDS: Geomaterials; constitutive modeling; deep learning; cutting plane algorithm; stress integration

1 Introduction

Constitutive modeling of geomaterials remains challenging because soils and other geotechnical materials often exhibit strong nonlinearity, path dependence, anisotropy, pressure sensitivity, and strain softening, while available laboratory data usually provide only limited coverage of relevant stress states and loading paths. Accurately representing stress–strain responses across different initial conditions and environmental factors has therefore long relied on restrictive analytical assumptions and carefully designed numerical integration procedures [1–3].

Data-driven approaches provide an alternative by learning constitutive responses directly from curated experiments or numerical datasets. Early studies explored material informatics and neural approximators for constitutive representation [4–6], motivated in part by the universal approximation capability of neural networks [7]. More recent developments have incorporated physical structure to improve robustness and interpretability, including constitutive artificial neural networks with frame indifference, thermodynamic consistency, and polyconvexity for hyperelasticity [8], iCANN for inelastic behavior with pseudo-potential and thermodynamic constraints [9], and frameworks that jointly learn free energies, flow rules, and yield

conditions [10]. These thermodynamics- or architecture-based approaches, such as TANN and iCANN-type models, aim to embed physical admissibility into the network formulation through free-energy functions, pseudo-potentials, internal variables, and evolution equations [11]. Another closely related line is level-set plasticity, where yield surfaces are represented as implicit neural functions and augmented signed-distance fields are used to provide off-surface training information [12,13]. In geotechnical applications, multi-fidelity learning and level-set-based deep learning frameworks have further demonstrated the potential of data-driven methods for representing complex soil behavior and constructing reproducible modeling pipelines [14,15]. Graph-network and learnable-physics-engine approaches have also recently been explored for geomaterials and solid-mechanics problems, but these methods usually operate closer to the solver or field scale [16].

Despite these advances, two practical challenges still limit the use of neural constitutive models in geomechanics. First, many existing models originate from metal plasticity and adopt pressure-insensitive yield criteria such as von Mises, which are not well suited to geomaterials whose mechanical response is strongly affected by confining pressure [17]. Second, many deep-learning-based constitutive frameworks embed the learned yield function in a closest-point projection method and rely on second-order derivative information. This becomes inconvenient for non-smooth or cornered yield surfaces typical of Mohr–Coulomb-type behavior and may reduce the robustness of stress integration [18,19].

To address these issues, this study develops a deep-learning-based constitutive method that separately learns the yield function and its stress gradient from a level-set-augmented stress-state dataset and incorporates them into classical return-mapping procedures. In this context, the main contribution of this work is the construction and assessment of a neural stress-integration framework for Mohr–Coulomb-type geomaterials. The framework learns the yield function and its stress-gradient information and uses them in a CPA-based first-order stress-update procedure. The focus is therefore on the use of learned yield-surface geometry in local plastic stress integration, rather than on discovering a complete thermodynamics-by-design constitutive architecture or replacing an analytical Mohr–Coulomb return mapping. The proposed method is assessed through triaxial compression simulations under different confining pressures, and its algorithmic characteristics are compared with those of the classical closest-point projection method. The results show that the proposed method reproduces the reference response along the examined monotonic triaxial compression paths and requires fewer network calls per update than the finite-difference CPPM implementation considered in this study.

The remainder of this paper is organized as follows. Section 2 presents the proposed methodology, including dataset preparation, neural network training, and the CPPM- and CPA-based stress-integration procedures. Section 3 describes the preparation of the Mohr–Coulomb benchmark dataset, the training settings of the two neural networks, and the triaxial-compression validation results. Section 4 discusses the computational characteristics of the proposed method and examines the effects of dataset range and network architecture on model performance. Finally, Section 5 summarizes the main conclusions, while the Supplementary Appendices provide additional derivations, pseudocode, and data-preparation details.

2 Methods

2.1 Overview

The method starts from raw stress data obtained either from experiments or from established yield criteria, followed by level-set augmentation to generate stress states located on and off the yield surface [12]. Two compact fully connected networks are then trained in PyTorch [20] to predict the yield function and its stress gradient. Consistent normalization is maintained throughout data preparation, network

training, and inference. The trained networks are subsequently embedded into return-mapping algorithms for stress integration, including the cutting plane algorithm (CPA) and the closest-point projection method (CPPM). Fig. 1 summarizes the overall procedure, in which data collection and level-set augmentation constitute the data-preparation stage.

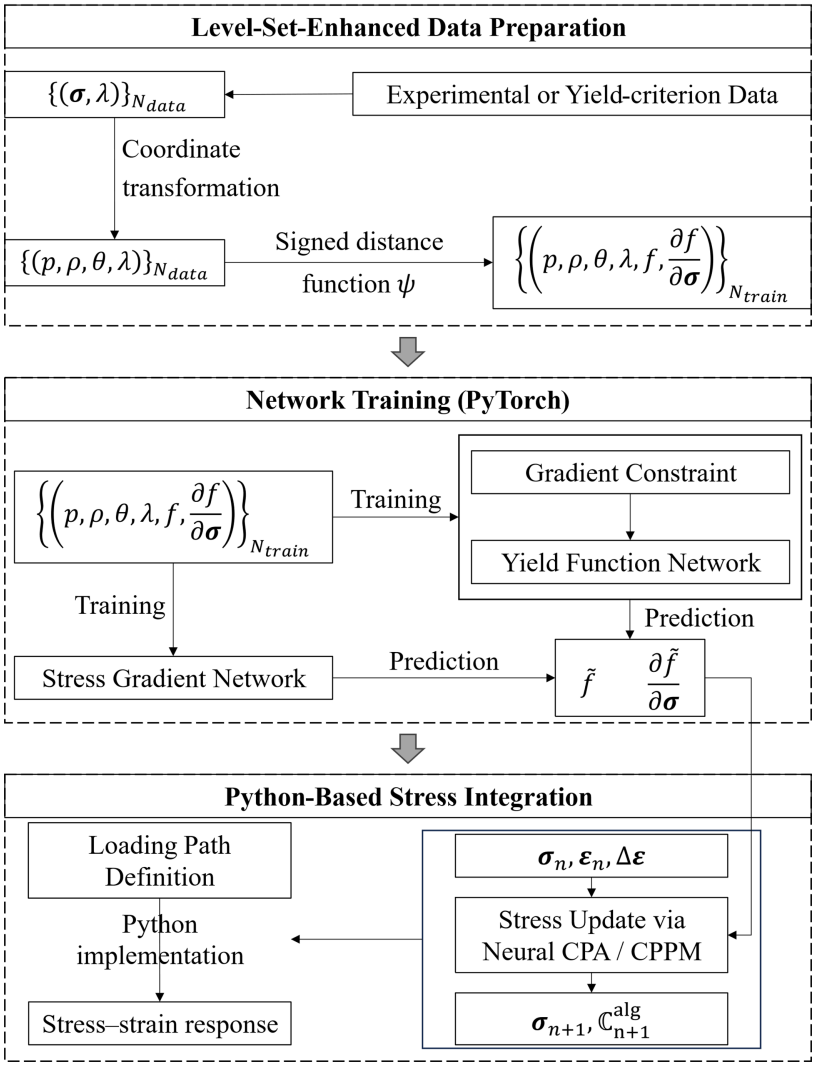

Figure 1: Overview of the proposed method.

2.2 Neural Network Training

2.2.1 Network Architecture

To balance predictive capability and numerical robustness, the network architecture is kept compact. The proposed models consist of stacked Dense layers with optional element-wise Multiply blocks, and the layer width is treated as a tunable hyperparameter. The Rectified Linear Unit (ReLU) activation is adopted because of its simple form and computational efficiency in repeated stress-update evaluations [21]. For a hidden layer, the mapping is written as

$$\boldsymbol{h}^{(l+1)} = \mathrm{ReLU}\left(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l)} + b^{(l)}\right) \tag{1}$$

where $\boldsymbol{W}^{(l)}$ is the weight matrix of layer l, $b^{(l)}$ is the bias vector.

Under the Sobolev training strategy introduced in Section 2.2.3, the network is required to approximate both the target function and its gradient. To improve representational flexibility, an element-wise Multiply layer is introduced,

$$\boldsymbol{h}^{(l+1)} = \text{Multiply}\left(\boldsymbol{h}^{(l)}\right) = \boldsymbol{h}^{(l)} \circ \boldsymbol{h}^{(l)} \tag{2}$$

where $\circ$ denotes the Hadamard product.

Both the yield-function network and the stress-gradient network adopt the same fully connected architecture, as illustrated in Fig. 2. Here, WIDTH denotes the number of neurons in each hidden layer. The influence of network architecture is examined in Section 4.3. Other hyperparameters, such as the learning rate and optimizer, are not the main focus of this study. Standard settings are therefore adopted, while these parameters can be further tuned using common optimization tools such as Ray Tune [22].

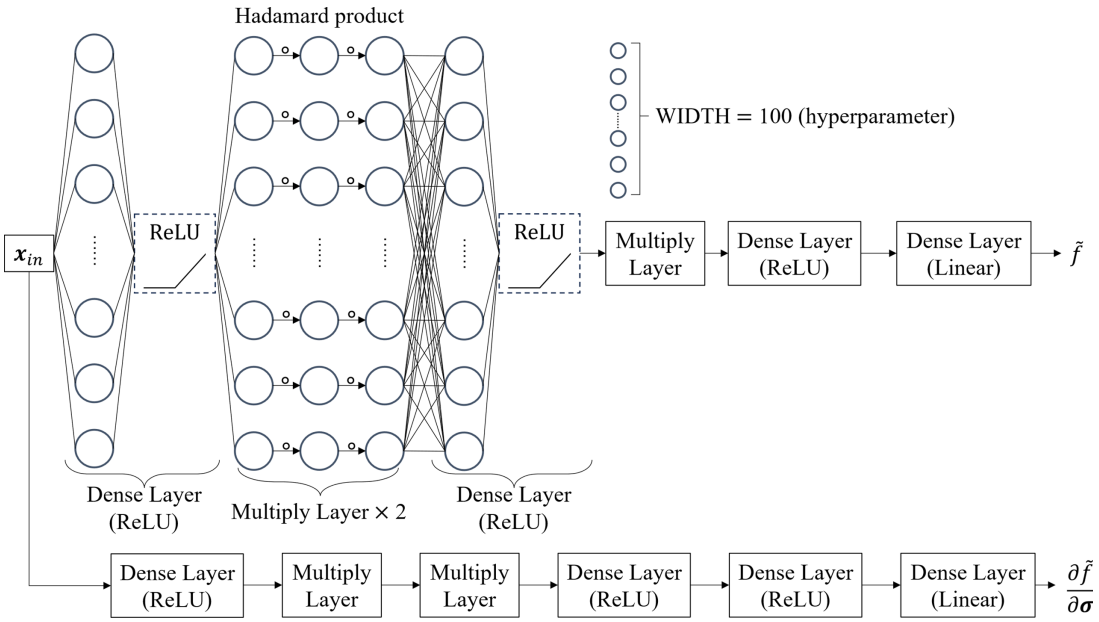

Figure 2: Architectures of the yield-function network and the stress-gradient network.

2.2.2 Dataset Preparation and Augmentation

Consider a convex yield function f. Under loading, the yield surface may evolve with internal variables ξ. For a given internal variable, the elastic domain is defined by [23]:

$$\mathbb{E} = \{(\boldsymbol{\sigma}, \xi) \mid f(\boldsymbol{\sigma}, \xi) \leq 0\} \tag{3}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress. In this study, the internal variable ξ is simplified as the accumulated plastic strain λ. More generally, the formulation is written as $f = f(p, \rho, \theta, \lambda)$. Therefore, the framework is not limited to a fixed yield surface and can, in principle, be extended to evolving yield surfaces associated with hardening or softening. For the non-softening Mohr–Coulomb benchmark used in Section 3, however, λ is a passive input and the yield function is independent of it.

To improve data efficiency, material symmetry is exploited. For isotropic plasticity, the stress state is represented in the cylindrical coordinate system of the π-plane as $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(p, \rho, \theta)$, where p is the mean-stress coordinate under the tension-positive convention, and ρ and θ are the Lode radius and Lode angle, respectively [13]. The transformation between cylindrical coordinates and principal stresses follows the standard relations in [23], and the details are provided in Appendix S1 in the Supplementary Material. The original dataset therefore consists of stress states and accumulated plastic strain expressed in cylindrical coordinates,

$$\hat{\boldsymbol{x}}_\lambda\left(p, \rho, \theta, \lambda\right) = \boldsymbol{x}_\lambda\left(\sigma_1, \sigma_2, \sigma_3, \lambda\right) \tag{4}$$

Since yielded samples lie on the surface $f = 0$, training only on boundary data provides limited information for nearby interior and exterior states. To improve generalization and stress-integration robustness, a level-set-based augmentation strategy is adopted [12]. The yield function is represented by a signed distance function ψ,

$$\psi\left(\hat{\boldsymbol{x}}, \lambda\right) = \begin{cases} d\left(\hat{\boldsymbol{x}}\right) & f\left(\hat{\boldsymbol{x}}, \lambda\right) > 0 \\ 0 & f\left(\hat{\boldsymbol{x}}, \lambda\right) = 0 \\ -d\left(\hat{\boldsymbol{x}}\right) & f\left(\hat{\boldsymbol{x}}, \lambda\right) < 0 \end{cases} \tag{5}$$

where $\hat{\boldsymbol{x}}$ denotes an arbitrary stress point in cylindrical coordinates, and $d(\hat{\boldsymbol{x}})$ is the minimum Euclidean distance from $\hat{\boldsymbol{x}}$ to the yield surface $f(\hat{\boldsymbol{x}}, \lambda) = 0$ at a given λ, defined as

$$d\left(\hat{\boldsymbol{x}}\right) = \min\left(\left\| \hat{\boldsymbol{x}} - \hat{\boldsymbol{x}}_\lambda \right\|\right) \tag{6}$$

where $\| \cdot \|$ denotes the L^2 norm.

The signed-distance representation satisfies the Eikonal equation [24],

$$\left\|\nabla^{\hat{\boldsymbol{x}}}\psi\right\| = 1 \tag{7}$$

which states the unit-gradient property near the interface. In cylindrical coordinates, Eq. (7) becomes:

$$\left(\frac{\partial\psi}{\partial\rho}\right)^2 + \frac{1}{\rho^2}\left(\frac{\partial\psi}{\partial\theta}\right)^2 = 1 \tag{8}$$

In the actual data augmentation, no global Eikonal solver is used. Auxiliary samples are generated directly from each original on-surface point by scaling the ρ coordinate while keeping p, θ, and λ unchanged. In this study, $N_\psi = 10$ auxiliary levels are used in the interval $[0, 2]$; the level closest to 1 is set to 1 to retain the original yield-surface samples, and the first level is set to $\delta = 10^{-5}$ to avoid generating samples exactly at $\rho = 0$.

For more general hardening/softening cases, the level-set representation may be viewed as evolving with the internal variable λ. In that setting, a corresponding Hamilton–Jacobi form may be written as:

$$\frac{\partial\psi}{\partial\lambda} + \mathbf{v}\cdot\nabla_{\hat{\mathbf{x}}}\psi = 0, \mathbf{v} = F\mathbf{n}, \quad \mathbf{n} = \frac{\nabla_{\hat{\mathbf{x}}}\psi}{\left\| \nabla_{\hat{\mathbf{x}}}\psi \right\|} \tag{9}$$

where ψ is the level-set function, $\hat{\mathbf{x}}$ denotes the stress coordinates, $\mathbf{v}$ is the pseudo-velocity field, F is a scalar magnitude, and $\mathbf{n}$ is the normal direction of the level set. This relation is introduced here only to clarify the extensibility of the framework, while the benchmark studied in Section 3 remains the non-softening Mohr–Coulomb case.

Let N_{data} denotes the number of original on-surface samples. Based on the signed distance function, N_{ψ} auxiliary layers are generated along the normal direction on both sides of the yield surface. The final training set contains $N_{train} = N_{\psi} \times N_{data}$ samples, written as $\{(\boldsymbol{\sigma}, \lambda)\}_{N_{train}}$. The augmentation procedure is illustrated in Fig. 3, where the red points denote the original yield-surface data and the blue points denote the augmented samples.

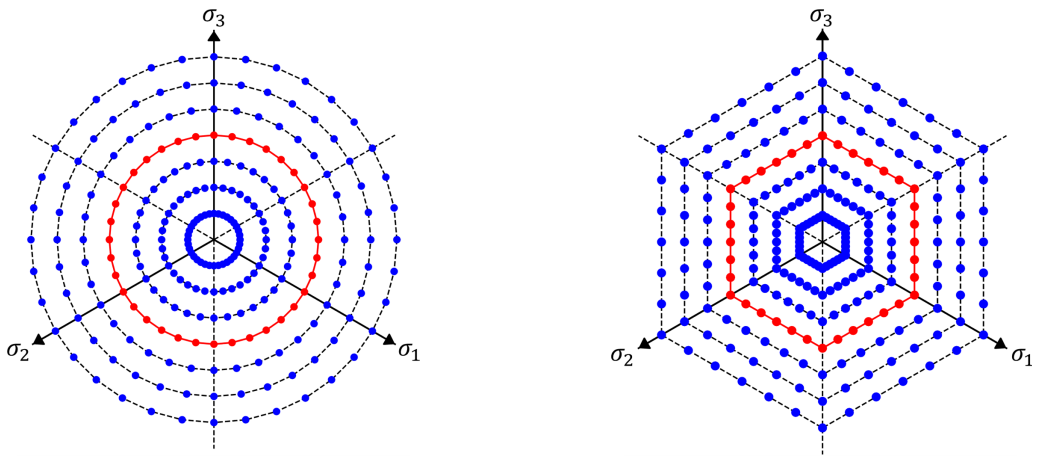

Figure 3: Auxiliary samples generated by the level-set method.

2.2.3 Learning the Yield Function and Stress Gradient

After data augmentation, the original yield-surface samples are transformed into a signed-distance representation, giving the training dataset $\{(p, \rho, \theta, \lambda, f)\}_{N_{train}}$. Here, the level-set form of the yield function is defined as $f(p, \rho, \theta, \lambda) = \psi(\hat{\boldsymbol{x}}, \lambda)$, and $\tilde{f} = \tilde{f}(p, \rho, \theta, \lambda)$ denotes the neural-network prediction. It should be noted that the signed-distance representation is used here as a geometric learning target and does not imply that the Mohr–Coulomb yield surface becomes globally smooth. At smooth portions of the yield surface, the gradient gives a unique outward normal; at corners or apexes, the classical normal is not single-valued and should be understood in a generalized sense. Thus, the learned level-set field is interpreted as a regularized first-order geometric direction for stress integration rather than as an exact pointwise reconstruction of the physical edge.

Standard mean squared error (MSE) regression matches only function values and does not explicitly constrain derivatives [25]. To improve both function approximation and gradient consistency, Sobolev training is adopted [26]. A similar derivative-aware learning idea has also been used in graph-network-based learnable physics engines for crack propagation and coalescence, where Sobolev-type training improves the physical consistency and predictive stability of learned solid-mechanics models [27]. The loss function is written as

$$\mathcal{L}_f^* = \frac{1}{N} \sum_{i=1}^{N} \left[\left(f - \tilde{f} \right)^2 + \eta \left\| \nabla^{\hat{\boldsymbol{x}}} f - \nabla^{\hat{\boldsymbol{x}}} \tilde{f} \right\|^2 \right] \tag{10}$$

where $N = N_{train} = N_{\psi} \times N_{data}$ is the total number of training samples, and η balances the function-value and gradient errors. In this study, $\eta = 1$ was used after input-output normalization, giving equal weight to the function-value and gradient terms in the Sobolev loss. According to Eq. (7), the signed-distance function satisfies $| \nabla^{\hat{\boldsymbol{x}}} f | = 1$. The target gradients can therefore be obtained from the level-set construction or by automatic differentiation in Python [28].

Sobolev training is introduced here to improve the local consistency between the learned level-set function and its gradient. However, it does not by itself provide a rigorous guarantee of global convexity or full thermodynamic admissibility. In the present framework, convexity is inherited from the convex benchmark used for supervised data generation, while stricter convexity-preserving and thermodynamic constraints remain a topic for future work.

Plastic return mapping requires the first derivative of the yield function. Under an associated flow rule, the plastic flow direction is given by [29]:

$$\dot{\boldsymbol{\varepsilon}}^{p} = \dot{\lambda}\boldsymbol{a}, \quad \boldsymbol{a} = \frac{\partial f}{\partial \boldsymbol{\sigma}} \tag{11}$$

where $\dot{\lambda}$ is the plastic multiplier. To provide this quantity directly during stress integration, a second neural network is trained to predict the stress gradient, using the supervised dataset $\left\{(p, \rho, \theta, \lambda, f, \frac{\partial f}{\partial \sigma})\right\}_{N_{train}}$. This network is a supervised approximation to the analytical reference gradient and is not constrained to be the exact autograd derivative of the trained yield-function network. This design enables forward-only evaluation of the return direction during stress integration.

This stress-gradient network minimizes the discrepancy between the target stress gradient $\frac{\partial f}{\partial \sigma_i}$ and the network prediction $\frac{\partial \tilde{f}}{\partial \sigma_i} = \frac{\partial \tilde{f}}{\partial \sigma_i}(p, \rho, \theta, \lambda)$. Its loss function is taken as the MSE,

$$\mathcal{L}_{\partial f} = \frac{1}{N}\sum_{j=1}^{N}\left(\sum_{i=1}^{3}\left\|\frac{\partial f}{\partial \sigma_i} - \frac{\partial \tilde{f}}{\partial \sigma_i}\right\|^{2}\right) \tag{12}$$

2.3 Neural Network Inference and Stress Integration

2.3.1 Neural Network Inference in Python

After training, the network parameters are exported from PyTorch and loaded into the stress-update procedure implemented in Python. The layer topology, weights, and biases are read once before the loading-path simulation and remain fixed during the analysis. At each stress-update step, the input variables are normalized using the same procedure as in training, and forward inference is performed to evaluate the yield function and the stress gradient required by the return-mapping algorithm.

Bias vectors are stored in ***BArr***, and weight matrices are stored in ***WArr***. For each stress-update step, the normalized input forms ***xiLayr*** for the first layer. The forward propagation of a dense layer consists of matrix-vector multiplication, bias addition, and application of the activation function $\sigma(\cdot)$, producing the activated output ***reLayr***. For a dense layer i with n_{i-1} inputs and n_i outputs, the forward update is

$$\begin{aligned} \boldsymbol{xoLayr}(j) &= \sum_{k=1}^{n_{i-1}} \boldsymbol{WArr}(j,k,i)\,\boldsymbol{xiLayr}(k) + \boldsymbol{BArr}(j,i), \quad j = 1,\ldots,n_i, \\ \boldsymbol{reLayr}(j) &= \sigma\left(\boldsymbol{xoLayr}(j)\right), \quad j = 1,\ldots,n_i \end{aligned} \tag{13}$$

This procedure is repeated layer by layer until the final network output is obtained. The predicted quantities are then passed to the subsequent stress-update algorithm. Appendix S2 in the Supplementary Material provides pseudocode for neural network inference within the stress-update procedure implemented in Python.

2.3.2 CPPM-Based Stress Update

For stress integration in the plastic regime, a deep-learning-based CPPM return-mapping scheme is adopted. Once the material enters plastic loading, the incremental constitutive update at step $n+1$ is written as [18]:

$$\begin{aligned}
&\boldsymbol{\varepsilon}_{n+1} = \boldsymbol{\varepsilon}_n + \Delta\boldsymbol{\varepsilon} \\
&\boldsymbol{\varepsilon}^p_{n+1} = \boldsymbol{\varepsilon}^p_n + \Delta\lambda_{n+1}\boldsymbol{a}_{n+1} \\
&\boldsymbol{\sigma}_{n+1} = \boldsymbol{D}^e\left(\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}^p_{n+1}\right) \\
&f_{n+1} = f\left(\boldsymbol{\sigma}_{n+1}\right) = 0
\end{aligned} \tag{14}$$

Under the associated flow rule, the plastic flow direction is given by $\mathbf{a} = \partial f/\partial\boldsymbol{\sigma}$. In the present framework, this quantity is provided by the stress-gradient neural network, denoted by $f_{grad}(\boldsymbol{\sigma}, \lambda)$, while the yield-function network is denoted by $f_{NN}(\boldsymbol{\sigma}, \lambda)$. The stress update starts from the elastic predictor. Assuming a purely elastic response during the current increment, the trial stress $\boldsymbol{\sigma}^{trial}_{n+1}$ is first computed as:

$$\boldsymbol{\sigma}^{\text{trial}}_{n+1} = \boldsymbol{\sigma}_n + \boldsymbol{D}^e\Delta\boldsymbol{\varepsilon} \tag{15}$$

If the trial state violates the yield condition, a plastic correction is required. In the CPPM framework, the trial stress is projected back toward the yield surface along the direction determined by the associated flow rule, which can be written as

$$\begin{aligned}
&\Delta\boldsymbol{\varepsilon}^p_{n+1} = \Delta\lambda_{n+1}\boldsymbol{a}_{n+1} \\
&\Delta\boldsymbol{\sigma}_{n+1} = -\boldsymbol{D}^e\Delta\boldsymbol{\varepsilon}^p_{n+1} = -\Delta\lambda_{n+1}\boldsymbol{D}^e\boldsymbol{a}_{n+1} \\
&\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}}_{n+1} + \Delta\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}}_{n+1} - \Delta\lambda_{n+1}\boldsymbol{D}^e\boldsymbol{a}_{n+1}
\end{aligned} \tag{16}$$

In order to satisfy the consistency condition, it is necessary to solve the nonlinear equation for $\Delta\lambda$. At loading step $n+1$, Newton iteration is used, where n denotes the loading step and k the local iteration. Taking the increment $\delta\lambda^k$ of $\Delta\lambda$ in the k-th iteration, the nonlinear equation is approximated by first-order Taylor expansion:

$$\begin{aligned}
&\boldsymbol{g}^k + \left(\frac{d\boldsymbol{g}}{d\Delta\lambda}\right)^k\delta\lambda^k = 0 \\
&\Delta\lambda^{k+1} = \Delta\lambda^k + \delta\lambda^k
\end{aligned} \tag{17}$$

From Eq. (14), the residual form $\boldsymbol{r}$ is obtained. At this time, the nonlinear equations to be solved are:

$$\begin{cases}
\boldsymbol{r} = -\boldsymbol{\varepsilon}^p + \boldsymbol{\varepsilon}^p_n + \Delta\lambda\boldsymbol{a} = 0 \\
f = f\left(\boldsymbol{\sigma}\right) = 0
\end{cases} \tag{18}$$

At the k-th iteration, the above equation is first-order linearized as:

$$\begin{cases}
\boldsymbol{r}^k + \left[\boldsymbol{D}^e\right]^{-1}\Delta\boldsymbol{\sigma}^k + \Delta\lambda^k\Delta\boldsymbol{a}^k + \delta\lambda^k\boldsymbol{a}^k = 0 \\
f^k + \left(\boldsymbol{a}^k\right)^T\Delta\boldsymbol{\sigma}^k = 0
\end{cases} \tag{19}$$

where $\delta\lambda^k$ is the correction of $\Delta\lambda$ in this iteration. We denote the yield-function network prediction at the k-th iteration by $f^k = f_{\text{NN}}(\boldsymbol{\sigma}^k, \lambda^k)$. Substituting $\Delta\boldsymbol{a}^k = \frac{\partial\boldsymbol{a}^k}{\partial\boldsymbol{\sigma}}\Delta\boldsymbol{\sigma}^k$ into Eq. (19), the solution of the equation set is:

$$\delta\lambda^k = \frac{f^k - \left(\boldsymbol{a}^k\right)^T \boldsymbol{R}^k \boldsymbol{r}^k}{\left(\boldsymbol{a}^k\right)^T \boldsymbol{R}^k \boldsymbol{a}^k}$$
$$\Delta\boldsymbol{\sigma}^k = -\boldsymbol{R}^k \boldsymbol{r}^k - \delta\lambda^k \boldsymbol{R}^k \boldsymbol{a}^k \tag{20}$$

where

$$\boldsymbol{R}^k = \left(\boldsymbol{I} + \Delta\lambda^k \boldsymbol{D}^e \frac{\partial \boldsymbol{a}^k}{\partial \boldsymbol{\sigma}}\right)^{-1} \boldsymbol{D}^e$$

where $\frac{\partial \mathbf{a}^k}{\partial \boldsymbol{\sigma}} = \frac{\partial^2 f^k}{\partial \boldsymbol{\sigma}^2}$. Because the gradient network returns only first-order derivatives, CPPM still requires second-order information. We approximate this term by finite differences; the forward-difference scheme is detailed in Appendix S3 in the Supplementary Materials. The k-th iteration stress increment, updated stress, and plastic strain are then obtained by Eq. (20):

$$\left(\boldsymbol{\varepsilon}^p\right)^{k+1} = \left(\boldsymbol{\varepsilon}^p\right)^k + \Delta\left(\boldsymbol{\varepsilon}^p\right)^k = \left(\boldsymbol{\varepsilon}^p\right)^k - \left(\boldsymbol{D}^e\right)^{-1} \Delta\boldsymbol{\sigma}^k$$
$$\Delta\lambda^{k+1} = \Delta\lambda^k + \delta\lambda^k$$
$$\boldsymbol{\sigma}^{k+1} = \boldsymbol{\sigma}^k + \Delta\boldsymbol{\sigma}^k \tag{21}$$

After each local iteration, the updated stress $\boldsymbol{\sigma}^{k+1}$ is substituted into Eq. (18) to evaluate the yield-function value and the residual norm. If both satisfy the prescribed tolerance, the local iteration is terminated. The converged stress $\boldsymbol{\sigma}^{k+1}$ and plastic strain $\left(\boldsymbol{\varepsilon}^p\right)^{k+1}$ are then recorded as the constitutive state at loading step $n+1$. The complete algorithmic procedure of the deep-learning-based CPPM stress update is summarized in Appendix S4 in the Supplementary Materials.

2.3.3 CPA-Based Stress Update

Both CPPM and CPA are iterative return-mapping schemes. In CPPM, the trial stress is corrected using the final gradient, whereas CPA updates the stress incrementally with the current gradient at each local iteration [30].

For non-smooth yield criteria such as Mohr–Coulomb, CPPM becomes more involved because second-order derivatives are difficult to define or evaluate robustly near corners or apex regions. By contrast, CPA only requires the current yield-function value and a first-order return direction, which makes it more compatible with a regularized level-set representation of cornered yield surfaces [31]. For perfectly plastic materials with piecewise linear yield surfaces, CPA and CPPM are equivalent in the returned stress and consistent tangent, indicating that CPA can recover the same local constitutive solution [18]. Analogously to Eq. (16), the stress update at the k-th local iteration is written as

$$\Delta\boldsymbol{\sigma}^k = \boldsymbol{\sigma}^{k+1} - \boldsymbol{\sigma}^k = -\boldsymbol{D}^e \Delta\left(\boldsymbol{\varepsilon}^p\right)^k \tag{22}$$

where k denotes the local iteration counter. The associated flow rule is imposed at the beginning of each local iteration,

$$\Delta\left(\boldsymbol{\varepsilon}^p\right)^k = \Delta\lambda^k \boldsymbol{a}^k \tag{23}$$

and substituting Eq. (23) into Eq. (22) gives

$$\Delta\boldsymbol{\sigma}^k = -\Delta\lambda^k \boldsymbol{D}^e \boldsymbol{a}^k \tag{24}$$

For the scalar yield function $f(\boldsymbol{\sigma}) = 0$, the value at the updated stress state $f(\boldsymbol{\sigma}^{k+1})$ is obtained by a first-order Taylor expansion about $\boldsymbol{\sigma}^k$, neglecting higher-order terms,

$$f^{k+1} = f^k + \left(\boldsymbol{a}^k\right)^T \left(\boldsymbol{\sigma}^{k+1} - \boldsymbol{\sigma}^k\right) \tag{25}$$

Combining Eq. (24) with the consistency condition $f^{k+1} = 0$ yields:

$$\Delta\lambda^k = \frac{f^k}{\left(\boldsymbol{a}^k\right)^T \boldsymbol{D}^e \boldsymbol{a}^k} \tag{26}$$

In CPA, the stress gradient **a** and the yield-function value f are predicted by the trained networks $f_{grad}(\boldsymbol{\sigma}, \lambda)$ and $f_{NN}(\boldsymbol{\sigma}, \lambda)$, respectively. The complete iterative procedure is summarized in Appendix S5 in the Supplementary Materials. Since only first-order information is required, CPA is simpler than CPPM and is more naturally compatible with non-smooth or cornered yield-surface representations in a first-order return-mapping sense.

3 Results

3.1 Training Dataset Preparation

The Mohr–Coulomb model [32] is adopted as the reference constitutive model in this study. Owing to its simple form and clear physical interpretation through cohesion and friction angle, it remains widely used in geotechnical analysis, especially for identifying failure modes, deformation trends, and stability characteristics. A well-known difficulty of the Mohr–Coulomb criterion is that its yield surface contains corners, which leads to non-smooth stress gradients. Unless otherwise stated, stresses are expressed using the tension-positive convention in this study; therefore, compressive stresses are negative. In principal stress space, assuming the algebraic ordering $\sigma_1 \geq \sigma_2 \geq \sigma_3$, the Mohr–Coulomb yield criterion is written as [33]:

$$f = (\sigma_1 - \sigma_3) + (\sigma_1 + \sigma_3)\sin\phi - 2c\cos\phi = 0 \tag{27}$$

where c and ϕ represent the cohesion and friction angle of the material.

To avoid repeated direct manipulation of principal stresses, the yield criterion is further expressed in invariant form using the standard formulation for isotropic yield functions [34]. Nayak and Zienkiewicz proposed a convenient set of stress-invariant definitions [35]:

$$\begin{aligned}
\sigma_m &= \frac{1}{3}\left(\sigma_x + \sigma_y + \sigma_z\right) \\
\overline{\sigma} &= \sqrt{\frac{1}{2}\left(s_x^2 + s_y^2 + s_z^2\right) + \tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2} \\
\theta &= \frac{1}{3}\sin^{-1}\left(-\frac{3\sqrt{3}}{2}\frac{J_3}{\overline{\sigma}^3}\right), -30^\circ \leq \theta \leq 30^\circ
\end{aligned} \tag{28}$$

where

$$\begin{aligned}
J_3 &= s_x s_y s_z + 2\tau_{xy}\tau_{yz}\tau_{zx} - s_x\tau_{yz}^2 - s_y\tau_{zx}^2 - s_z\tau_{xy}^2 \\
s_x &= \sigma_x - \sigma_m, \quad s_y = \sigma_y - \sigma_m, \quad s_z = \sigma_z - \sigma_m
\end{aligned}$$

By using the above stress invariants, the principal stress can be re-expressed as:

$$\sigma_1 = \frac{2}{\sqrt{3}}\overline{\sigma}\sin\left(\theta + 120^\circ\right) + \sigma_m$$
$$\sigma_2 = \frac{2}{\sqrt{3}}\overline{\sigma}\sin\left(\theta\right) + \sigma_m$$
$$\sigma_3 = \frac{2}{\sqrt{3}}\overline{\sigma}\sin\left(\theta - 120^\circ\right) + \sigma_m \tag{29}$$

Substituting the expressions of σ_1 and σ_3 into Eq. (27), we get:

$$f = \sigma_m \sin\phi + \overline{\sigma}K\left(\theta\right) - c\cos\phi = 0$$
$$K\left(\theta\right) = \cos\theta - \frac{1}{\sqrt{3}}\sin\phi\sin\theta \tag{30}$$

In this benchmark, c and ϕ are taken as constants, so Eq. (30) does not include hardening or softening evolution with λ. Based on Eq. (30), the discrete sampling parameters N_{σ_m}, N_θ, and N_λ are introduced to generate stress states by prescribing the ranges of λ, σ_m, and θ. The ordering and permutation of the three principal stresses are then used to construct the six symmetry sectors of the Mohr–Coulomb yield surface. Thus, Eq. (27) is written for one ordered principal-stress sector, whereas the dataset and Fig. 4 include the sectors generated by principal-stress permutations. As a result, the initial dataset contains $N_{data} = N_\lambda \times 6 \times \left(N_{\sigma_m} \times N_\theta\right)$ samples.

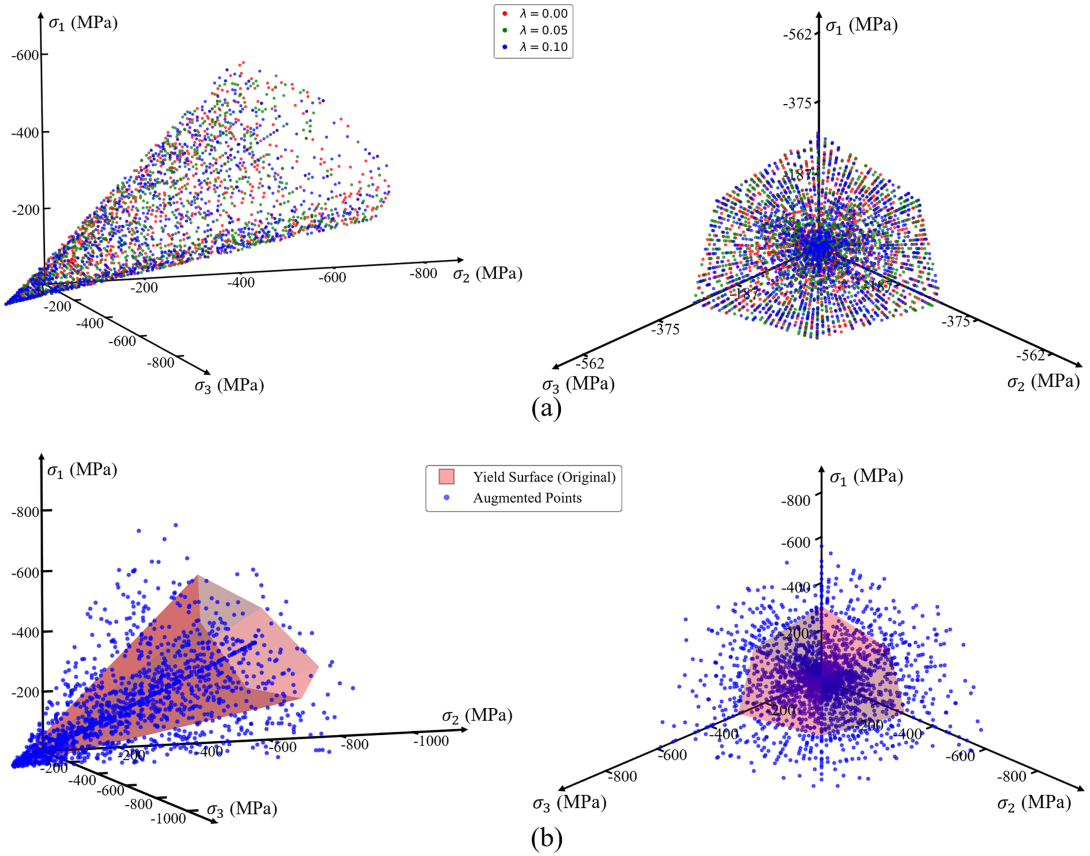

Figure 4: (Continued)

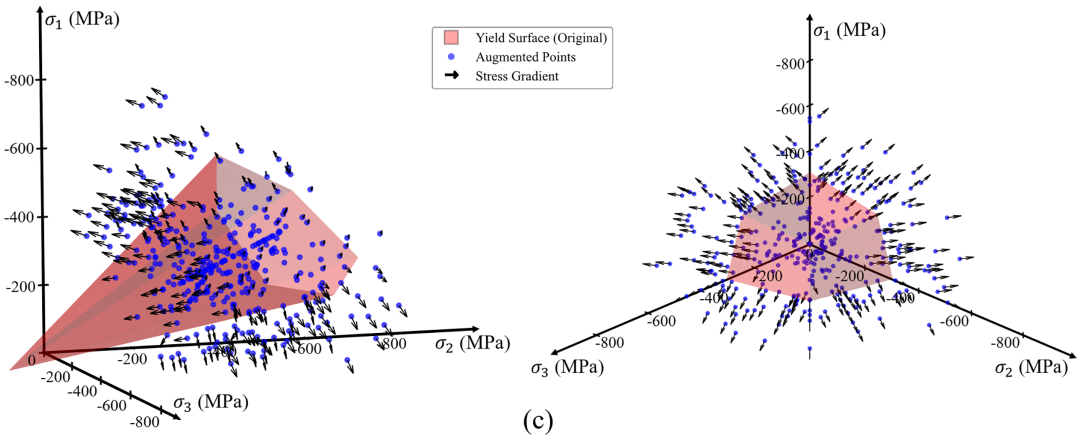

Figure 4: Data preparation and augmentation for neural network training: (**a**) original dataset, (**b**) augmented dataset, and (**c**) representative gradients of augmented samples. All stress components are plotted using the tension-positive convention; negative values correspond to compressive stress states.

In this study, the target stress gradient is obtained analytically from the Mohr–Coulomb model rather than by numerical differentiation of the discrete signed-distance labels, which is more efficient and avoids additional differentiation noise, especially for a large dataset of approximately 10^6 samples. In invariant form, Eq. (30) can be written as:

$$f = \sigma_m \sin\phi + \sqrt{J_2} K(\theta) - c\cos\phi = 0 \tag{31}$$

where

$$K(\theta) = \cos\theta - \frac{1}{\sqrt{3}} \sin\phi \sin\theta$$

$$\theta = \frac{1}{3} \arcsin\left(-\frac{3\sqrt{3}}{2} \frac{J_3}{J_2^{3/2}} \right)$$

$$\sigma_m = \frac{\sigma_1 + \sigma_2 + \sigma_3}{3}$$

$$J_2 = \frac{1}{6}\left[(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2 \right]$$

$$J_3 = (\sigma_1 - \sigma_m)(\sigma_2 - \sigma_m)(\sigma_3 - \sigma_m)$$

here, f is regarded as a function of σ_m, J_2, and θ, and these three quantities are themselves functions of σ_1, σ_2, and σ_3. By the chain rule, for each principal stress component σ_i $(i = 1, 2, 3)$,

$$\frac{\partial f}{\partial \sigma_i} = \frac{\partial f}{\partial \sigma_m}\frac{\partial \sigma_m}{\partial \sigma_i} + \frac{\partial f}{\partial J_2}\frac{\partial J_2}{\partial \sigma_i} + \frac{\partial f}{\partial \theta}\frac{\partial \theta}{\partial \sigma_i} \tag{32}$$

the full derivation is provided in Appendix S6 in the Supplementary Materials.

The level-set method introduced in Section 2.2.2 is then used to augment the original dataset, leading to the complete training set $\left\{ (p, \rho, \theta, \lambda, f, \frac{\partial f}{\partial \sigma}) \right\}_{N_{train}}$. After normalization, the dataset is used for neural network training. The overall data-preparation procedure is summarized in Appendix S7 in the Supplementary Materials.

3.2 Model Training

Using the classical Mohr–Coulomb model without strain softening as the reference, the proposed neural networks are trained to learn the yield function and its stress gradient. Accordingly, the sampled λ range is retained only to keep a unified input format for possible evolving-yield-surface extensions; no hardening or softening law is activated in this benchmark. The material parameters are taken as cohesion c = 20 MPa, friction angle $\phi = 18^{\circ}$, Young's modulus E = 20,000 MPa, and Poisson's ratio ν = 0.2.

During data generation, 100 sampling points $(N_{\sigma_m} = 100)$ are taken along the mean-stress axis, with the minimum mean stress specified as $\sigma_{m,\min} = -400$ MPa. Twenty points $(N_\theta = 20)$ are sampled along the Lode-angle axis over the range from $-\pi/6$ to $\pi/6$, and twenty points $(N_\lambda = 20)$ are sampled along the accumulated plastic strain axis over the range from 0 to 0.1. Accordingly, the number of samples in the original dataset is $N_{data} = N_\lambda \times 6 \times (N_{\sigma_m} \times N_\theta) = 240{,}000$. These samples are then augmented using the signed distance function introduced in Section 2.2.2. With the number of augmentation layers set to $N_\psi = 10$, the final training dataset used for supervised learning contains $N_{train} = 2{,}400{,}000$ samples. Fig. 4 illustrates the original dataset, the augmented dataset, and representative stress gradients.

The sampling density, the minimum mean stress, and the range of accumulated plastic strain all influence the performance of the trained model in subsequent stress-integration simulations. In particular, if the absolute value of $\sigma_{m,\min}$ is too small, the model may become less accurate under high compressive stress states, whereas insufficient sampling density may reduce the quality of the learned yield surface. These effects are discussed further in Section 4.2.

Training is carried out using the NAdam optimizer [36] together with a ReduceLROnPlateau scheduler. The yield-function network is trained for 3000 epochs with a batch size of 128 and an initial learning rate of 0.0001, whereas the stress-gradient network is trained for 5000 epochs with a batch size of 256 and an initial learning rate of 0.001. Thus, the learning rate is adjusted during training rather than kept as a large fixed value. Both networks use 100 hidden neurons per layer. Fig. 5a–c reports the loss histories of the yield function network trained with Sobolev: (a) shows the value loss, (b) the gradient loss, and (c) their sum as the total loss. Fig. 5d shows the loss history of the stress-gradient network. The red curves in Fig. 5 denote raw, unsmoothed validation losses recorded during training rather than final test losses. No exponential moving average or moving-average smoothing is applied. Their stronger fluctuations mainly result from raw epoch-wise evaluation, the sensitivity of derivative-related Sobolev loss terms to local validation samples, and the logarithmic scale at very small loss levels.

3.3 Validation

To evaluate the predictive performance of the trained neural networks, triaxial compression simulations were carried out at the material-point level under different confining pressures. Axial strain increments were prescribed step by step, while the lateral stresses were maintained at the target confining pressure through iterative adjustment of the lateral strain increments. Based on the network predictions, the stress state was updated incrementally along the prescribed loading path until the current step satisfied the required loading condition. Repeating this procedure over all loading increments yielded the complete triaxial stress–strain response under a given confining pressure.

Fig. 6 compares the analytical solution and the neural-network prediction of triaxial compression responses under confining pressures of 0, 5, and 10 MPa. The vertical axis represents the deviatoric stress $\sigma_1 - \sigma_3$, while the left and right branches correspond to the lateral and axial strain responses, respectively. For all confining pressures, the neural-network results agree closely with the analytical curves, including both the initial loading stage and the post-yield response. This agreement indicates that the trained networks

reproduce the pressure-dependent yield characteristics and the associated constitutive evolution along the examined monotonic triaxial compression paths. The present validation should therefore be regarded as a benchmark assessment of the neural stress-integration procedure, rather than an exhaustive validation under arbitrary loading histories.

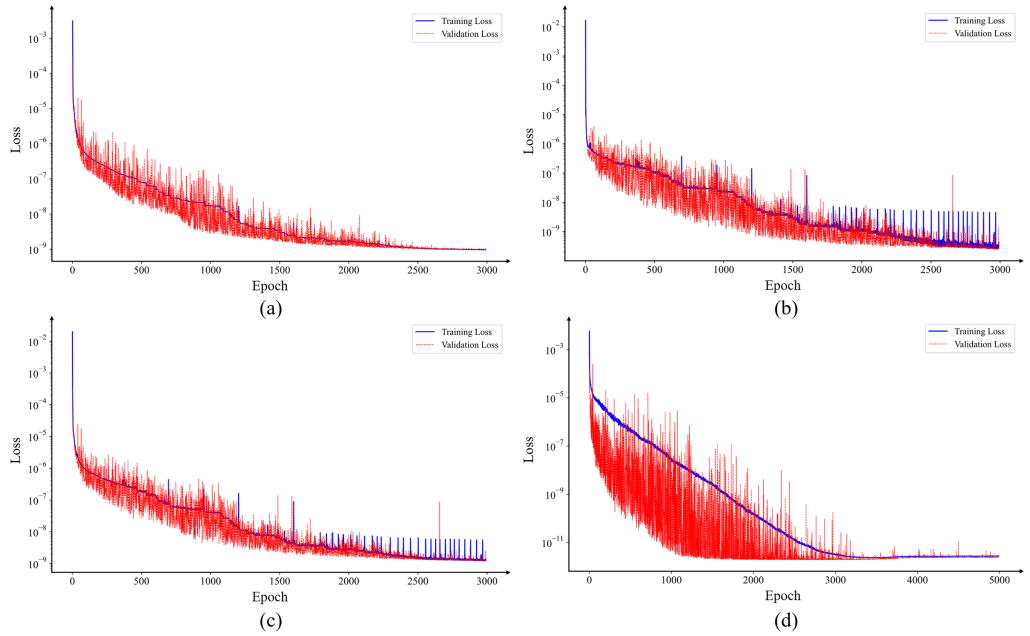

Figure 5: Training histories of the two neural networks: (**a**–**c**) yield-function network and (**d**) stress-gradient network.

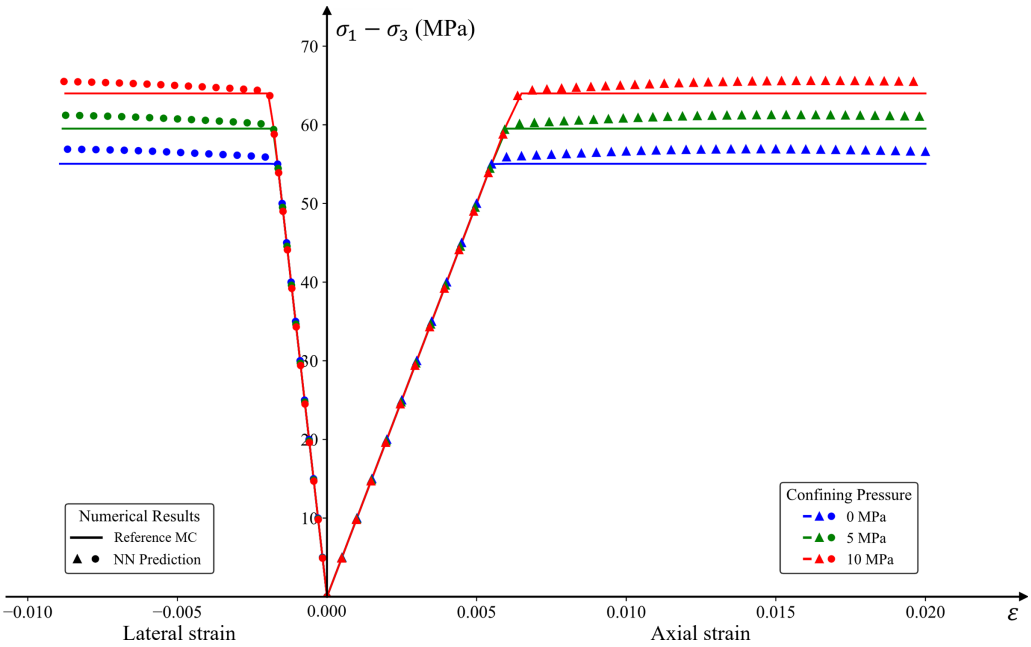

Figure 6: Triaxial compression response: reference MC vs. NN at different confining pressures.

4 Discussion

This section compares the computational characteristics of CPA and CPPM, and investigates the effects of dataset range and network architecture, including depth, width, and Multiply layers, on method performance. Recent graph-network-based learnable physics engines have shown accurate and efficient forward prediction for OSB-PD Drucker–Prager elastoplastic boundary-value problems. Those studies mainly assess solver-level field prediction, whereas the present work focuses on material-point stress integration for a learned Mohr–Coulomb-type yield function. Therefore, the accuracy, computational efficiency, and convergence discussed below should be interpreted at the local stress-update level [16]. Robustness is evaluated using a material-point uniaxial compression path implemented in Python with axial strain increasing incrementally from 0.1% to 40%. The performance index is the maximum sustainable axial strain, defined as the first increment at which the constitutive update fails to converge or the predicted response becomes nonphysical. This metric provides a simple and consistent measure of numerical robustness.

4.1 Comparison of Deep Learning-Based CPA and CPPM

To compare the computational cost of a single stress update in the neural-network-based CPA and CPPM, the number of floating-point operations (FLOPs) is used as a complexity metric [37]. The CPPM considered here is the finite-difference implementation used in the present two-network framework. It should therefore be distinguished from an autograd-based CPPM implementation in which Hessian-related terms are computed directly from the yield-function network. For the yield-function network, one forward evaluation can be approximated as $C_f \approx 2\sum_{k=1}^{L} n_{k-1}n_k$ FLOPs. The gradient network has the same order of complexity, denoted by $C_{\nabla f}$. Since the remaining vector and matrix operations are minor compared with network inference, the per-step cost is estimated mainly from the number of network calls.

In CPA, the yield-function network is first evaluated at the trial state. If $f_{trial} > 0$, a cutting-plane correction is performed. Each local correction iteration requires one call to the stress-gradient network and, if necessary, one additional call to the yield-function network to check the updated state. The per-step complexity is therefore

$$C_{\mathrm{CPA}} \approx C_f + N_{\mathrm{correct}}\left(C_{\nabla f} + C_f\right) \tag{33}$$

when only one local correction iteration is needed $N_{correct} = 1$, this reduces to $C_{CPA} \approx 2C_f + C_{\nabla f}$.

By contrast, CPPM requires additional evaluations to approximate the second-order term. After the trial-state check, a local Newton iteration is performed to enforce the consistency condition. In each iteration, the yield-function network is evaluated for the residual, and the stress-gradient network is called for the return direction. To obtain the second-order term $\partial \mathbf{a}/\partial \boldsymbol{\sigma} = \partial^2 f/\partial \boldsymbol{\sigma}^2$, the forward-difference scheme is used to perturb the first-order stress gradient in several independent directions. The per-step complexity is thus written as

$$C_{\mathrm{CPPM}} \approx C_f + N_{\mathrm{iter}}\left(C_f + 4C_{\nabla f}\right) \tag{34}$$

with a single Newton iteration, this reduces to $C_{CPPM} \approx 2C_f + 4C_{\nabla f}$. The higher cost mainly arises from the approximation of the second-order term.

To complement the theoretical complexity analysis, the computational efficiency of CPA and the finite-difference CPPM implementation was further evaluated in the triaxial-compression benchmark implemented in Python, while keeping the material parameters, loading path, number of loading steps, trained networks, and stopping tolerances unchanged and varying only the stress-update scheme. The timing test was conducted on a laptop computer equipped with an AMD Ryzen 9 6900HX CPU and 32 GB RAM,

using Python 3.8 in CPU-only mode. Although GPU acceleration was used during neural-network training, the GPU was disabled during this stress-integration benchmark.

Fig. 7 shows that CPA requires about 2.0 s of wall-clock time and 0.8 s of CPU user time, whereas the finite-difference CPPM implementation requires about 4.0 and 3.2 s, respectively. These results agree with the call-count analysis for the present implementation. However, an autograd-based CPPM implementation may reduce the cost of the second-order terms; therefore, the reported speed difference should not be interpreted as a general advantage of CPA over all CPPM variants. A related issue is the consistency between the predicted yield function and the return direction. A single yield-function network with autograd gradients would provide exact consistency with the predicted yield function, whereas the present two-network design treats the stress-gradient network as a forward-inference approximation to the analytical reference gradient; a quantitative comparison between these two formulations is left for future work.

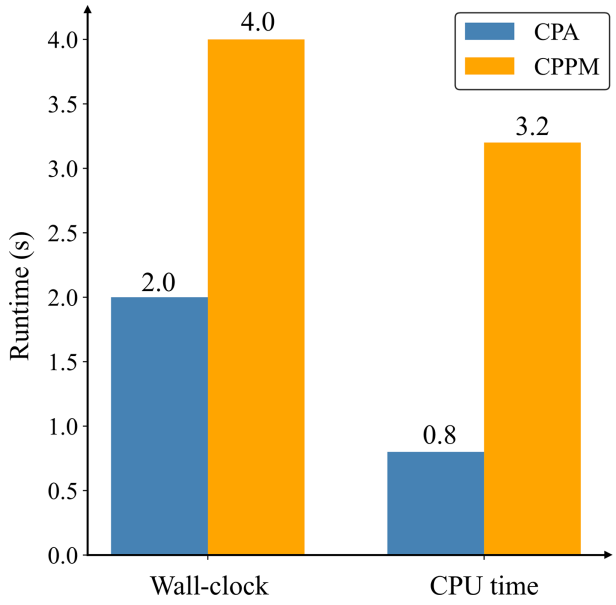

Figure 7: Runtime comparison of CPA and CPPM in the triaxial-compression benchmark implemented in Python.

It should also be noted that the purpose of this comparison is not to show that the neural implementation is faster than an analytical Mohr–Coulomb return mapping. For this simple reference model, an analytical return mapping is expected to be computationally cheaper; the Mohr–Coulomb model is used here as a transparent benchmark for evaluating the learned yield-function representation and the neural stress-update procedure.

4.2 Influence of Dataset Range

The approximation domain of the neural network is limited by the range of the training data. During dataset generation, this range is controlled by the sampling densities N_{σ_m}, N_θ, and N_λ, together with the prescribed bounds of λ, σ_m, and θ. Although neural networks with sufficient capacity can approximate continuous functions over a given domain [38], the actual error still depends on both model capacity and data coverage [39].

For isotropic materials, performance depends more strongly on the coverage of the original dataset than on the absolute number of samples. The most influential bounds are the upper limit of accumulated plastic

strain λ_{max} and the lower limit of mean stress $\sigma_{m,\min}$. Since the effect of θ is less restrictive once the full Lode-angle period is covered, the following discussion focuses on λ_{max} and $\sigma_{m,\min}$.

Their influence is evaluated using a material-point uniaxial compression path implemented in Python. The performance index is the maximum converged axial strain $\varepsilon_{c,\max}$. Two groups of parametric tests are conducted: in the first group, λ_{max} is varied while $\sigma_{m,\min}$ is fixed; in the second group, $\sigma_{m,\min}$ is varied while λ_{max} is fixed. All other settings are kept unchanged. The corresponding parameter ranges are listed in Table 1, and the results are summarized in Fig. 8.

Table 1: Parameter ranges of the original dataset and training settings of the neural networks.

$\lambda_{\min}$	$\lambda_{\max}$	N_λ	$\sigma_{m,\min}$ (MPa)	N_{σ_m}	Epoch $\_$ f_{NN}	Epoch $\_$ df_{NN}	c (MPa)	φ (°)
0	0.1 ~ 1	40	−600	50	200	375	12	33
0	1	20	−100 ~ −1000	100	200	375	10	30

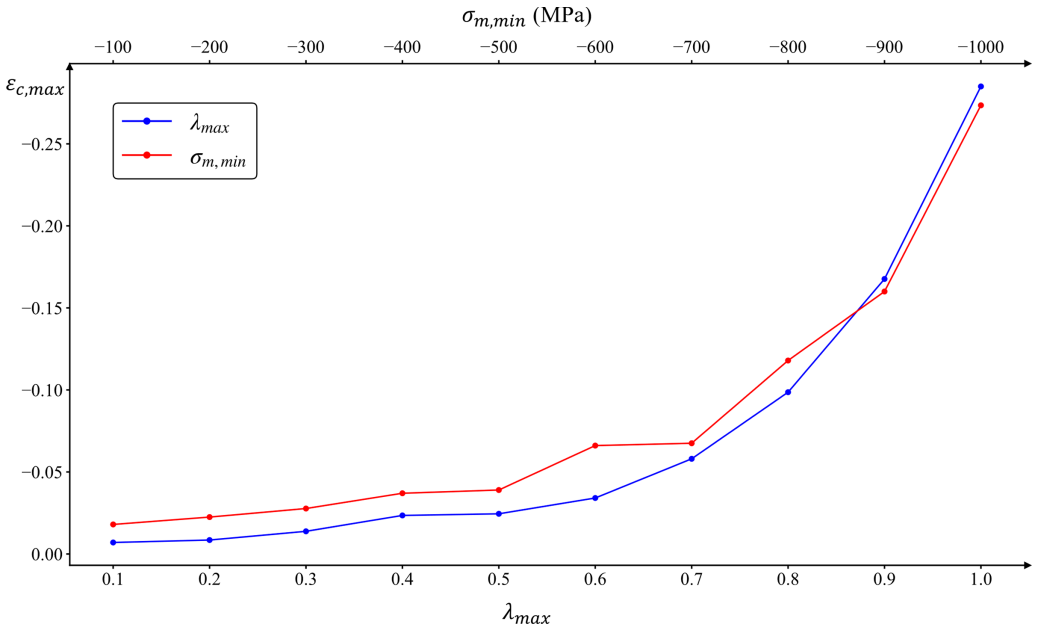

Figure 8: Effect of original dataset range on network performance.

Fig. 8 shows that the attainable axial strain depends on how well the loading path remains inside the original training domain. Increasing λ_{max} enlarges the admissible range of accumulated plastic strain and generally increases $\varepsilon_{c,\max}$ until a plateau is reached. Similarly, making $\sigma_{m,\min}$ more negative expands the training domain toward higher compression, reduces extrapolation in the later stage of loading, and improves robustness. If either bound is too restrictive, the loading path enters poorly covered regions and the constitutive response becomes more fragile. This trend should be interpreted as a training-domain coverage effect rather than as a new constitutive property of the reference model.

In practical applications, the present model should be used only when the training dataset covers the expected ranges of mean stress and accumulated plastic strain along the target loading path, with a reasonable safety margin. Moderate extrapolation may still be tolerated, but both convergence and response quality deteriorate rapidly once the loading path moves too far outside the trained domain. Therefore, the current

framework should be understood as a data-driven constitutive model with a domain-dependent applicability, rather than a fully general extrapolative constitutive model. As a first screening tool for boundary-value simulations, a range-based out-of-distribution (OOD) indicator can be defined for the current input vector $z = (\sigma_m, \rho, \theta, \lambda)$:

$$I_{\mathrm{OOD}}(\mathbf{z}) = \max_i \left[\max \left(\frac{z_i - z_i^{\max}}{z_i^{\max} - z_i^{\min}}, \frac{z_i^{\min} - z_i}{z_i^{\max} - z_i^{\min}}, 0 \right) \right] \tag{35}$$

where z_i is the i-th component of z, and z_i^{min} and z_i^{max} are the lower and upper bounds of this component in the training dataset. $I_{\mathrm{OOD}} = 0$ indicates that the current input remains within the coordinate-wise training bounds, whereas $I_{\mathrm{OOD}} > 0$ indicates that at least one component has moved outside the trained range. This indicator is only a first screening measure; more refined OOD detection may require density estimation, nearest-neighbor distance, or uncertainty-based methods.

4.3 Influence of Network Architecture

Network architecture affects constitutive performance. Increasing depth and width improves representational capacity but may also increase training difficulty. ReLU is suitable for cornered yield surfaces such as Mohr–Coulomb, while the Multiply block enhances nonlinear coupling with limited added complexity.

To examine architectural effects, three designs were compared by varying depth, width, and the use of the Multiply block. Fig. 9 shows two variants relative to the baseline architecture in Fig. 2: in Fig. 9a, one Dense layer and one Multiply block are added to both the yield-function and stress-gradient networks, whereas Fig. 9b shows a Dense-only architecture without Multiply layers. All other hyperparameters were fixed, and hidden-layer widths of 50, 100, and 150 were tested; the settings are listed in Table 2.

Table 2: Hyperparameter settings for training different neural network architectures.

$\lambda_{\min}$	$\lambda_{\max}$	N_λ	$\sigma_{m,\min}$ (MPa)	N_{σ_m}	Epoch $\_$ f_{NN}	Epoch $\_$ df_{NN}	c (MPa)	φ (°)
0	1	20	−800	80	200	375	10	30

Performance is evaluated by the maximum converged axial strain $\varepsilon_{c,\max}$ in a material-point uniaxial compression path implemented in Python. Nine architectures are tested in total, corresponding to three designs and three widths, and each configuration is repeated three times to account for randomness in weight initialization. For each configuration j, the three repeated values $\{\varepsilon_{c,max}^{(j,1)}, \varepsilon_{c,max}^{(j,2)}, \varepsilon_{c,max}^{(j,3)}\}$ are processed using Eq. (36) to compute the arithmetic mean $\bar{\varepsilon}_{c,max}^{(j)}$ and the min–max range $\mathcal{R}_\varepsilon^{(j)}$. Because only three repetitions are used, the range is reported as a descriptive measure of run-to-run variability rather than estimating a coefficient of variation. The results are summarized in Table 3. Here, NoMul denotes the Dense-only architecture in Fig. 9b, Orig denotes the baseline architecture in Fig. 2, and Add denotes the enhanced architecture in Fig. 9a. The symbol d represents the hidden-layer width, and the remaining symbols follow Eq. (36).

$$\bar{\varepsilon}_{c,\max}^{(j)} = \frac{1}{3}\sum_{k=1}^{3} \varepsilon_{c,\max}^{(j,k)}, \quad \mathcal{R}_\varepsilon^{(j)} = \left[\min_{k=1,2,3} \varepsilon_{c,\max}^{(j,k)}, \max_{k=1,2,3} \varepsilon_{c,\max}^{(j,k)} \right] \tag{36}$$

where j denotes the network configuration, k denotes the repeated run with different random initialization, $\bar{\varepsilon}_{c,max}^{(j)}$ is the mean value of the three runs, and $\mathcal{R}_\varepsilon^{(j)}$ is the corresponding min–max range.

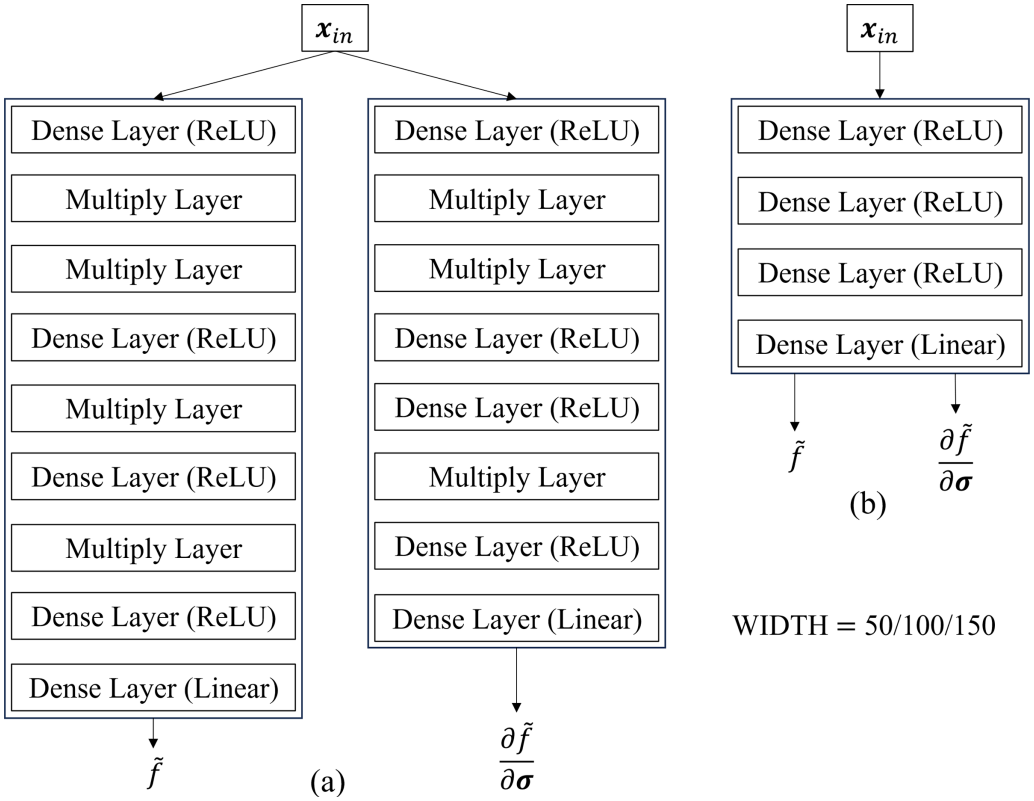

Figure 9: Comparison of neural network architectures: (**a**) Dense + Multiply and (**b**) Dense only.

Table 3: Performance summary of different neural network architectures.

Architecture	d	$\bar{\varepsilon}_{c,\max}$ $(\times 10^{-2})$	$\varepsilon_{c,\max}$ Min–Max Range $(\times 10^{-2})$
NoMul	50	0.69	0.66–0.72
	100	2.17	2.07–2.27
	150	3.33	3.20–3.46
Orig	50	0.96	0.94–0.98
	100	2.27	2.19–2.35
	150	9.58	9.20–9.96
Add	50	2.42	2.36–2.48
	100	3.3	3.17–3.43
	150	6.9	6.77–7.03

Because only three repetitions are available for each configuration, the run-to-run variability is reported using the min–max range rather than the coefficient of variation. The ranges are used only as a descriptive measure of run-to-run variability under different random initializations. At the tested widths, the baseline architecture gives higher mean $\varepsilon_{c,max}$ than the Dense-only architecture, suggesting that the Multiply layer improves stress-update robustness in this benchmark. Based on the mean values in Table 3, the relative increase is defined as

$$R_{\text{orig/nomul}}(d) = \frac{\bar{\varepsilon}_{\max}^{\text{orig},d}}{\bar{\varepsilon}_{\max}^{\text{nomul},d}} - 1 \tag{37}$$

yielding $R(50) = 38.0\%$, $R(100) = 4.6\%$, and $R(150) = 187.5\%$. The improvement is most pronounced at $d = 150$, where the baseline architecture reaches 2.88 times the load-bearing capacity of the Dense-only network.

The enhanced architecture gives higher mean $\varepsilon_{c,max}$ than the baseline at $d = 50$ and $d = 100$, but not at $d = 150$. This non-monotonic behavior indicates that adding a Dense + Multiply block does not guarantee improved stress-update robustness under fixed training settings. The lower result of the Add architecture at $d = 150$ may be related to optimization sensitivity and random initialization, rather than to a clear representational advantage or disadvantage. Fig. 10 summarizes these trends.

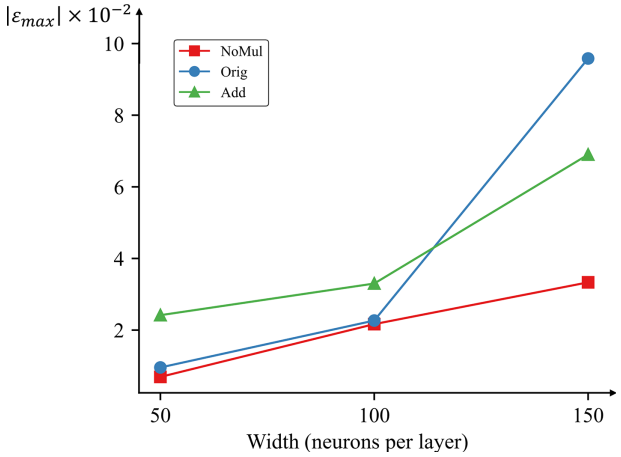

Figure 10: Depth-performance curves of three neural constitutive model architectures.

Overall, increasing width tends to improve the mean performance for the tested architectures, while the benefit of additional Multiply layers is architecture- and training-setting dependent. For relatively compact networks, a small Dense + Multiply unit offers an efficient way to improve nonlinear representation without substantially increasing architectural complexity.

5 Conclusions

This study presents a deep-learning-based constitutive method for geomaterials, in which two compact neural networks are trained separately to represent the yield function and its stress gradient. Based on these predictions, a CPA-based neural stress-update procedure is constructed for stress integration. Following the classical first-order character of CPA, the proposed strategy avoids explicit evaluation of second-order derivatives and is more naturally compatible with pressure-sensitive and non-smooth yield criteria such as Mohr–Coulomb in a first-order return-mapping sense.

The results show that the proposed method can reproduce the response of the reference model along the examined monotonic triaxial compression paths and achieve lower computational cost than the finite-difference CPPM implementation considered in this study. The study also indicates that robustness depends strongly on the coverage of the training dataset and the choice of network architecture. Accordingly, the present method is most suitable for applications in which the relevant stress range and history-variable range can be estimated in advance and adequately represented in the training dataset. In particular, appropriate

ranges of mean stress and accumulated plastic strain are important for stable predictions, while the benefit of Multiply layers is architecture- and training-setting dependent in the tested benchmark.

Future work will further examine unloading–reloading paths, stress paths traversing multiple deviatoric-plane sectors, true triaxial paths with varying Lode angle, and apex-region paths. Extensions to anisotropy, cyclic loading, evolving hardening/softening surfaces, and stronger physically informed constraints, including convexity-preserving and thermodynamic constraints, will also be considered.

Acknowledgement: None.

Funding Statement: This research was funded by the Postgraduate Research & Practice Innovation Program of Jiangsu Province, grant number SJCX25_0268 (Zijie He).

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Qingxiang Meng and Zijie He; methodology, Zijie He and Qingxiang Meng; software, Zijie He; validation, Zijie He, Yajun Cao and Weijiang Chu; formal analysis, Zijie He; investigation, Zijie He; resources, Qingxiang Meng; data curation, Zijie He; writing—original draft preparation, Zijie He; writing—review and editing, Qingxiang Meng, Yajun Cao and Weijiang Chu; visualization, Zijie He; supervision, Qingxiang Meng; project administration, Qingxiang Meng; funding acquisition, Zijie He. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data supporting the findings of this study are available within the article and its Supplementary Materials. The implementation scripts used for data generation, neural-network training, and material-point stress-update calculations are not released as a public software package at this stage. The implementation scripts, trained network parameters, normalization parameters, and example input files are available from the corresponding author upon reasonable request.

Ethics Approval: Not applicable.

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

Supplementary Materials: The supplementary material is available online at https://www.techscience.com/doi/10.32604/cmes.2026.083227/s1. These materials include the coordinate transformation between cylindrical coordinates and principal stresses (Appendix S1), pseudocode for neural network inference (Appendix S2), the forward-difference scheme for second-order approximation in CPPM (Appendix S3), the complete CPPM stress-update procedure (Appendix S4), the complete CPA stress-update procedure (Appendix S5), the analytical derivation of the Mohr–Coulomb stress gradient (Appendix S6), and the overall data-preparation procedure (Appendix S7).

Abbreviations

CPA	Cutting Plane Algorithm
CPPM	Closest-Point Projection Method
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
NN	Neural Network

References

1. Armero F, Pérez-Foguet A. On the formulation of closest-point projection algorithms in elastoplasticity—Part I: the variational structure. Int J Numer Methods Eng. 2002;53(2):297–329. doi:10.1002/nme.278.
2. Piccolroaz A, Bigoni D. Yield criteria for quasibrittle and frictional materials: a generalization to surfaces with corners. Int J Solids Struct. 2009;46(20):3587–96.
3. Wood DM. Geotechnical modelling. Boca Raton, FL, USA: CRC Press; 2017.

4. Ghaboussi J, Sidarta DE. New nested adaptive neural networks (NANN) for constitutive modeling. Comput Geotech. 1998;22(1):29–52. doi:10.1016/s0266-352x(97)00034-7.
5. Kalidindi SR, Niezgoda SR, Salem AA. Microstructure informatics using higher-order statistics and efficient data-mining protocols. Jom. 2011;63(4):34–41. doi:10.1007/s11837-011-0057-7.
6. Mozaffar M, Bostanabad R, Chen W, Ehmann K, Cao J, Bessa M. Deep learning predicts path-dependent plasticity. Proc Natl Acad Sci. 2019;116(52):26414–20. doi:10.1073/pnas.1911815116.
7. Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Netw. 1989;2(5):359–66. doi:10.1016/0893-6080(89)90020-8.
8. Linka K, Hillgärtner M, Abdolazizi KP, Aydin RC, Itskov M, Cyron CJ. Constitutive artificial neural networks: a fast and general approach to predictive data-driven constitutive modeling by deep learning. J Comput Phys. 2021;429:110010.
9. Holthusen H, Lamm L, Brepols T, Reese S, Kuhl E. Theory and implementation of inelastic constitutive artificial neural networks. Comput Methods Appl Mech Eng. 2024;428(3):117063. doi:10.1016/j.cma.2024.117063.
10. Boes B, Simon J-W, Holthusen H. Accounting for plasticity: an extension of inelastic constitutive artificial neural networks. Eur J Mech-A/Solids. 2025;117:105998.
11. Masi F, Stefanou I. Multiscale modeling of inelastic materials with Thermodynamics-based Artificial Neural Networks (TANN). Comput Methods Appl Mech Eng. 2022;398(7):115190. doi:10.1016/j.cma.2022.115190.
12. Vlassis NN, Sun W. Component-based machine learning paradigm for discovering rate-dependent and pressure-sensitive level-set plasticity models. J Appl Mech. 2022;89(2):021003. doi:10.1115/1.4052684.
13. Suh HS, Kweon C, Lester B, Kramer S, Sun W. A publicly available PyTorch-ABAQUS UMAT deep-learning framework for level-set plasticity. Mech Mater. 2023;184(2):104682. doi:10.1016/j.mechmat.2023.104682.
14. Sheil B, Anagnostopoulos C, Buckley R, Ciantia MO, Febrianto E, Fu JL, et al. Artificial intelligence transformations in geotechnics: progress, challenges and future enablers. Comput Geotech. 2026;189:107604.
15. Su M, Guo N, Yang Z. A multifidelity neural network (MFNN) for constitutive modeling of complex soil behaviors. Int J Numer Anal Methods Geomech. 2023;47(18):3269–89. doi:10.1002/nag.3620.
16. Zhou X-P, Feng K. The novel learnable physics engines for interpretable elastoplastic models of geomaterials based on the message passing neural network. Int J Rock Mech Min. 2025;194(2):106244. doi:10.1016/j.ijrmms.2025.106244.
17. Lubliner J. Plasticity theory. North Chelmsford, MA, USA: Courier Corporation; 2008.
18. Huang J, Griffiths D. Observations on return mapping algorithms for piecewise linear yield criteria. Int J Geomech. 2008;8(4):253–65. doi:10.1061/(asce)1532-3641(2008)8:4(253).
19. Mazzucco G, Pomaro B, Salomoni VA, Majorana CE. Apex control within an elasto-plastic constitutive model for confined concretes. Math Comput Simul. 2019;162:221–32.
20. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, et al. Automatic differentiation in pytorch. 2017.
21. Bishop CM, Nasrabadi NM. Pattern recognition and machine learning. Berlin/Heidelberg, Germany: Springer; 2006.
22. Liaw R, Liang E, Nishihara R, Moritz P, Gonzalez JE, Stoica I. Tune: a research platform for distributed model selection and training. arXiv:1807.05118. 2018.
23. Borja RI. Plasticity. Berlin/Heidelberg, Germany: Springer; 2013.
24. Osher S, Sethian JA. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. J Comput Phys. 1988;79(1):12–49.
25. Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning. Cambridge, UK: MIT Press; 2016.
26. Czarnecki WM, Osindero S, Jaderberg M, Swirszcz G, Pascanu R. Sobolev training for neural networks. Adv Neural Inf Process Syst. 2017;30:1–10.
27. Feng K, Zhou X-P. A novel graph networks based learnable physics engines for crack propagation and coalescence in solid mechanics. Eng Fract Mech. 2025;315:110800. doi:10.1016/j.engfracmech.2025.110800.
28. Maclaurin D, Duvenaud D, Adams RP. Autograd: effortless gradients in numpy. In: Proceedings of the ICML, 2015 AutoML Workshop; 2015 Jul 11; Lille, France.

29. Simo J, Hughes T. Classical rate-independent plasticity and viscoplasticity. Comput Inelasticit. 1998:1–112. doi:10.1007/0-387-22763-6_2.
30. Simo JC, Taylor RL. Consistent tangent operators for rate-independent elastoplasticity. Comput Methods Appl Mech Eng. 1985;48(1):101–18.
31. Huang J, Griffiths D. Return mapping algorithms and stress predictors for failure analysis in geomechanics. J Eng Mech. 2009;135(4):276–84.
32. Coulomb C-A. Essai sur une application des règles de maximis et minimis à quelques problèmes de statique, relatifs à l'architecture. Mémoires De Mathématique De L'académie R De Sci. 2023;175:1. doi:10.1051/geotech/2023019.
33. Mohr O. Abhandlungen aus dem Gebiete der technischen Mechanik. Berlin, Germany: Ernst & Sohn; 1914.
34. Abbo A, Sloan S. A smooth hyperbolic approximation to the Mohr-Coulomb yield criterion. Comput Struct. 1995;54(3):427–41. doi:10.1016/0045-7949(94)00339-5.
35. Nayak GC, Zienkiewicz OC. Convenient form of stress invariants for plasticity. J Struct Div. 1972;98(4):949–54. doi:10.1061/jsdeag.0003219.
36. Dozat T. Incorporating nesterov momentum into adam. 2016.
37. Hennessy JL, Patterson DA. Computer architecture: a quantitative approach. Amsterdam, The Netherlands: Elsevier; 2011.
38. Barron AR. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inf Theory. 2002;39(3):930–45. doi:10.1109/18.256500.
39. Anthony M, Bartlett PL. Neural network learning: theoretical foundations. Cambridge, UK: Cambridge university Press; 2009.