UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

A Three-Dimensional Multiscale Finite Element Framework for Nonlinear Composite Materials Based on Deep Learning

Permalink

https://escholarship.org/uc/item/3zs4x50p

Author

Zhou, Yijing

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

A Three-Dimensional Multiscale Finite Element Framework for Nonlinear Composite Materials Based on Deep Learning

A Thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Structural Engineering

by

Yijing Zhou

Committee in charge:

Professor Shabnam J. Semnani, Chair
Professor Jiun-Shyan Chen
Professor Petr Krysl

2024

Copyright

Yijing Zhou, 2024

All rights reserved.

The Thesis of Yijing Zhou is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

TABLE OF CONTENTS

Thesis Approval Page iii

Table of Contents iv

List of Figures vi

List of Tables x

Acknowledgements xi

Vita xii

Abstract of the Thesis xiii

Chapter 1 Introduction 1

Chapter 2 Literature Review 3

Chapter 3 RVE Modeling 7
3.1 Boundary Conditions of RVE Model 7
3.1.1 Periodic Boundary Conditions 7
3.1.2 Other Boundary Conditions 8
3.2 Material Properties 9
3.3 RVE Modeling 10

Chapter 4 Homogenization 15

Chapter 5 Machine Learning 17
5.1 RNN Models 17
5.1.1 Architecture of RNN Model 17
5.1.2 Architecture of GRU Model 18
5.2 Data Generation and Sampling 21
5.3 Training 24

Chapter 6 Multiscale Simulation 27
6.1 Multiscale Methods 27
6.2 Implementation of Trained GRU in UMAT 28

Chapter 7 Result and Discussion 32
7.1 Test the Performance of New Sampling Algorithm 32
7.2 Test in Microscale Problems 34
7.2.1 Test in Testing Dataset 35
7.2.2 Test in Triaxial Stress State 35
7.3 Test in Multiscale Problems 45

7.3.1 Cantilever Beam Tensile Test 46
7.3.2 Cantilever Beam Bending Test 49
7.3.3 Coupon Tensile Test 53
7.4 Test in Complex Multiscale Problems 57
7.4.1 Cyclic Surface Traction Force Test 60
7.4.2 Cyclic Surface Displacement Test 64
7.4.3 Sinusoidal Surface Traction Force Test 67
7.5 Testing in a 2D Problem 71

Chapter 8 Expanding Research 76
8.1 Transformer Model 76
8.1.1 Attention Mechanism 76
8.1.2 Transformer Models 81
8.2 Training Strategies 84
8.2.1 Random Extract Training Algorithm 84
8.2.2 Learning Rate Adjustment Strategy 88
8.3 Performance of Transformer Surrogate Model 88
8.3.1 Performance of Random Extract Training Algorithm 90
8.3.2 Comparison Between Transformer and GRU 92

Chapter 9 Concluding Remarks 97

Bibliography 99

LIST OF FIGURES

Figure 1.1. The flowchart of our multiscale framework. 2

Figure 3.1. An RVE model generated by our preprocess script. 11

Figure 5.1. The structure of GRU. 19

Figure 5.2. The structure of our GRU based surrogate model. 20

Figure 5.3. An example of a strain series generated by our strain sampling algorithm. . 22

Figure 5.4. Training and testing loss curves. 26

Figure 7.1. Comparison of relative errors for our new sampling algorithm and the reference algorithms. 34

Figure 7.2. The true value and prediction versus time steps of an example of our testing dataset. 36

Figure 7.3. Four triaxial strain states we used to test our model (In (a), (b), and (d), $E_{11} = E_{22}$, $E_{12} = E_{13} = E_{23} = 0$; in (c), $E_{11} = E_{22}$). 37

Figure 7.4. The predicted and true stress versus time steps of monotonic triaxial strain state. The strain state is shown in Figure 7.3a. 38

Figure 7.5. The predicted and true stress versus time steps of cyclic triaxial strain state. The strain state is shown in Figure 7.3b. 39

Figure 7.6. The predicted and true stress versus strain of cyclic triaxial strain state. The strain state is shown in Figure 7.3b. 40

Figure 7.7. The predicted and true stress versus time steps of cyclic triaxial strain state with nonzero shear strain. The strain state is shown in Figure 7.3c. 41

Figure 7.8. The predicted and true stress versus strain of cyclic triaxial strain state with nonzero shear strain. The strain state is shown in Figure 7.3c. 42

Figure 7.9. The predicted and true stress versus time steps of sinusoidal cyclic triaxial strain state. The strain state is shown in Figure 7.3d. 44

Figure 7.10. The predicted and true stress versus strain of sinusoidal cyclic triaxial strain state with nonzero shear stress. The strain state is shown in Figure 7.3d. . . 44

Figure 7.11. The geometry and boundary condition of the cantilever beam in the tensile test ($l/L = 0.1$). 47

Figure 7.12. The comparison of displacement fields of high-fidelity and multiscale simulation in a tensile test of a cantilever beam. 48

Figure 7.13. The maximum displacement in the z-direction versus the load applied to the cantilever beam (tensile). 49

Figure 7.14. The true value and stress prediction at an integration point versus time steps in a cantilever beam (tensile). 50

Figure 7.15. The boundary condition of the cantilever beam bending test ($l/L = 0.25$). . 50

Figure 7.16. The comparison of displacement fields of high-fidelity and multiscale simulation in a bending test of a cantilever beam. 52

Figure 7.17. The von Mises stress and strain fields of multiscale simulation in a bending test of a cantilever beam. 52

Figure 7.18. The maximum displacement in the y-direction versus the load applied to the cantilever beam (bending). 53

Figure 7.19. The true value and stress prediction at an integration point versus time steps in a cantilever beam (bending). 54

Figure 7.20. The boundary condition of the coupon tensile test ($l/L = 0.25$). 55

Figure 7.21. The comparison of displacement fields of high-fidelity and multiscale simulation of a tensile test of a coupon. 56

Figure 7.22. The von Mises stress and strain fields in multiscale simulation of a tensile test of a coupon. 56

Figure 7.23. The maximum displacement in the z-direction versus the traction applied to the coupon. 57

Figure 7.24. The true value and stress prediction at an integration point versus time steps of a coupon. 58

Figure 7.25. Geometry of thick plate with a circular perforation. 59

Figure 7.26. The mesh of multiscale simulations, along with the elements and nodes of interest in the thick plate model. 60

Figure 7.27. The comparison of the displacement distribution in y-direction of high-fidelity and multiscale simulation in the 75th time step (cyclic surface traction force). 62

Figure 7.28. The comparison of the displacement distribution in the z-direction of high-fidelity and multiscale simulation in the 75th time step (cyclic surface traction force). 63

Figure 7.29. The von Mises stress and strain distribution of multiscale simulation in the 75th time step (cyclic surface traction force). 63

Figure 7.30. The displacement in y-direction of node “a” and the displacement in z-direction of node “b” versus time step (cyclic surface traction force). 64

Figure 7.31. The predicted and true stress of integration point 7 of element “C” versus time steps (cyclic surface traction force). 65

Figure 7.32. The von Mises stress and strain distribution of multiscale simulation in the 75th time step (cyclic surface displacement). 66

Figure 7.33. The predicted and true reaction force in the upper surface of the model versus cyclic surface displacement. 67

Figure 7.34. The predicted and true stress at integration point 8 of element “D” versus time steps (cyclic surface displacement). 68

Figure 7.35. The comparison of the displacement distribution in y-direction of high-fidelity and multiscale simulation in the 75th time step (sinusoidal surface traction force). 69

Figure 7.36. The comparison of the displacement distribution in z-direction of high-fidelity and multiscale simulation in the 75th time step (sinusoidal surface traction force). 70

Figure 7.37. The von Mises stress and strain distribution of multiscale simulation in the 75th time step (sinusoidal surface traction force). 70

Figure 7.38. The displacement in y-direction of node “a” and the displacement in z-direction of node “b” versus time step (sinusoidal surface traction force). . 71

Figure 7.39. The predicted and true stress at integration point 7 of element “C” versus time steps (sinusoidal surface traction force). 72

Figure 7.40. The boundary condition of 2D problem ($l/L = 0.1$). 74

Figure 7.41. The predicted and true reaction force on the right surface versus U_1 in the 2D problem. 75

Figure 8.1. Architecture of the transformer model in the original paper [79]. 82

Figure 8.2. Architecture of the transformer surrogate model 1. 85

Figure 8.3. Comparison of the prediction of transformers trained by traditional and random extract training algorithm. 91

Figure 8.4. Comparison of the prediction of transformer and GRU versus time step under triaxial sinusoidal cyclic strain state with the shear strain of 0 (steps 101 to 200). 93

Figure 8.5. Comparison of the prediction of transformer and GRU versus strain under triaxial sinusoidal cyclic strain state with the shear strain of 0 (steps 101 to 200). 94

Figure 8.6. Comparison of the prediction of transformer and GRU versus time step under triaxial sinusoidal cyclic strain state with the non-zero shear strain (steps 101 to 200). 95

Figure 8.7. Comparison of the prediction of transformer and GRU versus strain under triaxial sinusoidal cyclic strain state with the non-zero shear strain (steps 101 to 200). 96

LIST OF TABLES

Table 5.1. Hyper-parameters of the GRU model in this work. 21

Table 5.2. Parameters of data generation. 24

Table 5.3. Hyper-parameters of training process. 25

Table 8.1. Comparison of the relative errors of the transformers trained by traditional and RET algorithm. 92

Table 8.2. Comparison of the relative errors of the transformer and GRU models over the entire prediction process and in the latter part of the predictions (triaxial sinusoidal cyclic strain state with the shear strain of 0). 94

Table 8.3. Comparison of the relative errors of the transformer and GRU models over the entire prediction process and in the latter part of the predictions (triaxial sinusoidal cyclic strain state with the non-zero shear strain). 96

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Shabnam J. Semnani for her support as the chair of my committee. She made a contribution to the guidance and revision of the thesis. I would like to express my sincere appreciation for her unwavering guidance, support, and patience throughout my Master's studies. At the same time, I also hope to thank Professor Jiun-Shyan Chen and Professor Petr Krysl for their contribution as defense committee members. I also want to thank my mother for her financial support.

I extend my profound gratitude to the University of California San Diego and the Jacobs School of Engineering for offering a platform for my learning and research.

Chapters 3 to 7, in full, have been submitted for publication of the material, Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this paper.

Chapter 8, in part, is currently being prepared for submission for publication of the material. Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this material.

VITA

2022 Bachelor of Engineering, Hubei University of Education

2024 Master of Science, University of California San Diego

FIELDS OF STUDY

Major Field: Engineering (Specialization or Focused Studies)

Studies in Computational Mechanices
Professor Shabnam J. Semnani

Studies in Machine Learning
Professor Shabnam J. Semnani

ABSTRACT OF THE THESIS

A Three-Dimensional Multiscale Finite Element Framework for Nonlinear Composite Materials Based on Deep Learning

by

Yijing Zhou

Master of Science in Structural Engineering

University of California San Diego, 2024

Professor Shabnam J. Semnani, Chair

Hierarchical multiscale simulations of composite materials face difficulties due to the enormous computational cost they require. Recent research has found that multiscale finite element methods based on homogenization and Recurrent Neural Networks (RNN) have the potential to address this challenge. In this research, we implement a Gated Recurrent Unit (GRU) based multiscale computational mechanics framework for nonlinear composite materials. We propose a Fortran algorithm to reconstruct a trained GRU model in ABAQUS UMAT as the surrogate model and implement it in multiscale finite element simulations. We use our UMAT code with a trained GRU model to solve a collection of 3D and 2D numerical problems

under different load cases and geometry in this thesis to verify the performance of our model in multiscale simulation. We also test the performance of our GRU model at the integration points. Meanwhile, we also propose a novel sampling algorithm based on random increments and selection, which enables the generated strain series to provide more information on the material's plastic stage.

In the expanding research, we design a transformer-based surrogate model to address the issue of GRU performance decline with long sequences and to further improve the accuracy of the surrogate model. Additionally, we propose a random extract training algorithm to enhance the generalization capability of the transformer model. We conducted tests and comparisons of the transformer and GRU models using longer sequences, demonstrating that the transformer surrogate model exhibits superior accuracy and stability in processing longer sequences.

Chapter 1

Introduction

Multiscale simulations of nonlinear composite materials are a critical topic in multiscale computational mechanics since the direct high-fidelity simulation of composites is impractical due to the substantial computational costs. In long continuous fiber composite materials, the diameter of fibers is usually in the micrometer range. Compared to the meter-scale structure, the scale of fibers is usually five to six orders of magnitude smaller. In order to accurately simulate the stress response of composite materials at the microscale, using traditional finite element methods requires an extremely fine mesh to adapt to the microstructure of the fibers and matrix of the composite materials. However, this approach significantly increases the computational costs and demands a large amount of computing resources. In some cases, the immense computational requirements can even render our simulation infeasible. Thus, finding an alternative method to address this challenge is highly significant.

In this work, we implement a multiscale computational mechanics framework for long continuous fiber composite materials based on Recurrent Neural Networks (RNN). The flowchart of our multiscale framework is illustrated in Figure 1.1. In this work, we propose an innovative sampling algorithm based on series selection to generate a macroscale strain dataset, which is used for the offline simulation of the RVE models. We develop a preprocessing code to generate the models of RVE and apply periodic boundary conditions to them. We also design a postprocess and homogenization code to extract data from RVE simulation results and use the

homogenization technique to generate the stress series dataset. In our research, Gated Recurrent Unit (GRU) is used to learn the microscale constitutive relationship of long continuous composite materials. We use the datasets we generated earlier to train and test our GRU model and use the trained GRU model as the surrogate model in multiscale simulation. Additionally, we propose a Fortran algorithm to reconstruct the trained GRU model in ABAQUS UMAT, and we implement the multiscale framework in several 3D and 2D boundary condition problems to verify the effectiveness of our surrogate model.

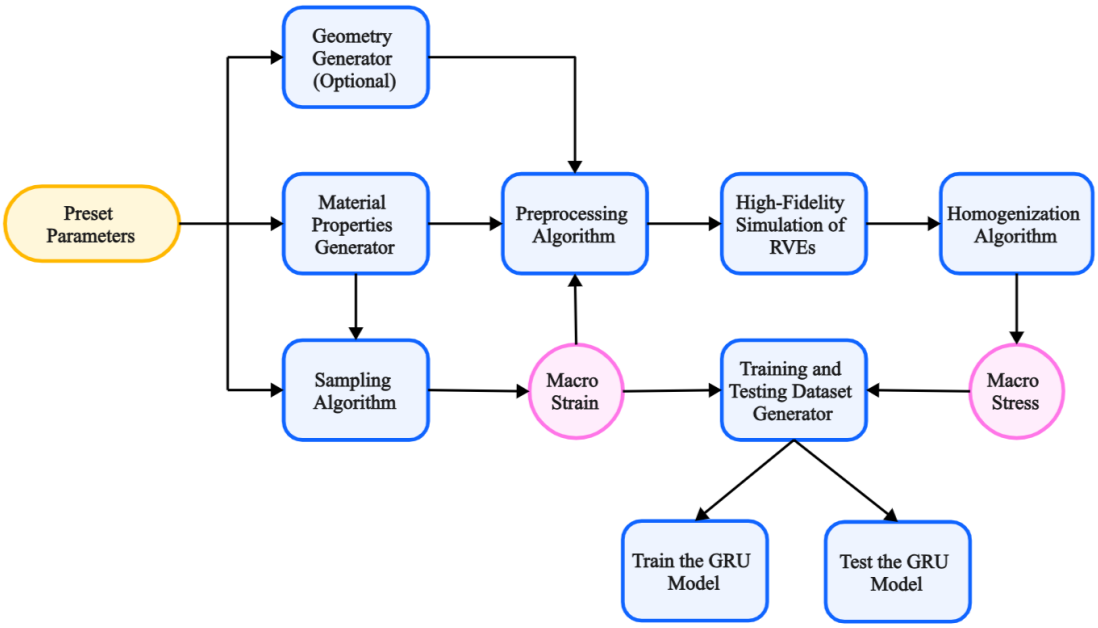


Figure 1.1. The flowchart of our multiscale framework.

Finally, in the chapter on extended research, we design a transformer-based surrogate model to learn the constitutive relationships of nonlinear materials. Our research indicates that the transformer model exhibits higher accuracy and stability than the GRU model in processing long sequences. Additionally, we proposed a novel training algorithm to enhance the transformer's generalization ability across sequences of varying lengths.

Chapter 2

Literature Review

Using traditional methods for direct high-fidelity simulation of composite materials is infeasible due to the enormous computational cost. In order to address this challenge, many scholars have proposed a range of multiscale simulation methods, such as the FE^2 method. The FE^2 method can achieve more efficient multiscale simulation by transmitting microscale information to macroscopic scales [1]. Specifically, this method uses the strain transmitted from the macroscopic scale to perform mechanical simulations on representative volume elements (RVE), homogenizes the simulation results of RVE, and transfers the homogenized results back to the integration points of the macroscopic scale for macroscopic simulation. An RVE is a small volume unit used to represent and describe the microstructure of composite materials, providing geometric information of material microstructure [2, 3]. The advantage of this method is that coarse grids can be used for simulation at the macroscale, while the information we need at the microscale is achieved through the simulation of RVE. The FE^2 method reduces the computational cost required for composite material simulation through clever design, making multiscale simulation feasible. Kanouté et al. review various multiscale methods for composite materials in their paper [4].

However, the computational cost brought by the FE^2 method is also very large, often requiring very large computational resources. The reason is that in this method, we still need to conduct a large number of RVE simulations, which depends on the number of integration

points. The disadvantage is particularly evident for complex structural components because if the geometry of the component itself is complex, we need to use finer mesh for simulation, which rapidly increases the computational cost. In practical engineering applications, components often have very complex geometries, rendering the FE^2 method impractical. In this case, we need to improve this method to further accelerate multiscale simulation.

In order to further reduce computational costs and accelerate computation, machine learning methods are being applied to multiscale simulations. Machine learning models are capable of capturing the nonlinear relationships inherent in datasets, so we can use them to learn the constitutive relationships of materials and use trained models to predict the stress responses of materials. Using trained machine learning models such as artificial neural networks (ANN) as surrogate models can accelerate multiscale simulation, as the prediction process of machine learning models often only requires a few steps of matrix computation, which is much simpler than RVE simulation. Bishara et al. [5], Bock et al. [6], Liu et al. [7], Fish et al. [8], and Liu et al. [9]. summarize the latest advances in machine learning in the field of computational mechanics and the history of finite element method.

There are many machine learning models used in the field of computational mechanics. In chronological terms, artificial neural networks were first commonly used by scholars in this field. Settgast et al. obtained a training dataset by simulating the RVE of porous structures and used the trained model as a surrogate model [10]. Ali et al. [11] and Im et al. [12] used the trained ANN model for the mechanical simulation of crystal structures. Jang et al. used the ANN model as a surrogate model for nonlinear plastic loading of J2 plastic materials [13]. Furukawa and Yagawa [14], Benabou [15], Jung and Ghaboussi [16, 17], and Li [18] applied the ANN model to the field of rate-dependent material behaviors.

There are many machine learning models used to accelerate the FE^2 method. For long continuous fiber composite materials, Recurrent Neural Networks (RNN) is a class of commonly used surrogate models. The reason is that the matrix of composite material is usually plastic material such as aluminum alloy so the stress response is related to the previous step. RNN is a

class of models that has memory ability because the output of RNN depends on input and the hidden status of the previous step, which means RNN can use the information from previous steps. This capability enables RNN to perform effectively in handling step-dependent series, as a result, RNN is widely used as a constitutive surrogate model for plastic materials. Variations of RNN, such as Gated Recurrent Unit (GRU), introduced the gate control mechanism and have improved the structure to better address issues such as gradient vanishing, further enhancing the accuracy of the model. Mozaffar et al. [19], He and Semnani [20], Bonatti and Mohr [21], Zhang et al. [22], Im et al. [23], Heidenreich and Mohr [24], Guan et al. [25], Tancogne et al. [26], and Dettmer et al. [27] used the trained RNN model as a surrogate model for inelastic constitutive models to predict stress. Wu et al. [28], Ghavamian and Simone [29], Logarzo et al. [30], Wang and Sun [31], Haghighi and Na [32], Calleja et al. [33], Wu et al. [34], Kim et al. [35], and Vijayaraghavan et al. [36] applied RNN to learn the path-dependent constitutive relationships in homogenization based multiscale methods. Gorji et al. [37] and Friedmann et al. [38] implemented some recurrent neural networks to predict the stress response of foam materials and short-fiber composite materials, respectively. Chen proposed using RNN and LSTM models to learn the constitutive relationship of viscoelasticity [39]. Tandale et al. [40], Maia et al. [41], Deng et al. [42], Borkowski et al. [43], Qu et al. [44], and Maia et al. [45] proposed some RNN models that combine physical information to improve the performance of the models. Fish and Yu use GRU for a hybrid data-physics driven reduced-order homogenization [46]. Bessa et al. designed some methods to address the preprocessing and postprocessing of RVE and sampling issues [47]. Omairey et al. designed an ABAQUS plugin to generate periodic boundary conditions and homogenize the RVE model [48]. Heidenreich et al. [49] and Qu et al. [50] combine RNN with transfer learning to learn the constitutive relation of plastic materials. Abueidda et al. [43] and Tandale and Stoffel [51] compare the performance of GRU with various other machine learning models.

In recent research, in addition to RNN models, many scholars have also applied convolutional neural network (CNN) models to the simulation of composite materials, which is

often used to find the mapping between the microstructure and stress response of composite materials. Bhaduri et al. [52] and Gupta et al. [53] utilized CNN in their research to map the spatial arrangement of fibers to the corresponding stress field, accelerating multiscale simulation of composite materials, Saha et al. expand this method from an elastic matrix composite to an elastoplastic matrix [54], Khorrami et al. expand this method to material with elasto-viscoplastic grain microstructures [55]. Yang et al. [56], Su et al. [57], Chang et al. [58], Mianroodi et al. [59], and Mahdi et al. [60] used CNN to predict the properties of composite or nanoporous materials. Yang et al. used CNN to predict the entire stress-strain behavior of binary composite materials along the entire failure path [61]. Rao and Liu [62] and Peng [63] use a three-dimensional convolutional neural network (3D-CNN) to predict the anisotropic effective material properties of RVE for random inclusions or for computational homogenization.

In addition, the application of graph neural network (GNN) has also emerged in recent research. Gulakala et al. [64], Jiang and Chen [65], and Maurizi et al. [66] used GNN models to accelerate finite element simulation in their papers. Dai et al., Perera et al., Karapiperis and Kochmann, and Thomas et al. proposed in their papers that GNN has wide applications in predicting material properties [67], predicting material fracture mechanics behavior [68, 69], and material fatigue detection [70]. Recently, Pitz and Pochiraju applied the transformer model to the field of composite material simulation [71].

Chapter 3

RVE Modeling

3.1 Boundary Conditions of RVE Model

3.1.1 Periodic Boundary Conditions

In order to conduct multiscale modeling of composite materials, we must define boundary value problems at the microscale and clarify scale transition. In this section, we introduce the boundary conditions of our microscopic problem. In our research, we use periodic boundary conditions (PBC) to define our microscale problem.

PBC is a special boundary condition that maintains periodicity in the displacement and stress on each pair of relative boundaries of RVE to simulate the behavior of heterogeneous materials. By applying PBC on RVE, it can be ensured that the stress and strain states of its internal microstructure can be correctly reflected in the macroscopic behavior of the entire material.

We now define the problem at the microscale. $\mathbf{x}$, $\mathbf{X}$, $\mathbf{u}$, and $\mathbf{U}$ are the microscopic and the macroscopic positions and displacement vectors. If we let $\mathbf{u}'$ represent periodic displacement, then we have the following equation:

$$\mathbf{u}(\mathbf{X},\mathbf{x},t) = \boldsymbol{E}(\mathbf{X},t)\cdot\mathbf{x} + \mathbf{u}'(\mathbf{x},t) \tag{3.1}$$

Equation 3.2 indicates that the total strain is composed of the sum of the macroscopic strain and the periodic strain. The periodic fluctuation $\mathbf{u}'$ is taken as the same value on each pair of relatively parallel surfaces of RVE. Equation 3.3 demonstrates that the average value of the periodic strain over the RVE is zero. Therefore we have Equation 3.4, the total strain is equal to the macroscopic strain.

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \boldsymbol{E} + \boldsymbol{\varepsilon}(\mathbf{u}') \tag{3.2}$$

$$\boldsymbol{E}(\mathbf{u}') = \frac{1}{V}\int_V \boldsymbol{\varepsilon}(\mathbf{u}')\mathrm{d}V = 0 \tag{3.3}$$

$$\boldsymbol{E}(\mathbf{u}) = \boldsymbol{E} + \boldsymbol{E}(\mathbf{u}') = \boldsymbol{E} \tag{3.4}$$

We let $\mathbf{u}_i$, $\mathbf{u}_j$, $\mathbf{x}_i$, and $\mathbf{x}_j$ be the displacement and position of each pair of relatively parallel material points on the RVE surface, respectively. Moreover, $\mathbf{F}$ is an operator that defines the micro relationships between the stress and strain. The micro problem is formulated as Equation 3.5:

$$\begin{cases} \boldsymbol{\sigma} = \mathbf{F}(\mathbf{x}, \boldsymbol{\varepsilon}(\mathbf{u}(\mathbf{x}))), \quad \forall \mathbf{x} \in V \\ div_x(\boldsymbol{\sigma}(\mathbf{x})) = 0, \quad \forall \mathbf{x} \in V \\ \mathbf{u}_i - \mathbf{u}_j = \boldsymbol{E}\cdot(\mathbf{x}_i - \mathbf{x}_j), \quad \forall \mathbf{x} \in V \end{cases} \tag{3.5}$$

3.1.2 Other Boundary Conditions

In addition to periodic boundary conditions, there are also various different boundary

conditions applied at the microscale, such as Static Uniform Boundary Conditions (SUBCs), Kinematic Uniform Boundary Conditions (KUBCs), Zero Average Fluctuation Boundary Conditions (ZAFBCs), etc.

Static Uniform Boundary Conditions (SUBCs) are also called Neumann Boundary Conditions. For SUBCs, boundary nodes are assigned a condition where they experience a uniform static load or displacement, ensuring that all nodes along the boundary are subject to the same static force or movement. This kind of boundary condition provides a more accurate estimate for compliant matrix structures. If we let an outward normal vector as $\mathbf{n}$ and a domain variable as ζ. Equation 3.6 shows the SUBCs [72, 73].

$$\frac{\partial \zeta}{\partial \mathbf{n}} \equiv \alpha \quad \text{where} \quad \alpha = \text{constant} \tag{3.6}$$

Kinematic Uniform Boundary Conditions (KUBCs) are also called Dirichlet Boundary Conditions or homogeneous displacement boundary conditions. The main characteristic of these boundary conditions is to apply consistent motion or deformation to the nodes on the boundary. These boundary conditions provide a more accurate estimate of the effective elastic properties of stiff matrix composites [73]. Equation 3.7 shows the KUBCs

$$\zeta = c \quad \text{where} \quad c = \text{constant} \tag{3.7}$$

3.2 Material Properties

This thesis focuses on the machine learning-based multiscale simulation of carbon fiber-reinforced metal matrix composites, which have extensive applications in aerospace, the automotive industry, and various other fields. Carbon fiber is a high-temperature resistant fiber with high strength and modulus, mainly composed of carbon atoms. Carbon fiber has the characteristics of high hardness, high strength, lightweight, high chemical resistance, high

temperature resistance, and low thermal expansion, making it widely used in the aerospace field. One of the commonly used matrix materials for carbon fiber-reinforced composites in the aerospace field is aluminum alloy, such as 7075 aluminum alloy. Aluminum alloy has the characteristics of low density, good mechanical properties, good processing performance, nontoxicity, easy recycling, excellent conductivity, heat transfer, and corrosion resistance. It is widely used in the aerospace, marine industry, chemical industry, metal packaging, transportation, and other fields [74–76].

The fibers of composite selection in our research is intermediate modulus carbon fiber, T1100, which is produced by Toray company. The Young's modulus of this type of carbon fiber is 324 GPa, the tensile strength is 7000 MPa and the Poisson's ratio is 0.1 [1]. The matrix of composite selected in our research is 7075-T6 aluminum alloy. The Young's modulus of this type of alloy is 71.7 GPa and the Poisson's ratio is 0.33. The range of yield point of this alloy is 430 to 480 MPa, the range of tensile strength is 510 to 540 MPa and the elongation at break is 11 % [2]. According to the data above, we can estimate that the hardening modulus of this alloy is approximately 0.013. So, in this work, we assume the fiber of our composite material is linearly elastic material and the matrix is J2 plasticity material.

3.3 RVE Modeling

Our RVE models represent the microscopic structure of long continuous fiber composite materials. There are two parts of our RVE model: fiber and matrix. The fiber is in the center of our RVE model when we don't make geometry parameters of RVE as input of our GRU model. In order to generate those RVE models, a preprocessing script has been written and run in the ABAQUS CAE to get the models and convert them into input files. Those input files have been uploaded to a supercomputer, which we use to run our RVE simulations due to the huge

[1] https://www.toraycma.com/products/carbon-fiber/ (accessed January 1, 2024)

[2] http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA7075T6 (accessed January 1, 2024)

computational cost.

The main function of our preprocess code is to generate and assign the geometry, material properties, mesh, strain series (amplitude), and periodic boundary condition (PBC) of RVE. Our preprocess code is universal. If we have a multi-fiber geometry dataset, which is generated by our preprocess data generate code and modify our code slightly, we can use this script to generate RVEs with multiple fibers, which are randomly located in the RVE model. We design a collection of algorithms in the preprocessing stage. Figure 3.1 presents an example of an RVE model and mesh generated using our preprocess code. The green part of the model represents the fiber and the pale yellow part represents the matrix.

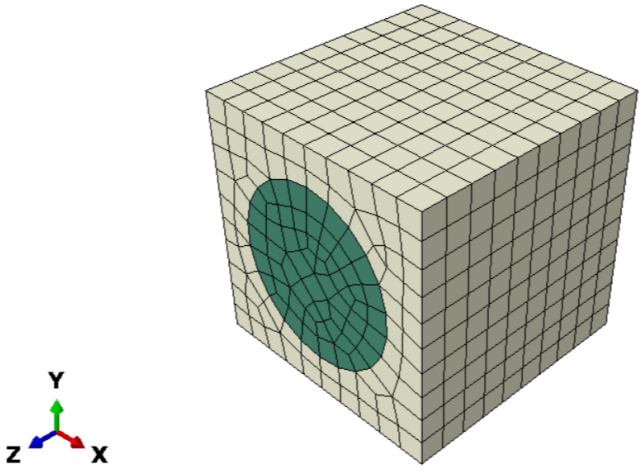


Figure 3.1. An RVE model generated by our preprocess script.

Applying periodic boundary conditions is the most complex part of our preprocess script. In our script, we use equation constraints to constrain corresponding nodes and reference points. We have nine reference points generated by our script for every RVE model to help us transfer strain series to the nodes on the surface of the RVE model. There is a unit displacement applied in every reference point and we use the amplitude of displacement to assign different strain series we generated to nodes in the surface of our RVE through equation constraints. The direction of these unit displacements is based on the subscript of the strain transmitted by the corresponding reference point.

$$Coef_1 \cdot DOF_1 + Coef_2 \cdot DOF_2 + Coef_3 \cdot DOF_3 = 0 \tag{3.8}$$

Equation 3.8 shows the constraint we apply to nodes and reference points. The DOF_1, DOF_2, and DOF_3 are the corresponding degrees of freedom (displacement) of the first node, the opposite node, and the reference point, respectively. According to Equation 3.5, we can know the $Coef_1$ is equal to 1, the $Coef_2$ is equal to -1, and the value of $Coef_3$ depends on the strain we want to apply.

Our script first reads the coordinates of all nodes in the model, then identifies the nodes located on the surface of the model and saves them as a set of surface nodes. Afterward, our script further divides these nodes into pure surface nodes, edge nodes, and vertex nodes through the coordinates of nodes, and creates corresponding subsets of nodes. Finally, we use an algorithm to transform the coordinates of nodes into a type of new index, which is unique for each node on a single surface and the index of the corresponding node on opposite surfaces is equal. Our algorithm uses this index to rank the nodes in each node set and ensure that the positions of corresponding nodes in the two node sets are the same. So, we can directly use the "for" loop to traverse each node in each node set and create equation constraints for each pair of corresponding nodes and reference points. The advantage of this algorithm is that it avoids a large number of iterations caused by comparing the coordinates of each pair of nodes, and reduces the computational cost. We need to constrain the degree of freedom (DOF) of every pair of nodes in the x, y, and z directions, so we need three equation constraints for each pair of nodes.

The details of our periodic boundary condition of the surfaces generation algorithm are in the box of Algorithm 1. In this algorithm, C^i_{aa} is the first coordinate of the i-th node in set a, C^i_{ba} is the second coordinate of the i-th node in set a, C^i_{ab} is the first coordinate of the i-th node in set b, C^i_{bb} is the second coordinate of the i-th node in set b, $new_index^i_a$ is the new index of i-th node in set a, and $new_index^i_b$ is the new index of i-th node in set b. a and b are a pair of opposite surfaces.

Algorithm 1: Periodic Boundary Condition Generation for Surface Algorithm

Input: $node_set_of_surface_a$, $node_set_of_surface_b$ (surface 1 and surface 2 are two opposite surfaces, so the lengths of those two sets are equal)

Result: periodic boundary conditions are generated

initialization;
$l \leftarrow$ length of node set;
$a \leftarrow$ 2 * (dimension of RVE / minimum difference of coordinates between nodes in amplified direction);
for $i = 0$ **to** l **do**
    $new_index_a^i \leftarrow a * C_{aa}^i + C_{ba}^i$;
    $new_index_b^i \leftarrow a * C_{ab}^i + C_{bb}^i$;
    save the new index in the information subset of the i^{th} node in each node set;
end
$node_set_of_surface_a \leftarrow$ sort($node_set_of_surface_a$) by new index;
$node_set_of_surface_b \leftarrow$ sort($node_set_of_surface_b$) by new index;
for $i = 0$ **to** l **do**
    Create equation constraints in x, y, and z direction between the i^{th} node from the node set 1 and 2 and corresponding reference points;
end

It should be pointed out that the amplification factor "a" theoretically only needs to be greater than the minimum value of the size of the RVE model in the nonmagnified direction divided by the minimum value of the coordinate difference of the nodes in the magnified direction to be feasible. In actual code, we can directly multiply that ratio by 2 or use a sufficiently large constant as the value of "a". The function of the amplification factor is to make the new index of each pair of nodes unique.

For the nodes purely on the surface (not on the edges and vertexes), we can directly apply a constraint for it and the corresponding node. However, that does not work in ABAQUS for nodes on edges and vertexes, because they have more than one corresponding node. If we directly apply more than one group of constraints for a node, ABAQUS throws an error and the job aborts. So, we constrain them one by one in a single direction and never go back or intersect. Every node can just appear one time as the first node of the equation constraint. The principle of applying constraint to the nodes on the edges is the same as the vertex nodes but we need to

check the coordinate of nodes in the direction along the edges.

Chapter 3, in full, has been submitted for publication of the material, Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this paper.

Chapter 4

Homogenization

One of the most important aspects of multiscale simulation methods is the transfer of stress and strain information between micro and macro scales, with the homogenization method serving as a bridge between the two scales. In this multiscale framework, we use the strain series applied in RVE model as the input for GRU model and the homogenized stress response from RVE simulation as the output. In this chapter, we introduce the principle of homogenization [77].

The research on homogenization methods originated when researchers began to research how to transform the mechanical properties of microstructures into equivalent properties at the macroscopic scale. Early research mainly focused on the field of solid mechanics, especially on the research of composite materials. The goal of homogenization methods is to handle the mechanical behavior of materials and structures at multiple scales, making the analysis of materials and structures more feasible and efficient. The reason is that it allows for the macroscopic mechanical properties of complex materials to be inferred by analyzing the behavior of relatively small representative volume elements. This significantly reduces computational complexity and time. This greatly reduces the complexity and time of computation. In multiscale modeling, we often use the homogenization principle to connect macroscopic and microscopic scales.

We now define the equations of the homogenization method. The core idea behind these equations is to derive the equivalent stress and strain at the macroscopic scale by averaging the

stress and strain at the microscopic scale. We denote $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ as the microscopic stress and strain tensors, and $\boldsymbol{\Sigma}$ and $\boldsymbol{E}$ as their macroscopic counterparts, respectively. V is the total volume of our RVE model.

$$\boldsymbol{\Sigma} = \frac{1}{V}\int_V \boldsymbol{\sigma} \mathrm{d}V \tag{4.1}$$

$$\boldsymbol{E} = \frac{1}{V}\int_V \boldsymbol{\varepsilon} \mathrm{d}V \tag{4.2}$$

Equation 4.1 and 4.2 are the formulas of the homogenization method, but in numerical simulation, we use Equation 4.3 and 4.4 to implement this method. We use a postprocess script in ABAQUS CAE to extract the data we need from Odb files and use our homogenization code to get the stress and integration point volume data and sum the product of the stress value in the integration point and the volume of the integration point of every stress component and then we divide it by the volume of the entire RVE model to get the homogenization result of stress.

$$\boldsymbol{\Sigma} = \frac{1}{V}\sum_{i=0}^{N_{ip}} v_i \boldsymbol{\sigma}_i \tag{4.3}$$

$$\boldsymbol{E} = \frac{1}{V}\sum_{i=0}^{N_{ip}} v_i \boldsymbol{\varepsilon}_i \tag{4.4}$$

$\boldsymbol{\Sigma}$ and $\boldsymbol{E}$ in the above equations represent macro stress and strain tensors, respectively. $\boldsymbol{\sigma}_i$ and $\boldsymbol{\varepsilon}_i$ represent micro stress and strain tensors at the integration point i, respectively. The v_i is the volume of integration point i and the N_{ip} is the number of integration points in this RVE model. V is the total volume of this RVE model.

Chapter 4, in full, has been submitted for publication of the material, Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this paper.

Chapter 5

Machine Learning

5.1 RNN Models

5.1.1 Architecture of RNN Model

Recurrent neural networks (RNN) are a type of neural network model used for processing series data. Unlike traditional feedforward neural networks, RNN can learn the temporal dependence of series data. RNN introduces a loop in the network to transfer the hidden state from the previous moment to the next moment, thereby being able to remember the previously inputted information. This structure gives RNN advantages in processing tasks such as natural language processing (NLP) and time series prediction. The basic structure of RNN includes the input layer, hidden layer, and output layer. The state of the hidden layer is determined by the current input and the previous hidden state, which is shown in Equation 5.1.

$$\mathbf{h}_t = \tanh(\mathbf{x}_t \mathbf{W}_{ih}^T + \mathbf{b}_{ih} + \mathbf{h}_{t-1} \mathbf{W}_{hh}^T + \mathbf{b}_{hh}) \tag{5.1}$$

In Equation 5.1, $\mathbf{W}_{ih}$ and $\mathbf{W}_{hh}$ are the weight matrices for the input and hidden state, respectively, while $\mathbf{b}_{ih}$ and $\mathbf{b}_{hh}$ are the bias terms. $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$ are the current hidden state and the hidden state from the last time step, respectively, while $\mathbf{x}_t$ is the current input.

Although RNN can process series data, standard RNN has some challenges, such as vanishing gradients and exploding gradients, which make it less effective in handling long series. To address these issues, various improved models have been proposed, and Gated Recurrent Unit (GRU) is one of them.

5.1.2 Architecture of GRU Model

GRU is a variant of the RNN model, which is an improvement on RNN aimed at addressing long-term dependency issues. It introduces a gating mechanism that controls the flow and forgetting of information through update and reset gates. GRU can better solve the problem of vanishing gradients and has better performance in processing long series. The GRU model is used in our research to learn the mechanical behavior of RVE.

The reset gate of GRU determines how much old memory needs to be ignored. It determines how to combine new input with previous memory in order to control the flow of information when processing information at different time steps. The update gate of GRU determines how many old memory states need to be retained, and it controls the weight of new input information and previous memory states. The output value of the update gate is between 0 and 1, where 0 indicates complete disregard of previous memory and 1 indicates complete retention of previous memory. Meanwhile, GRU also has a hidden state, which is the output of the GRU layer at the current time step and also one of the inputs for the next time step. GRU improves the performance of the RNN model through a gate control mechanism.

$$\mathbf{r}_t = \sigma(\mathbf{W}_{ir}\mathbf{x}_t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_{hr}) \tag{5.2}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_{iz}\mathbf{x}_t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_{hz}) \tag{5.3}$$

$$\mathbf{n}_t = \tanh(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}_{in} + \mathbf{r}_t * (\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn})) \tag{5.4}$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) * \mathbf{n}_t + \mathbf{z}_t * \mathbf{h}_{t-1} \tag{5.5}$$

The Equations 5.2 to 5.5 illustrate the computations performed by each layer of the GRU for each element in the input series. In those functions, $\mathbf{h}_t$ is the hidden state at time t and $\mathbf{h}_{t-1}$ is the hidden state at time t-1 or the initial hidden state. The $\mathbf{r}_t$, $\mathbf{z}_t$, and $\mathbf{n}_t$ are the reset gate, update gate, and new candidate, respectively. σ is sigmoid function. The structure of GRU is illustrated more intuitively in Figure 5.1. * in the functions and Figure 5.1 is the Hadamard product. In this work, we use the PyTorch library to design and train our GRU model.

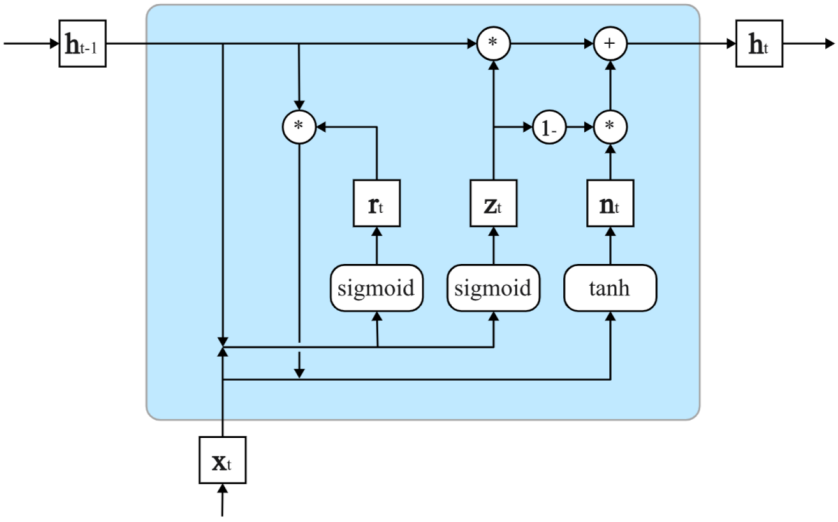


Figure 5.1. The structure of GRU.

$$\mathbf{o}_t = \mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o \tag{5.6}$$

Finally, we also need a linear layer to map the hidden state to the output, the input size of this layer is equal to the hidden size of GRU and the output size is equal to the output size we set.

Equation 5.6 is shown as the linear layer and $\mathbf{o}_t$ is the output of our model at time step t. In this work, when we use the strain series as input of the model, the input and output size is equal to 6, the hidden size is equal to 50 and the number of layers is equal to 3. Hyperparameters of our model are determined by tuning and often have slight adjustments for different factors such as the size of the training dataset. The structure of the entire RNN with the GRU model we use in this research is illustrated in Figure 5.2.

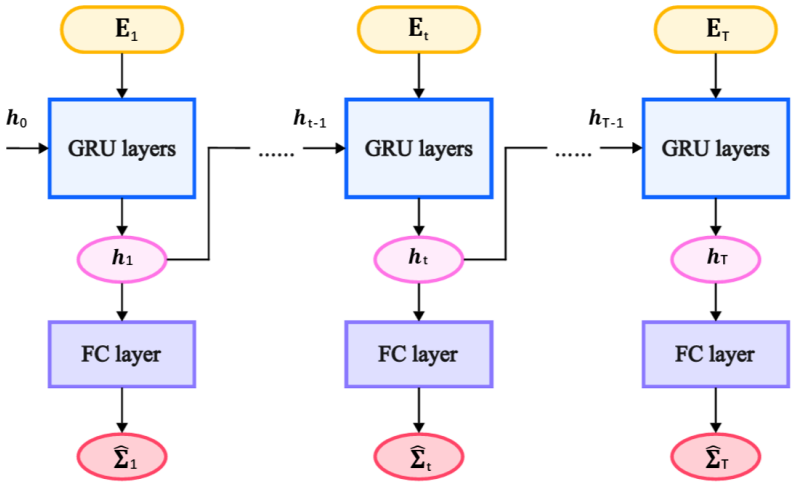


Figure 5.2. The structure of our GRU based surrogate model.

In this figure, $\mathbf{E}_t$ (Macroscopic strain), $\mathbf{h}_t$, and $\hat{\mathbf{\Sigma}}_t$ (Predicted values of macroscopic stress) are the input, hidden state, and output, respectively. The input should be the strain tensor and the output should be the stress tensor. T is the number of time steps or the length of the series. The hyper-parameters of the GRU model in this work are illustrated in the Table 5.1.

When tuning the model, we adjust each hyperparameter individually. Specifically, while tuning a particular hyperparameter, all other hyperparameters are kept constant. The model is trained using the training dataset, but the number of training epochs is set to one-fourth of the total epochs. We determine the optimal hyperparameters by comparing the average mean squared error (MSE) on the test dataset. The equation of MSE is illustrated in Equation 5.7. In this equation, N is the number of stress component, T is number of time steps, $\Sigma_n^{\langle t \rangle}$ is the true

Table 5.1. Hyper-parameters of the GRU model in this work.

Hyper-parameters	Value
Input size of GRU	6
Hidden size of GRU	50
Number of GRU layers	3
Input size of FC layer	50
Output size of FC layer	6
Number of FC layer	1

value of stress component n in time step t, and $\hat{\Sigma}_n^{\langle t \rangle}$ is the predicted value of stress component n in time step t.

$$\text{MSE} = \frac{1}{NT} \sum_{n=1}^{N} \sum_{t=1}^{T} \left\| \Sigma_n^{\langle t \rangle} - \hat{\Sigma}_n^{\langle t \rangle} \right\|_2^2 \tag{5.7}$$

5.2 Data Generation and Sampling

When we run our preprocess code in ABAQUS CAE, we need some preprocess datasets including the strain series dataset. Our preprocess data generation script includes all the parameters we need to generate the preprocess data, and it can call multiple packages to complete the data generation work. There are three packages we developed for preprocess data generation: material properties generation package, sampling package (random strain series generation package), and random geometry generation package. The reason we designed the latter two algorithms is that if we want to randomly assign the material properties and geometry of our RVE model and use those properties as input for our GRU model, we also need a material properties dataset and a geometry dataset.

The sampling package is the most significant package we need for preprocessing dataset generation. This package is capable of generating the random strain series required for our

simulations. This package uses random increments which are in a small range and can be positive or negative to generate strain series and the increments of different items of strain are different. The package implements the return mapping algorithm of J2 plasticity to compute stress and check how many points in this strain series make the stress series entry plastic range. If the number of plastic points of a series exceeds a threshold we set, this series is accepted and saved in our dataset. The advantage of this algorithm is we can set the minimum number of plastic points in the series we generated and don't affect the randomness of the series, which can give our GRU model enough information to learn the plastic behavior of the material and don't affect the generalization ability of our model. Meanwhile, this algorithm can make sure the strain doesn't exceed a maximum value. This ensures that the strain series generated by this algorithm does not cause the material to fracture. Strain and increment datasets is saved by this package. The details of our strain sampling algorithm are in the box of Algorithm 2 and there is an example of a strain series generated by our sampling algorithm that is illustrated in Figure 5.3.

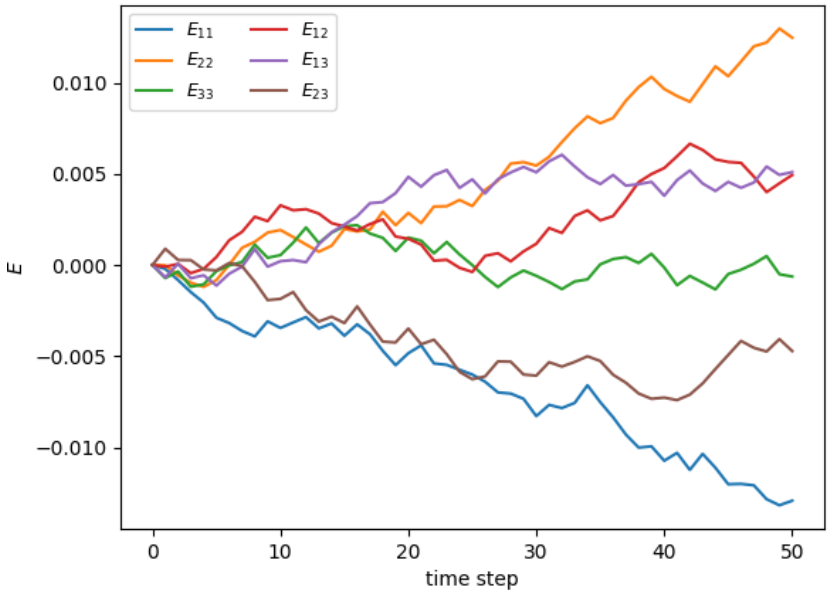


Figure 5.3. An example of a strain series generated by our strain sampling algorithm.

The material properties generation package can read the material properties ranges of

Algorithm 2: Strain Sampling Algorithm

Input: number of time steps, max size of strain increments, material properties dataset, number of series, min number of points in plastic range of a series, mixed hardening parameter (β), max strain before material fracture

Output: strain dataset, stress dataset (optional), increment dataset (optional)

```
initialization;
i ← 0;
strain_dataset ← empty tensor;
L ← number of series;
T ← number of time steps;
E_max ← max strain before material fracture;
min_num_p ← min number of plastic points;
max_inc ← max size of strain increments;
while i < L do
    initialization;
    j ← 0;
    number_of_plastic_points ← 0;
    strain ← zeros tensor;
    series ← empty tensor;
    while j < T do
        inc ← random.uniform(- max_inc, max_inc);
        strain ← strain + inc;
        if strain < E_max then
            save this point to the series;
            compute stress in this step by return mapping algorithm ;    /* Call a
             function */
            if this point is in plastic range then
                number_of_plastic_points ← number_of_plastic_points + 1;
            end
            j ← j + 1;
        else
            strain ← strain − inc;
        end
    end
    if number_of_plastic_points > min_num_p then
        save this series to strain_dataset;
        i ← i + 1;
    end
end
```

fiber and matrix from the preprocess data generation script and randomly generate material properties in those ranges. The result is written in two Txt files. The geometry dataset generation package can generate the data needed for an RVE model with multiple fibers randomly located in it, specifically which can generate the random coordinate dataset of the center of fibers. This package randomly generates the coordinate values for the fiber centers and checks the distances between fibers as well as between fibers and the edge of the RVE model to avoid overlaps. If the generated coordinates cause overlaps, the code deletes and regenerates them, repeating the check until they fit. The package also counts the number of iterations, as the process can sometimes get stuck due to previously generated coordinates. If the number of iterations exceeds a threshold, the code deletes all the coordinates of fibers in this RVE and restarts a new iteration. The parameters we choose for data generation in this work is illustrated in Table 5.2

Table 5.2. Parameters of data generation.

Parameters	Value
Number of series	10000
Number of time steps	50
Minimum number of plastic points	20
Maximum size of strain increments	0.001

5.3 Training

In this work, the initial learning rate is usually equal to 0.005. The learning rate decay mechanism is implemented in our training loop, which can multiply the current learning rate with a preset reduction factor when the number of epochs is equal to some preset values during our training. This strategy can help us save training time since we can use a larger initial learning rate to decrease the loss faster and then use a reduced learning rate to avoid the oscillation and non-convergence of the model. The hyper-parameters of training process is illustrated in Table

5.3. We train the model using the method of gradient descent.

Table 5.3. Hyper-parameters of training process.

Hyper-parameters	Value
Initial learning rate	0.005
Batch size	100
Number of epochs	1600
Optimizer	Adam

To train and test our GRU model, we divided 90% of the entire dataset into a training dataset and 10% into a testing dataset, since obtaining training data in this research is expensive. Before using our datasets to train the GRU model, we used the Min-Max Scaler to normalize our datasets, which is illustrated in Equation 5.8. In this equation, $\Sigma_n^{\langle t \rangle}$ and $E_n^{\langle t \rangle}$ are the original stress and strain component n in time step t of sequence m, $\tilde{\Sigma}_n^{\langle t \rangle}$ and $\tilde{E}_n^{\langle t \rangle}$ are the scaled stress and strain component n in time step t of sequence m, $\Sigma_{\min}$ and $\Sigma_{\max}$ are the minimum and maximum stress value for all components, all time steps, and all sequences, and $E_{\min}$ and $E_{\max}$ are the minimum and maximum strain value for all components, all time steps, and all sequences.

$$\tilde{\Sigma}_{mn}^{\langle t \rangle} = \frac{\Sigma_{mn}^{\langle t \rangle} - \Sigma_{\min}}{\Sigma_{\max} - \Sigma_{\min}}, \quad \tilde{E}_{mn}^{\langle t \rangle} = \frac{E_{mn}^{\langle t \rangle} - E_{\min}}{E_{\max} - E_{\min}} \tag{5.8}$$

When we train our model, the batch size is usually equal to 100. In addition, we plot the mean square error (MSE) of training and testing datasets with epochs. When we finish the training process, we plot an example of the true value and prediction of stress versus time steps in a single figure and compare them. Those plots help us to preliminarily judge the effectiveness of training and the performance of our trained model. The training MSE of our model when we finish the training process is about 2.6 e^{-7}. The training and testing loss curves of our training process are illustrated in Figure 5.4.

Chapter 5, in full, has been submitted for publication of the material, Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this paper.

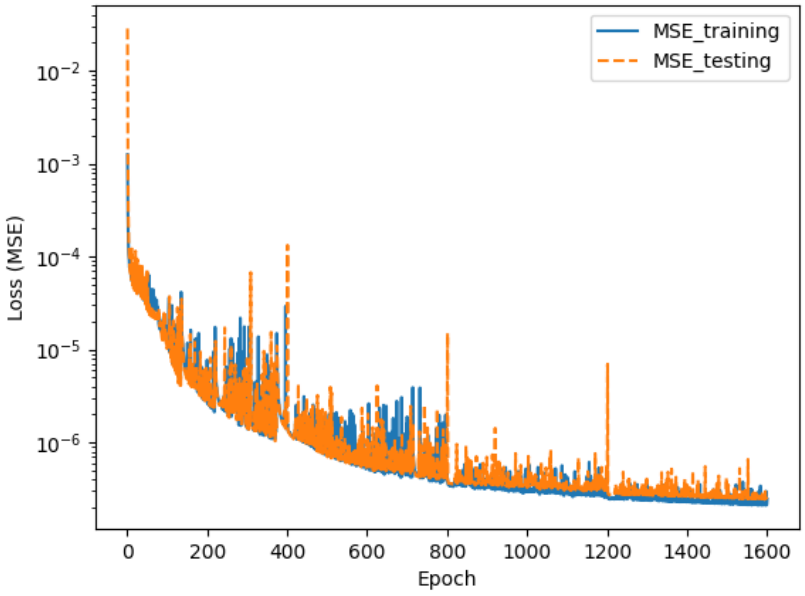


Figure 5.4. Training and testing loss curves.

Chapter 6

Multiscale Simulation

6.1 Multiscale Methods

There are many multiscale methods proposed by researchers. At present, researchers mainly classify these methods into two categories: hierarchical methods and concurrent methods.

Hierarchical Methods refer to techniques that derive equivalent coarse-scale models from fine-scale information, such as electronic structure or atomic scale, to describe behavior at a coarser scale, like material constituents and component scale. The core idea of these methods is to utilize the physical phenomena or mathematical relationships at the fine scale to predict the behavior at the coarse scale. Hierarchical Methods include the Fine Element Squared Method (FE^2 Method).

Concurrent Methods are approaches that simultaneously resolve fine-scale and coarse-scale responses in different regions of the problem domain, facilitating information exchange at the interfaces. These methods are particularly useful when scale separation does not exist or when the quantities of interest are at the fine scale, such as crack propagation. Concurrent Methods include the Bridging Domain Method.

We implement a FE^2 multiscale framework based on trained GRU in this research. In the traditional FE^2 method, we transfer the strain obtained at the macroscopic scale in finite element simulation to the microscopic scale, simulate the material's RVE models at the microscopic

scale, homogenize the simulation results, and then transfer the homogenized results back to the macroscopic scale as macroscopic stress. Since simulating a large number of RVE models is expensive, in our framework, we use a trained GRU model to replace the microscopic simulation of RVE models. In this case, We directly use the strain obtained at the macroscopic scale as input for GRU and use the output of GRU as macroscopic stress for macroscopic simulation.

In this thesis, we use commercial software ABAQUS to complete this process. In ABAQUS we can write a user material (UMAT) subroutine code to implement a constitutive relationship of material, which is not built-in in ABAQUS. We reconstructed our trained GRU model in UMAT as the material model. When we run a simulation in ABAQUS, we can command ABAQUS to call the UMAT subroutine to implement the trained GRU model in multiscale simulation.

6.2 Implementation of Trained GRU in UMAT

In this thesis, we designed a universal Fortran algorithm for reconstructing trained GRU models in ABAQUS UMAT, which is illustrated in Algorithm 3. We provide a detailed introduction to this algorithm in this section.

Our Fortran algorithm is divided into three parts. In the first part, we need to define some subroutines that need to be frequently called in subsequent code, including matrix operator subroutines, activation function subroutines, and Min-Max Scaler and Min-Max Scaler inverse subroutines. This section makes our subsequent code more concise. Meanwhile, the ability to call a large number of pre-written subroutines also makes our code easier to modify and less prone to errors.

The second part is the subroutine of our trained GRU, in which we reconstruct the structure of the GRU model. Firstly, we need to declare variables that store model parameters. Meanwhile, we also need to declare the temporary variables required for matrix operations. It

should be noted that due to the large number of model parameters, we use another Python script to directly write the model parameters to the UMAT file, avoiding the transfer of a large number of dataset files. Then we need to load the model parameters, input vector (Call the Min-Max Scaler subroutine at the same time), and hidden state vector. Finally, we call the matrix operation subroutine to construct the GRU model step by step, layer by layer, for the computation of each variable.

In the final part, we first need to write the UMAT interface, which is used by ABAQUS to call the UMAT subroutine. This part can be found in the documentation of ABAQUS and is immutable. In addition, it is necessary to declare certain variables that are required but not provided by the UMAT interface. Then we need to compute the Jacobian matrix. It should be noted that we use the Quasi-Newton solver in ABAQUS, which eliminates the need to update the Jacobian matrix at each step. Then we call the GRU subroutine to obtain the output vector and hidden state in this step. Finally, we need to call the Min-Max Scaler inverse subroutine to obtain and update the stress vector and directly update hidden states.

In Algorithm 3, STRAN_{t-1} and DSTRAN_t are used to represent the strain vector before the increment starts and the strain increment vector, respectively, while STRESS_t is used to represent the stress vector at the end of the increment. STATEV_{t-1} and STATEV_t represent the vectors used to store solution-dependent state variables before and after the start of strain increment in UMAT, respectively. We need to define their length in ABAQUS "Depvar". Here we use it to store the hidden state of the GRU model, and its length is equal to the sum of the lengths of the hidden state vectors of all GRU layers. DDSDDE is used to represent the Jacobian matrix.

After completing the above steps, we obtain a UMAT code without written parameters. In the part of loading parameters, we pre-write the name of the parameter matrix. A parameter writer we designed reads UMAT code without parameters, while also reading parameter datasets as matrices. The parameter writer can traverse each line of UMAT code, find the location where the parameters need to be written, and sequentially insert the parameter matrix into the

Algorithm 3: Trained GRU in UMAT Algorithm (Fortran)

Input: STRAN_{t-1}, STATEV_{t-1}, DSTRAN_t
Output: STRESS_t, STATEV_t

Section1 : Fundamental subroutines we need;
Define Matrix Addition Subroutine;
Define Matrix Minus Subroutine;
Define Matrix Multiplication Subroutine;
Define Matrix Hadamard Product Subroutine;
Define Tanh Function Subroutine;
Define Sigmoid Function Subroutine;
Define Min-Max Scaler Subroutine;
Define Min-Max Scaler Inverse Subroutine;

Section2 : Trained GRU Subroutine;
Declare variables that store GRU model parameters and outputs;
Declare temporary variables in GRU computations;
Declare other variables;
Load GRU model parameters;
Load value from STATEV_{t-1} to hidden vectors of the previous step of all the GRU layers;
Load value from STATEV_{t-1}, DSTRAN_t to GRU input vector;
Call Min-Max Scaler Subroutine for GRU input vector;
Call subroutines we defined in section 1 to reconstruct the trained GRU model and compute hidden vectors of this step and GRU output vector;

Section3 : UMAT Subroutine;
UMAT Interface; `/* Prohibit modifying UMAT interface */`
Declare variables of inputs and outputs of trained GRU subroutine;
Declare variables for DDSDDE computation;
Declare other variables;
Compute DDSDDE;
Call Trained GRU Subroutine to compute hidden vectors of this step and GRU output vector;
Call Min-Max Scaler Inverse Subroutine to obtain STRESS_t;
Update the hidden states of this step to STATEV_t

corresponding positions. Finally, the modified code is written into a new Fortran file to obtain the final executable UMAT subroutine with parameters.

Chapter 6, in full, has been submitted for publication of the material, Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this paper.

Chapter 7

Result and Discussion

7.1 Test the Performance of New Sampling Algorithm

To demonstrate the better performance of our sampling algorithm, we compared it with algorithms from previous literature. He and Semnani [20] Mozaffar et al. [19] proposed some data sampling algorithms; we chose the polynomial interpolation algorithm in the first paper and algorithm 2 in the second one as reference algorithms 1 and 2. We use those 2 algorithms and our new algorithm to generate datasets and train the GRU model with the same structure and same material properties respectively. In this test, the dataset size we generated for each algorithm is equal to 10000 and the number of time steps of each series is equal to 50. In addition, the minimum number of plastic points is set to 20. The material we used for this test is 7075 aluminum alloy, which is the same as the matrix of composite in this paper and the material parameters of it are shown in section 3.2. We use the linear cyclic load to do the tests and calculate the relative norm error of each algorithm. The protocol of this type of load is a linear cyclic pattern and the number of time steps is equal to 100. For example, the strain component starts at 0, increases from step 1 to step 25, then decreases from step 26 to step 75, and finally increases back to 0 from step 76 to step 100. The absolute value of the increment is constant. The initial increments of normal strain and shear strain are ± 0.0004 and ± 0.0001, respectively. In addition, we randomly change the symbols of six components, so that there are 64 protocols

in each incremental case. We use these 64 protocols to test the models and compute the relative error for each protocol, then we sum those errors and divide them by 64, ultimately obtaining the averaged error of each model. The formula for computing the relative error is shown in the Equation 7.1.

$$E(\hat{\boldsymbol{\sigma}}) = \frac{\| \hat{\boldsymbol{\sigma}} - \boldsymbol{\sigma}_{true} \|_2}{\| \boldsymbol{\sigma}_{true} \|_2}, \quad E(\hat{\mathbf{U}}) = \frac{\| \hat{\mathbf{U}} - \mathbf{U}_{true} \|_2}{\| \mathbf{U}_{true} \|_2}, \quad E(\hat{\mathbf{RF}}) = \frac{\| \hat{\mathbf{RF}} - \mathbf{RF}_{true} \|_2}{\| \mathbf{RF}_{true} \|_2} \tag{7.1}$$

In the equation above the $E(\hat{\boldsymbol{\sigma}})$, $E(\hat{\mathbf{U}})$, and $E(\hat{\mathbf{RF}})$ are the relative error of our model. The $\hat{\boldsymbol{\sigma}}$, $\hat{\mathbf{U}}$, and $\hat{\mathbf{RF}}$ are the predictions of stress, displacement, and reaction force and the $\boldsymbol{\sigma}_{true}$, $\mathbf{U}_{true}$, and $\mathbf{RF}_{true}$ are the true values computed by our homogenization code. It should be pointed out that, the $\hat{\boldsymbol{\sigma}}$, $\hat{\mathbf{U}}$, and $\hat{\mathbf{RF}}$ should be tensors when we consider the relative error of all the time steps. When we compute the relative error of a certain time step, the $\hat{\boldsymbol{\sigma}}$, $\hat{\mathbf{U}}$, and $\hat{\mathbf{RF}}$ can be scalars. The advantage of this formula is that the relative error can more intuitively measure the performance and accuracy of our trained model than the absolute error.

During the testing process, we enable the step size (strain increment size) of the loading protocol to change within a certain range. Specifically, we uniformly take 50 numbers between 0.75 and 5 as scaling factors. It should be pointed out that the scaling factor depends on the properties of the material, which means that the strain increment after scaling should not be too small to prevent the protocol from entering a plastic state, or too large to be detached from reality. In each increment size, the 64 loading case are multiplied by the corresponding scaling factors to ensure that the absolute increment values for each case are different. So, we have 3200 samples in our testing processes. Then we plot the relative norm error of those three models trained by the datasets generated from the above algorithms versus the maximum step size of the corresponding incremental case. The comparison of relative errors between our new sampling algorithm and the reference algorithms is illustrated in Figure 7.1. It is very clear, that the performance of the model trained by our new sampling algorithm is the best, regardless of whether large or small

time steps are considered. The reason is that the ratio of increment size to the initial yield point of the material is small in this work, which means that the material properties and increment size we chose in this paper require more steps to enter the plastic range. Our algorithm design solves the problem of series that is difficult to enter the plasticity range when the ratio of increment size to the initial yield point of the material is small.

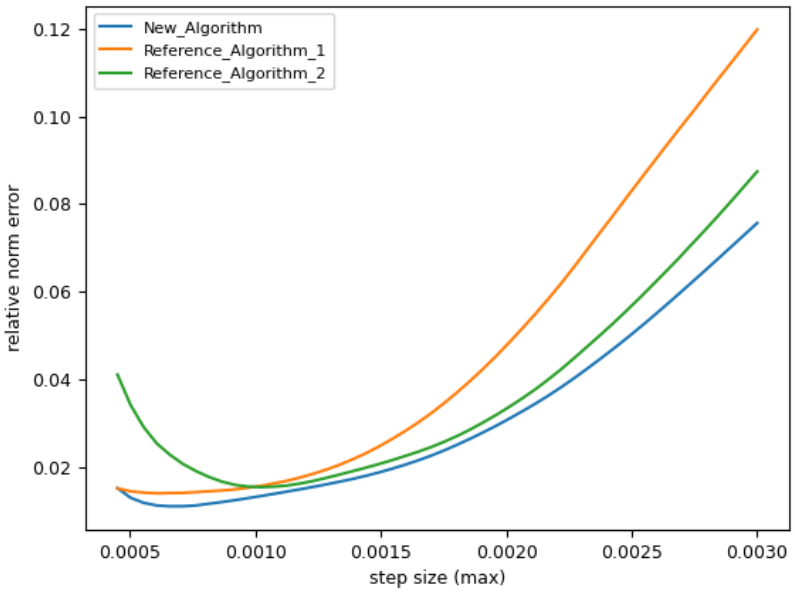


Figure 7.1. Comparison of relative errors for our new sampling algorithm and the reference algorithms.

7.2 Test in Microscale Problems

In this part, we use the data from our testing dataset and the triaxial strain state we designed as input to test the accuracy of our GRU model. We plot the prediction of our trained model and the true value of stress that is obtained by our RVE simulation and homogenization in a single figure to illustrate the performance of our trained model. In addition, we compute the relative error of our trained model and evaluate the accuracy of it.

In this chapter, we assume that our RVE is a cube with a single fiber in the center of the RVE model, similar to the RVE in Figure 3.1. We assume the dimension of the RVE is equal to 1 and the radius of the fiber is equal to 0.35. In addition, the size of the dataset we generated is equal to 10000, the number of time steps of each series is equal to 50, and the minimum number of plastic points is set to 20. To train and test our GRU model, we divided 90% of the entire dataset into a training dataset and 10% into a testing dataset.

7.2.1 Test in Testing Dataset

In this subsection, we discuss the test of our trained GRU model using the testing dataset we generated. We randomly choose a series from our testing dataset as input for our trained model and plot the true value and prediction versus time steps. The series in our testing dataset is extracted from the dataset generated by our new sampling algorithm and RVE simulation, so the generation method is the same as the series in our training dataset. That means the series in the testing dataset must have enough points in the plastic range, which can test our model very well not only in the elastic range but also in the plastic range. The performance of our model is illustrated in Figure 7.2.

In Figure 7.2, the red line represents the true value in this case and the blue dashed line represents the prediction. Figure 7.2a to Figure 7.2f illustrate each component of the stress vector, Σ_{11}, Σ_{22}, Σ_{33}, Σ_{12}, Σ_{13}, and Σ_{23} respectively. The relative error in this case is 0.00989. This figure demonstrates that the predicted values closely align with the actual values across all six stress components.

7.2.2 Test in Triaxial Stress State

In this subsection, we discuss the test of our trained GRU model using the triaxial strain state we designed. There are four kinds of strain states we designed. We plot those four strain

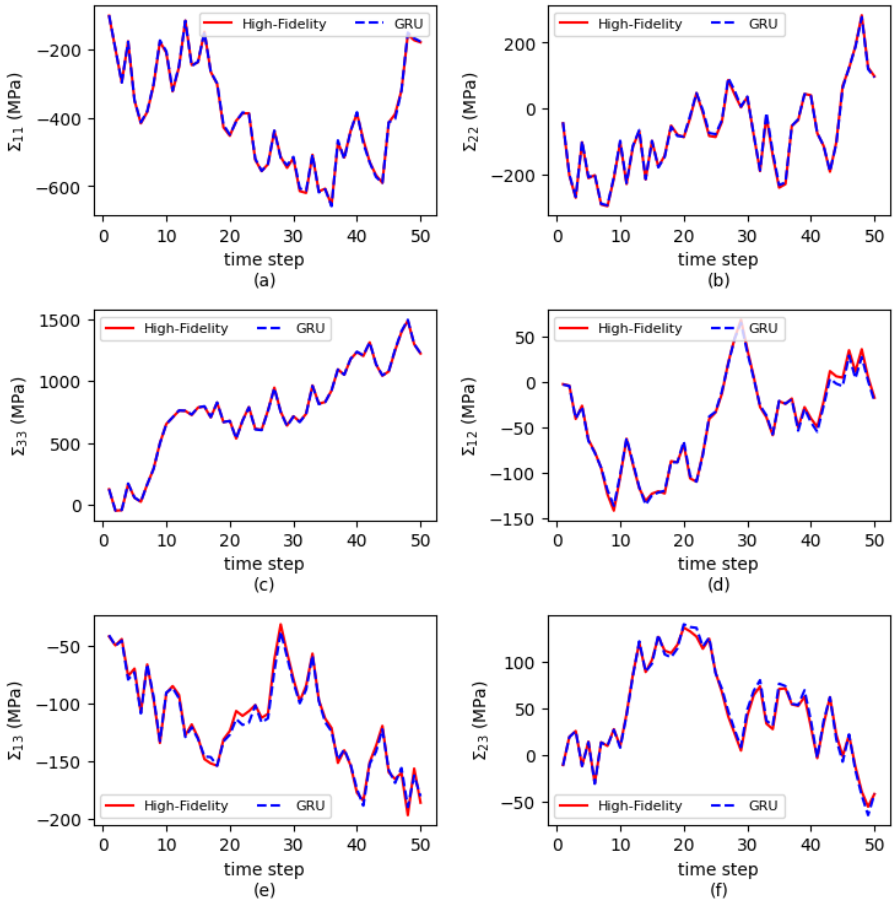


Figure 7.2. The true value and prediction versus time steps of an example of our testing dataset.

states in Figure 7.3. Figure 7.3a illustrates the monotonic triaxial strain state. Figure 7.3b illustrates the cyclic strain state (In this case, the shear strain is all equal to 0). Figure 7.3c illustrates the cyclic strain state with nonzero shear strain. Figure 7.3d illustrates the sinusoidal cyclic strain state with zero shear strain.

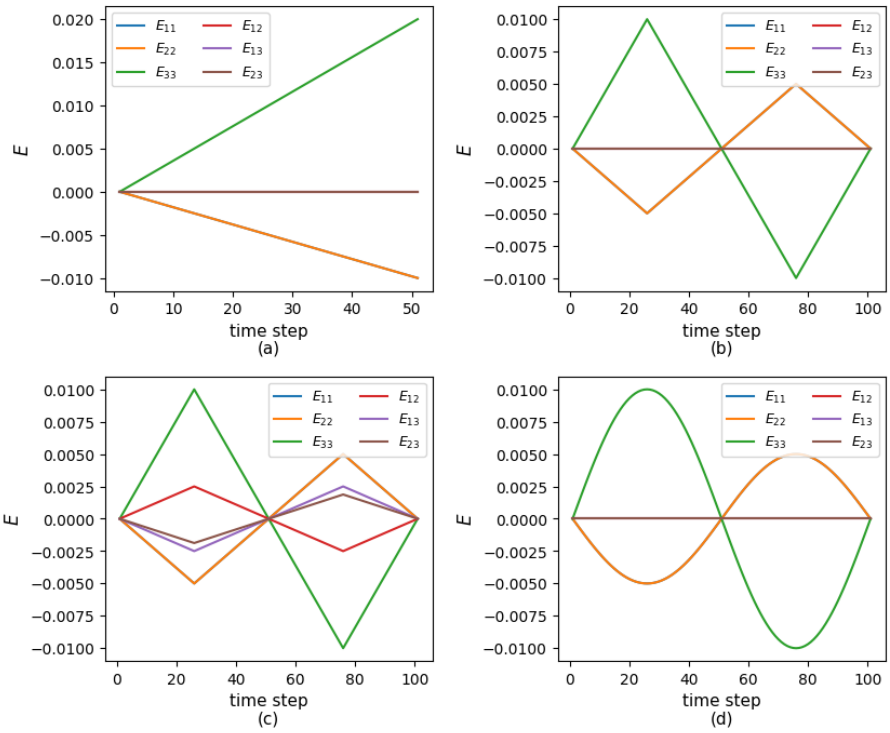


Figure 7.3. Four triaxial strain states we used to test our model (In (a), (b), and (d), $E_{11} = E_{22}$, $E_{12} = E_{13} = E_{23} = 0$; in (c), $E_{11} = E_{22}$).

In this part, we use the homogenization result of the high-fidelity simulation of our RVE model as the true value (reference value) in the test. To be more specific, we use our preprocess code to assign the generated strain data of these strain states to the RVE model through the corresponding PBC and run the simulation. The postprocess and homogenization code is used to compute the homogenized stress of this RVE as true value. The prediction is predicted by our trained GRU model.

In the first strain state, shown in Figure 7.3a, the E_{11} and E_{22} are monotonically decreasing from 0, and the E_{33} monotonic increase from 0. The minimum value and fixed increment of E_{11}

and E_{22} are -0.01 and -0.0002 respectively, and the maximum value and fixed increment of E_{33} are 0.02 and 0.0004 respectively. The value of strains E_{12}, E_{13}, and E_{23} is a constant and equal to 0. The number of time steps is equal to 50 in this state. The direction 3 (or z-direction) is the direction along the fiber in our RVE model. The performance of our model is illustrated in Figure 7.4.

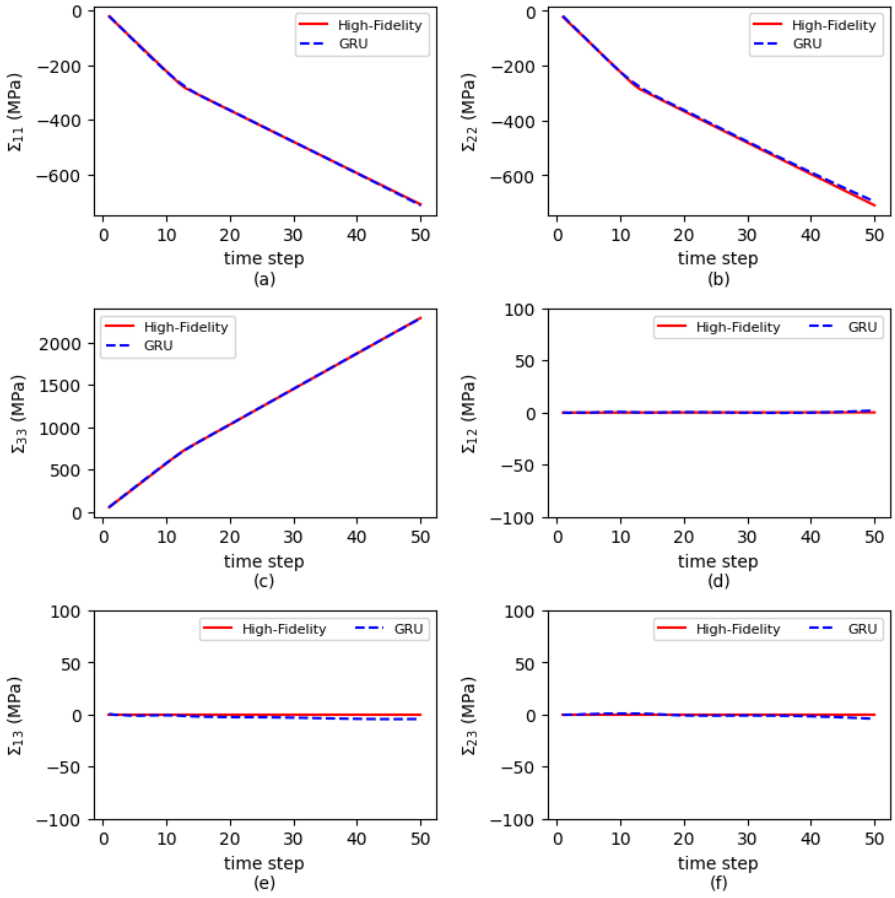


Figure 7.4. The predicted and true stress versus time steps of monotonic triaxial strain state. The strain state is shown in Figure 7.3a.

In Figure 7.4, the red line represents the true value in this case and the blue dashed line represents the prediction. Figure 7.4a to Figure 7.4f illustrate all components of the stress vector. The relative error in this case is 0.00485. This figure demonstrates that the predicted values closely align with the actual values across all six stress components and the prediction of Σ_{12},

Σ_{13}, and Σ_{23} is very close to zero.

In the second strain state, shown in Figure 7.3b, E_{11} and E_{22} are decreasing, increasing, and decreasing again from 0, and E_{33} is increasing, decreasing, and increasing again from 0. The peak value and fixed increment of E_{11} and E_{22} are ± 0.005 and ± 0.0002 respectively, and the peak value and fixed increment of E_{33} are ± 0.01 and ± 0.0004 respectively. The value of strains E_{12}, E_{13}, and E_{23} is a constant and equal to 0. The number of time steps is equal to 100 in this state. Figure 7.3b provides a more intuitive description of the details of this triaxial strain state. The performance of our model in this strain state is illustrated in Figure 7.5. In this state, we also plot the true value and prediction of Σ_{11} and Σ_{33} versus the strain, which is illustrated in Figure 7.6.

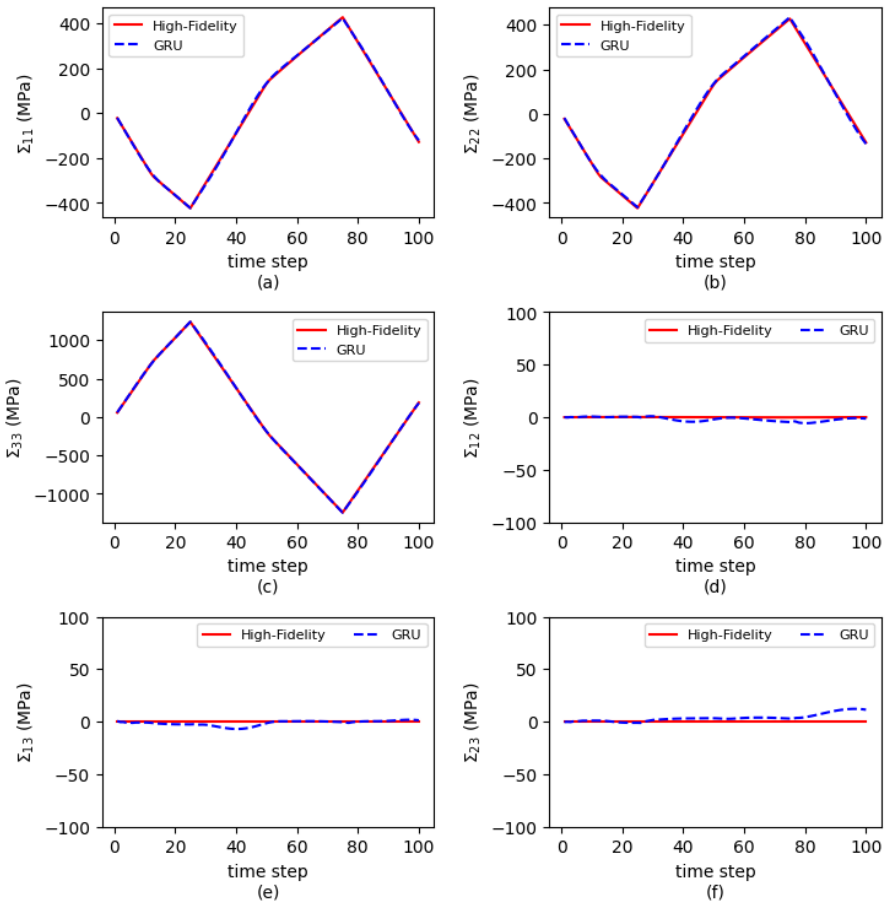


Figure 7.5. The predicted and true stress versus time steps of cyclic triaxial strain state. The strain state is shown in Figure 7.3b.

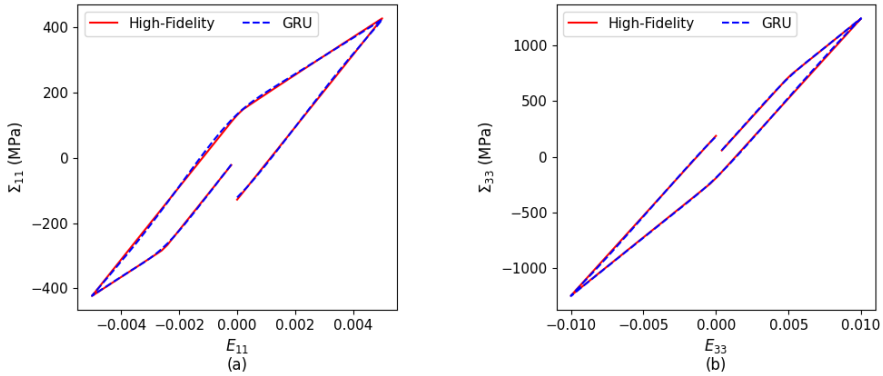


Figure 7.6. The predicted and true stress versus strain of cyclic triaxial strain state. The strain state is shown in Figure 7.3b.

In Figure 7.5 and Figure 7.6, the red line represents the true value in this case and the blue dashed line represents the prediction. Figure 7.6a and Figure 7.6b illustrate the Σ_{11} and Σ_{33}, respectively. The relative error in this case is 0.0147. Those two preceding figures demonstrate the prediction of Σ_{33} is the best, which is more accurate than the prediction of Σ_{11} and Σ_{22}. Meanwhile, the Σ_{12}, Σ_{13}, and Σ_{23} are very close to 0. The lines of predictions of stress follow the line of the true value of stress very well.

In the third strain state, shown in Figure 7.3c, E_{11} and E_{22} are decreasing, increasing, and decreasing again from 0, and E_{33} is increasing, decreasing, and increasing again from 0. The peak value and fixed increment of E_{11} and E_{22} are ± 0.005 and ± 0.0002 respectively, and the peak value and fixed increment of E_{33} are ± 0.01 and ± 0.0004 respectively. The increasing and decreasing patterns of E_{12}, E_{13}, and E_{23} is the same as normal strains, and the initial values of the increment of E_{12}, E_{13}, and E_{23} are 0.0001, -0.0001, and 0.000075 respectively. The number of time steps is equal to 100 in this state. The performance of our model in this strain state is illustrated in Figure 7.7. In this state, we also plot the true value and prediction of Σ_{11}, Σ_{33}, Σ_{12}, and Σ_{23} versus the strain, which is illustrated in Figure 7.8.

In Figure 7.7 and Figure 7.8, the red line represents the true value in this case and the blue dashed line represents the prediction. Figure 7.8a to Figure 7.8d illustrate the stress Σ_{11}, Σ_{33}, Σ_{12}, and Σ_{23}, respectively. The relative error in this case is 0.0152. Those two preceding figures

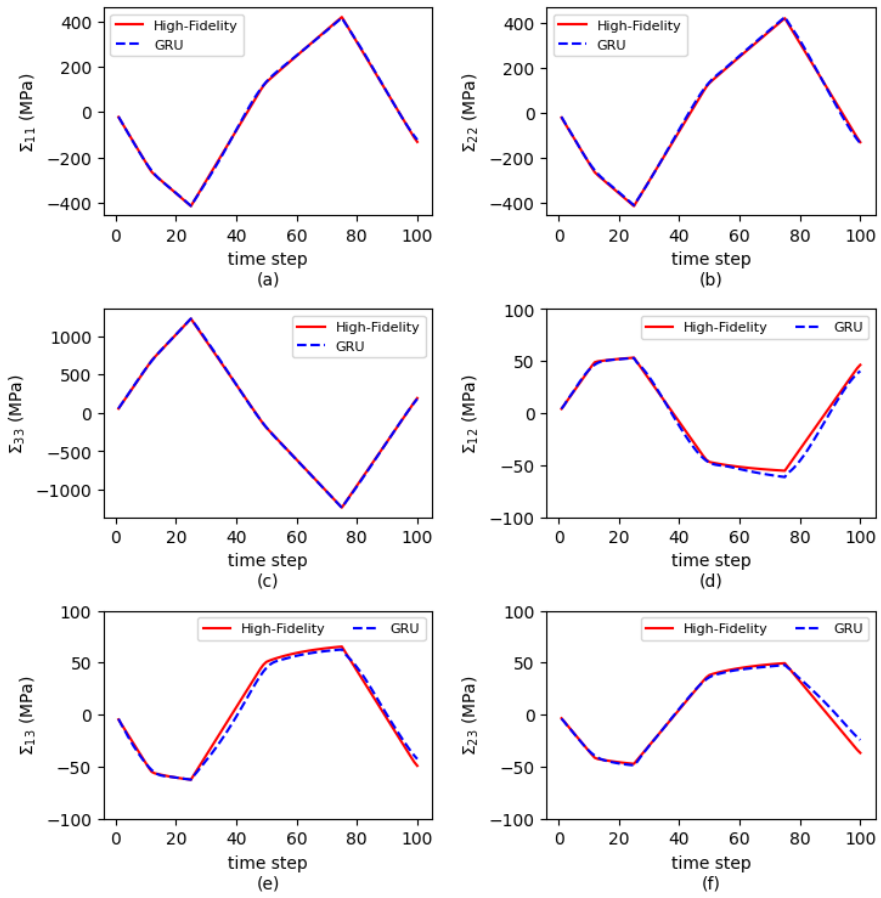


Figure 7.7. The predicted and true stress versus time steps of cyclic triaxial strain state with nonzero shear strain. The strain state is shown in Figure 7.3c.

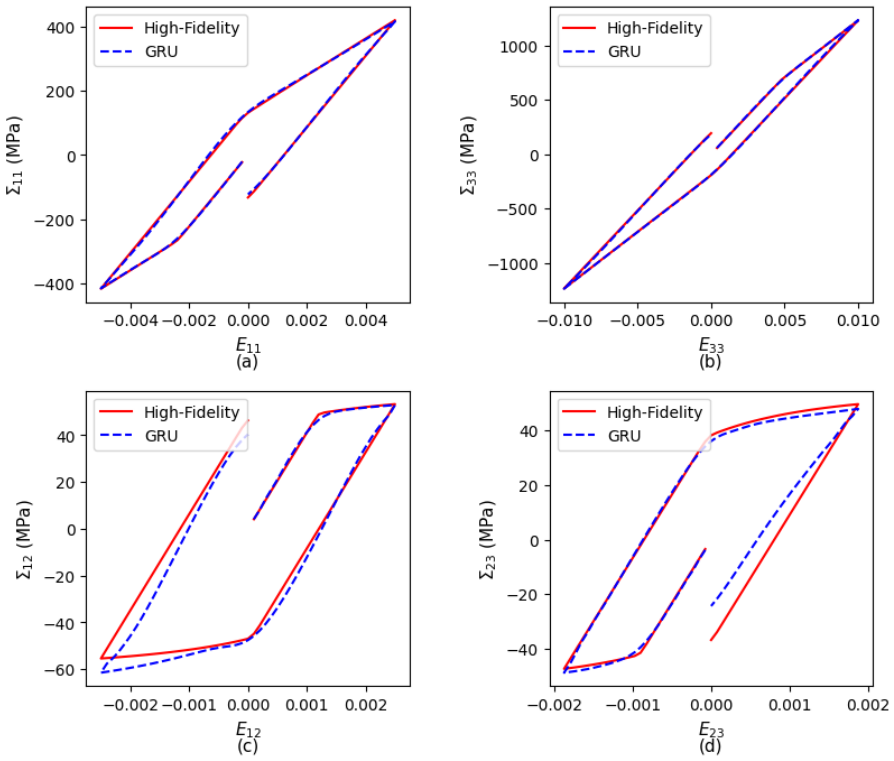


Figure 7.8. The predicted and true stress versus strain of cyclic triaxial strain state with nonzero shear strain. The strain state is shown in Figure 7.3c.

demonstrate the prediction of Σ_{33} is still the best, which is more accurate than the prediction of Σ_{11} and Σ_{22}. The lines of predictions of stress follow the line of the true value of stress very well. In addition, through the comparison of Figure 7.8a and Figure 7.8b with Figure 7.8c and Figure 7.8d, we can clearly see that our model has higher accuracy in predicting normal stress than shear stress in this case.

The Final strain state, shown in Figure 7.3d, is a sinusoidal version of the second case. The increasing and decreasing patterns of E_{11}, E_{22}, and E_{33} are the same as in case two. The peak value of E_{11} and E_{22} is ± 0.005, and the peak value of E_{33} is ± 0.01. However, in this case, our strain increment is not a constant and the strain series is no longer linear, but follows a sine function. Same as case 2, the value of E_{12}, E_{13}, and E_{23} is a constant and equal to 0. The number of time steps is equal to 100 in this state. The performance of our model in this strain state is illustrated in Figure 7.9. In this state, we also plot the true value and prediction of Σ_{11} and Σ_{33} versus the strain, which is illustrated in Figure 7.10.

In Figure 7.9 and Figure 7.10, the red line represents the true value in this case and the blue dashed line represents the prediction. Figure 7.10a and 7.10b illustrate Σ_{11} and Σ_{33}, respectively. The relative error in this case is 0.0250. Those two preceding figures demonstrate the prediction of Σ_{33} is still the best, which is more accurate than the prediction of Σ_{11} and Σ_{22}. Compared with the result of the second strain state, That the decrease in the accuracy of our model in this state is not significant.

Through testing at the microscale, we obtain the performance of our GRU model at a single integration point under certain strain states. These tests have preliminarily validated the capability of our model. In the next section, we reconstruct the trained GRU model in ABAQUS UMAT and use it for multiscale finite element simulation of actual macroscopic problems to verify the effectiveness of our model for practical problems.

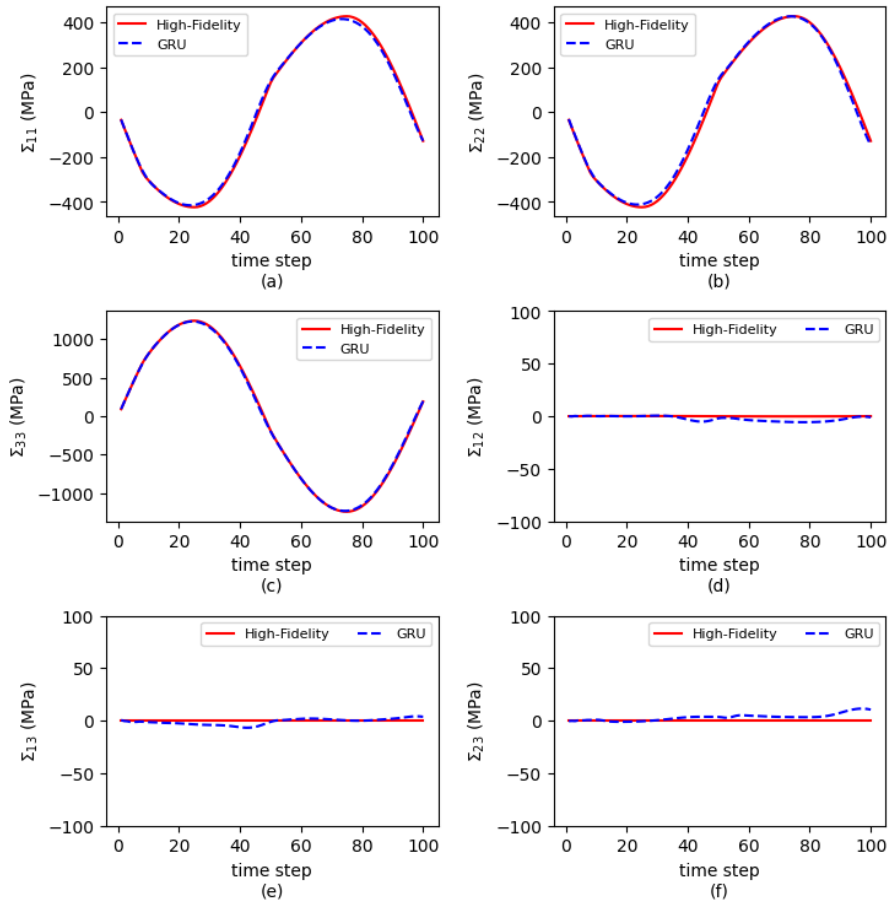


Figure 7.9. The predicted and true stress versus time steps of sinusoidal cyclic triaxial strain state. The strain state is shown in Figure 7.3d.

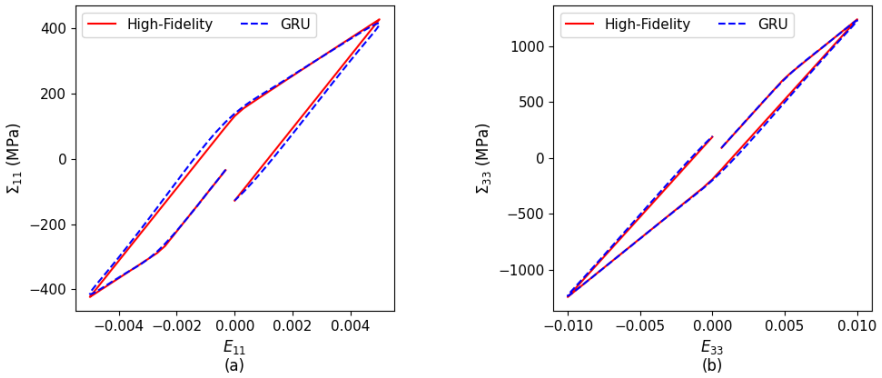


Figure 7.10. The predicted and true stress versus strain of sinusoidal cyclic triaxial strain state with nonzero shear stress. The strain state is shown in Figure 7.3d.

7.3 Test in Multiscale Problems

In this section, we implement our trained GRU model in a multiscale simulation framework to test its accuracy. The multiscale problems in this section are fundamental and are used to test the basic ability of the model in multiscale problems. In order to test our model, we design several problems of a multiscale cantilever beam and coupon made of long fiber composite materials, which is a carbon fiber aluminum alloy matrix composite material. The first problem is a tensile test on a cantilever beam, and the second problem is a bending test. In addition, we conduct tensile load tests with a coupon to test the ability of our model to deal with stress concentration. The geometry of our cantilever beam is illustrated in Figure 7.11, where the green parts of our model are fibers and the pale yellow part is matrix. The length of this cantilever beam is equal to 5, and the width and height of the interface are both equal to 1. The direction of fibers of this cantilever beam is along the z direction (or third direction).

We divide each set of tests into two steps. The first step is to generate a cantilever beam with the same microstructure as the real material and use a fine mesh to perform high-fidelity traditional finite element simulation on the cantilever beam, using the simulation results of this step as a true value (reference value). In this case, we directly use the built-in material model of ABAQUS to create material properties for the fibers and matrix of the cantilever beam, respectively. The second step is to generate a cantilever beam with only a macroscopic geometric structure and use the UMAT code containing our trained GRU model as a material subroutine for multiscale simulation. In this case, we use very coarse mesh to implement simulation. Finally, we compare the results of these two simulations and obtain the accuracy of our trained GRU model. In addition, we investigate the performance of our model at the integration point.

In this section, we always apply a surface traction force load to our model and draw displacement contour plots to compare the distribution of displacement fields in two tests, which is to examine the overall performance of our model. In addition, we select the integration points

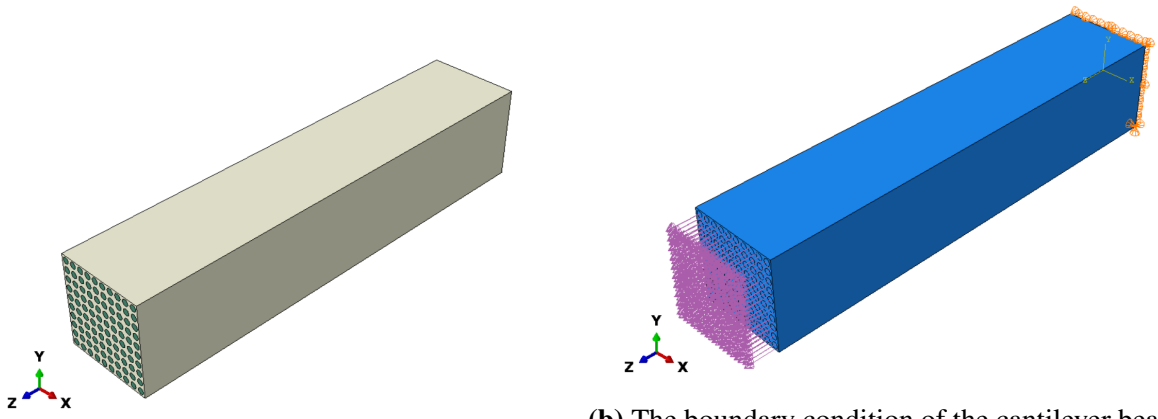
with stress concentration or those that we are interested in, plot their stress components, and compare them with the true values. The true value is obtained through high-fidelity simulation using the strain data obtained from those integration points.

In this work, we assume l is the dimension of our RVE model and L is the shortest side length of the cross-section perpendicular to the fiber direction in the high-fidelity model. In all high-fidelity simulations, the default value of l/L is 0.25. In some numerical problems, we also use smaller ratios, such as 0.1, for high-fidelity simulations and comparisons to verify the accuracy of high-fidelity simulations.

In the following test, we plot and compare the maximum absolute value of displacement of each group of test results. It should be pointed out that in the following text when we mention the maximum displacement, it defaults to the maximum absolute value of displacement. Because in some cases the displacement may be negative, this statement is made to avoid confusion. In our contour plots, we call the displacement in the x-direction (or direction 1) U_1, the displacement in the y-direction (or direction 2) U_2, and the displacement in the z-direction (or direction 3) U_3.

7.3.1 Cantilever Beam Tensile Test

In this test, we examine the performance of our trained GRU model in uniaxial tension simulation of a composite cantilever beam. The boundary conditions of high-fidelity simulation and multiscale simulation are the same, which are illustrated in Figure 7.11b. In the z-positive surface, there is surface traction along the z-positive direction, the magnitude is equal to 1600 MPa, and the number of time steps is equal to 50. In the z-negative surface, we set the displacement of the entire surface in the z-direction equal to 0 and set appropriate boundary conditions to prevent rigid body motion. Meanwhile, we use a coupling constraint on the z-positive plane to keep all nodes on this surface flat. In the high-fidelity simulation, we need fine mesh and the mesh size equal to 0.05 and 0.02 while in the multiscale simulation, we use coarse mesh and the mesh size equal to 0.5.



(a) The geometry of the cantilever beam

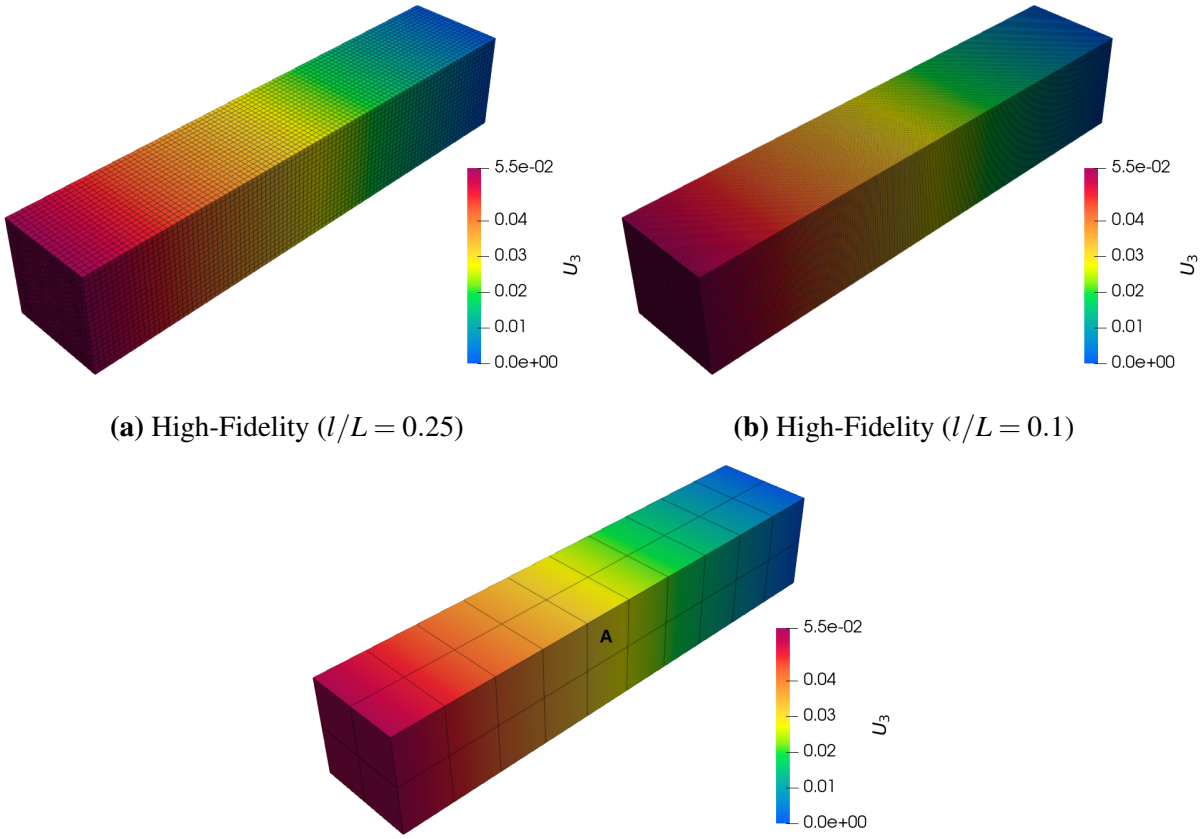
(b) The boundary condition of the cantilever beam tensile test

Figure 7.11. The geometry and boundary condition of the cantilever beam in the tensile test ($l/L = 0.1$).

In this paper, we use ParaView to generate contour plots of our simulations. Shen et al. developed a software that can transform the Odb files from ABAQUS to Vtu files that can be used for ParaView in their paper [78], which is used in this research. The displacement distributions in the z-direction of high-fidelity simulation are illustrated in Figure 7.12a and 7.12b when the l/L is equal to 0.25 and 0.1, respectively. The displacement distribution in the z-direction of multiscale simulation is illustrated in Figure 7.12c. Even if we employ a significantly coarser mesh compared to that used in high-fidelity simulations within multiscale simulations, the distribution of displacement in the z-direction is very similar in both simulations. The errors of maximum displacement between the two simulations are 0.0277 and 0.0279 when the l/L is equal to 0.25 and 0.1, respectively, which are very close. Due to the uniform distribution of stress and strain in the cantilever beam under this problem, we omitted the contour plots in this case.

The reason we don't compare the stress distribution of high-fidelity and multiscale simulation is that stress directly concentrates on fibers in the high-fidelity case. In real-world situations, the fiber is extremely smaller than the macroscopic geometry, so the fiber does not affect the distribution of stress. However, if we use the fiber size of real-world situations, it makes our simulation unfeasible. So, in this paper, we compare the distribution of displacement

instead of stress.



(a) High-Fidelity ($l/L = 0.25$)

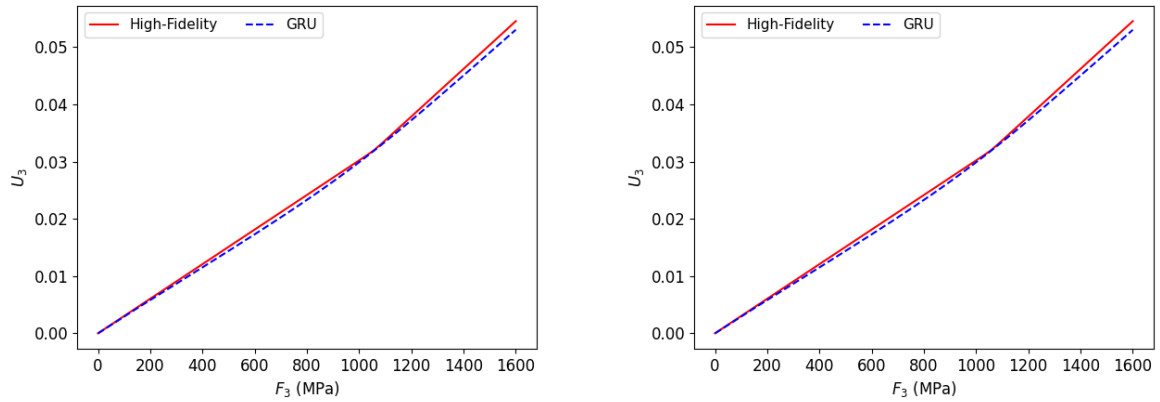
(b) High-Fidelity ($l/L = 0.1$)

(c) GRU

Figure 7.12. The comparison of displacement fields of high-fidelity and multiscale simulation in a tensile test of a cantilever beam.

In order to compare the true value and prediction of the maximum displacement in the z-direction with changes in load, we plot the maximum displacement in the z-direction of true value and prediction versus the load applied to the cantilever beam, which is illustrated in Figure 7.13a. In addition, we select an element in the model and name it "A", which is shown in Figure 7.12c. We extract the historical strain and stress data of integral point 1 of element A. It should be pointed out that the stress data from this integration point is the prediction of the GRU model. Meanwhile, we use strain data to perform RVE high-fidelity simulation and homogenization to obtain the true value of stress. We can directly call the preprocess, postprocess, and homogenization codes we designed to complete this process. Then we plot the

predicted value and true value of the six stress components separately for comparison, which is illustrated in Figure 7.14. The red line represents the true value in this case and the blue dashed line represents the prediction of our GRU model. At this integration point, the relative error of the trained GRU model is 0.0125.



(a) $l/L = 0.25$ **(b)** $l/L = 0.1$

Figure 7.13. The maximum displacement in the z-direction versus the load applied to the cantilever beam (tensile).

7.3.2 Cantilever Beam Bending Test

In this example, we test the performance of our trained GRU model in the bending simulation of a composite cantilever beam. The boundary condition of high-fidelity simulation and multiscale simulation is the same, which is illustrated in Figure 7.15. In the z-positive surface, there is surface traction along the y-negative direction and the magnitude is equal to 100 MPa. In the z-negative surface, we set the displacement of all the nodes in the entire surface in the x, y, and z-direction equal to 0, which means we fixed the entire z-negative surface of our simulation model. In the high-fidelity simulation, we need fine mesh and the mesh size equal to 0.05 and 0.02 while in the multiscale simulation, we use coarse mesh and the mesh size equal to 0.25.

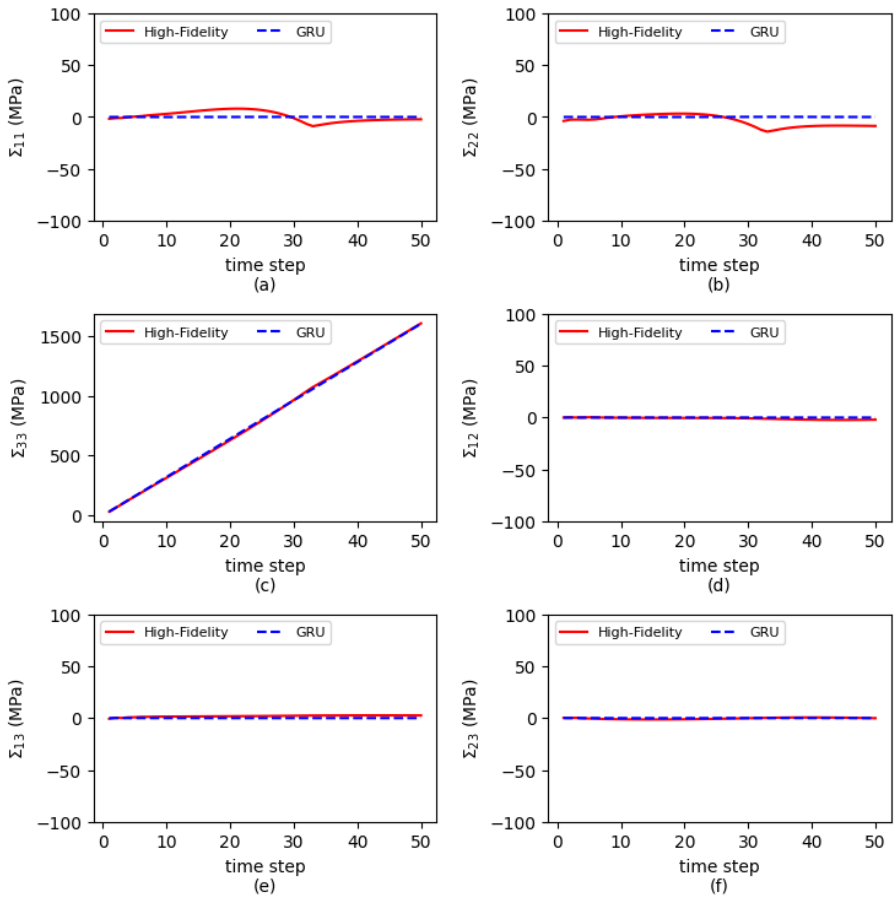


Figure 7.14. The true value and stress prediction at an integration point versus time steps in a cantilever beam (tensile).

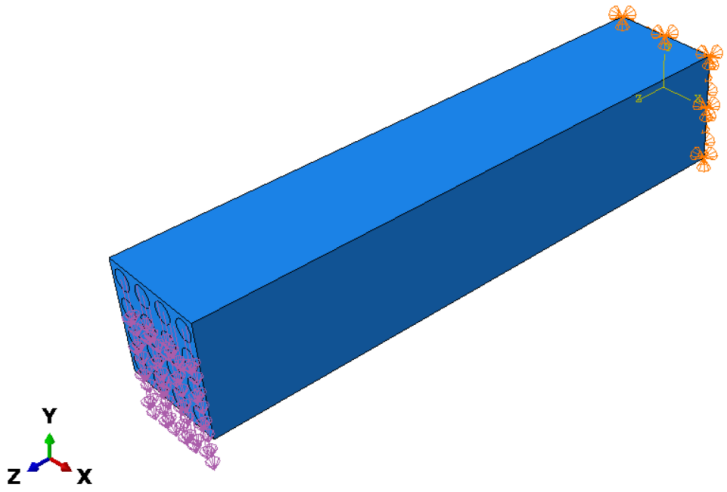
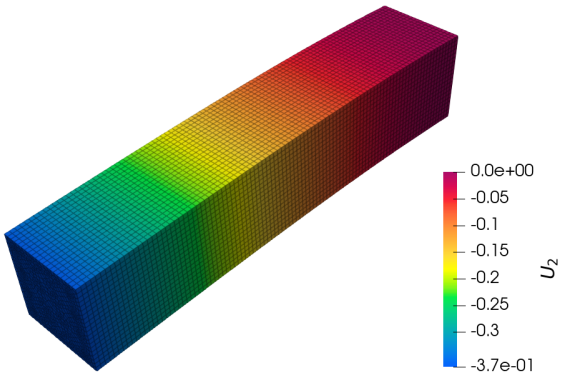


Figure 7.15. The boundary condition of the cantilever beam bending test ($l/L = 0.25$).

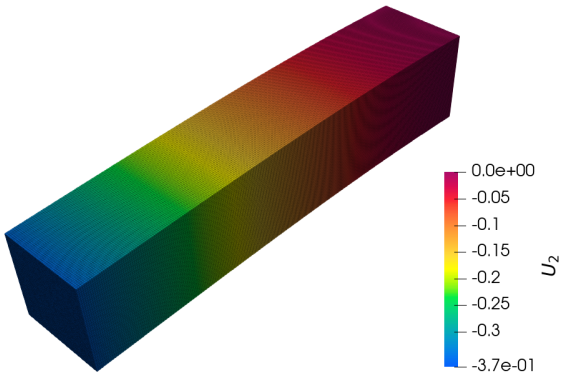
The displacement distributions in the y-direction of high-fidelity simulation are illustrated in Figure 7.16a and 7.16b when the l/L is equal to 0.25 and 0.1, respectively. The displacement distribution in the y-direction of multiscale simulation is illustrated in Figure 7.16c. Even if we employ a significantly coarser mesh compared to that used in high-fidelity simulations within multiscale simulations, the distribution of displacement in the y-direction is very similar in both simulations. The errors of maximum displacement between the two simulations are 0.0549 and 0.0221 when the l/L is equal to 0.25 and 0.1, respectively. It is very clear, that when the l/L is decreased the error decreases, since the accuracy of high-fidelity simulation increases. The contour plots of von Mises stress and von Mises strain of this problem are illustrated in Figure 7.17a and Figure 7.17b, respectively.

It should be pointed out that ABAQUS does not have built-in options to generate contour plots of von Mises strain. So, we need to develop a UAVRM user subroutine to enable ABAQUS to compute and save von Mises strain data. ABAQUS UVARM (User-defined Variables at Material points) is a user-defined variable feature in ABAQUS that allows users to define and output additional variables at material points. These user-defined variables can be used to track non-built-in variables in simulations. The ABAQUS documentation provides a detailed explanation of this subroutine. It should be noted that when we need to use UVARM, we need to declare the number of user-defined variables in the material module of ABAQUS, and also require the output of user-defined output variables in the field output request module.

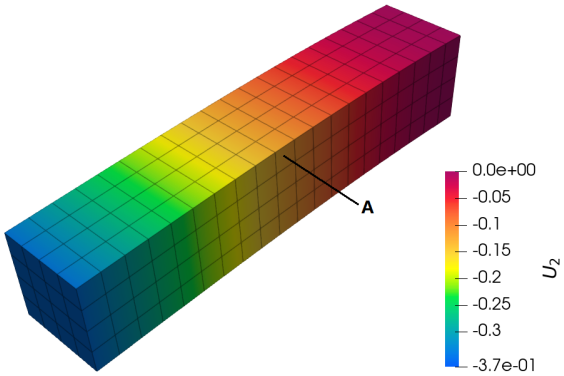
In order to compare the true value and prediction of the maximum displacement in the y-direction with changes in load, we plot the maximum displacement in the y-direction of true value and prediction versus the load applied to the cantilever beam, which is illustrated in Figure 7.18a. Meanwhile, we select an element in the model and name it “A”, which is shown in Figure 7.16c. We extract the historical strain and stress data of integral point 1 of element A. It should be pointed out that the stress data from this integration point is the prediction of the GRU model. Meanwhile, we use strain data to perform RVE high-fidelity simulation and homogenization to obtain the true value of stress. Then we plot the predicted value and true value of the six stress



(a) High-Fidelity ($l/L = 0.25$)

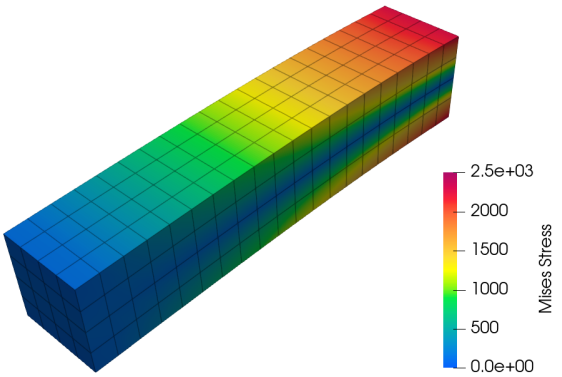


(b) High-Fidelity ($l/L = 0.1$)

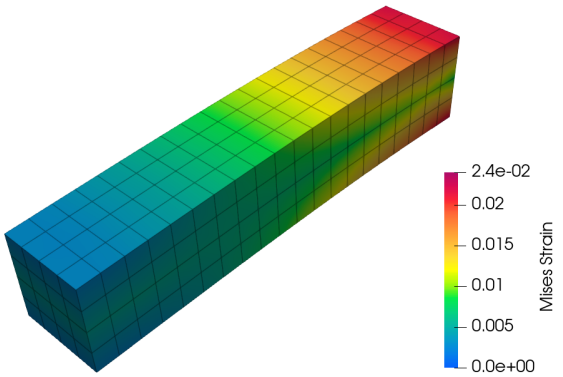


(c) GRU

Figure 7.16. The comparison of displacement fields of high-fidelity and multiscale simulation in a bending test of a cantilever beam.



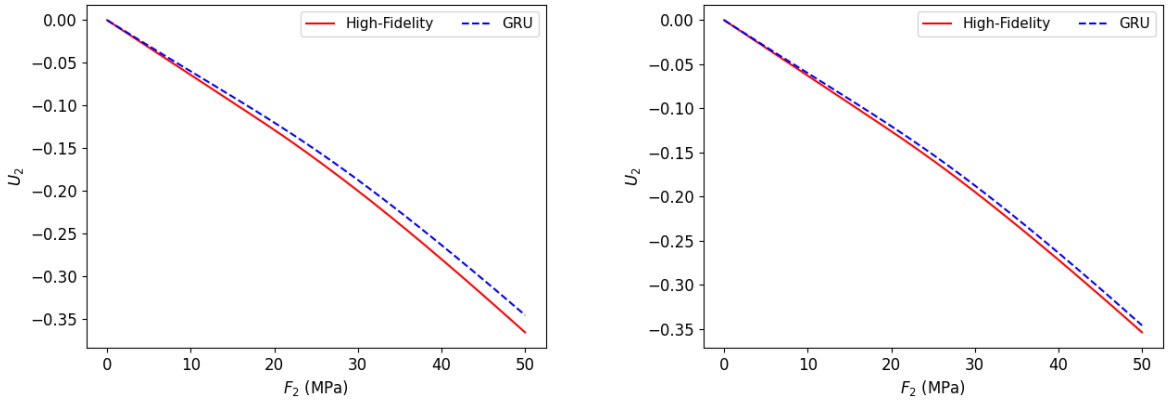
(a) von Mises stress



(b) von Mises strain

Figure 7.17. The von Mises stress and strain fields of multiscale simulation in a bending test of a cantilever beam.

components separately for comparison, which is illustrated in Figure 7.19. At this integration point, the relative error of the model is 0.020.



(a) $l/L = 0.25$ **(b)** $l/L = 0.1$

Figure 7.18. The maximum displacement in the y-direction versus the load applied to the cantilever beam (bending).

7.3.3 Coupon Tensile Test

To demonstrate that our surrogate model can still perform very well in simulations with stress concentration, we designed a new test. In this case, we test the performance of our trained GRU model in the tensile simulation of a coupon. The geometry of this coupon is illustrated in Figure 7.20. There are two semi-circular openings on both sides of the coupon, with a radius of 0.25 for each opening. The centers of both openings are located at the midpoint of the coupon in the z-direction.

The boundary conditions of high-fidelity simulation and multiscale simulation are the same, which is illustrated in Figure 7.20. In the z-positive surface, there is surface traction along the z-positive direction and the magnitude is equal to 600 MPa. In the z-negative surface, we set the displacement of the entire surface in the z-direction to 0 and set appropriate boundary conditions to prevent rigid body motion. Meanwhile, we use a coupling constraint on the

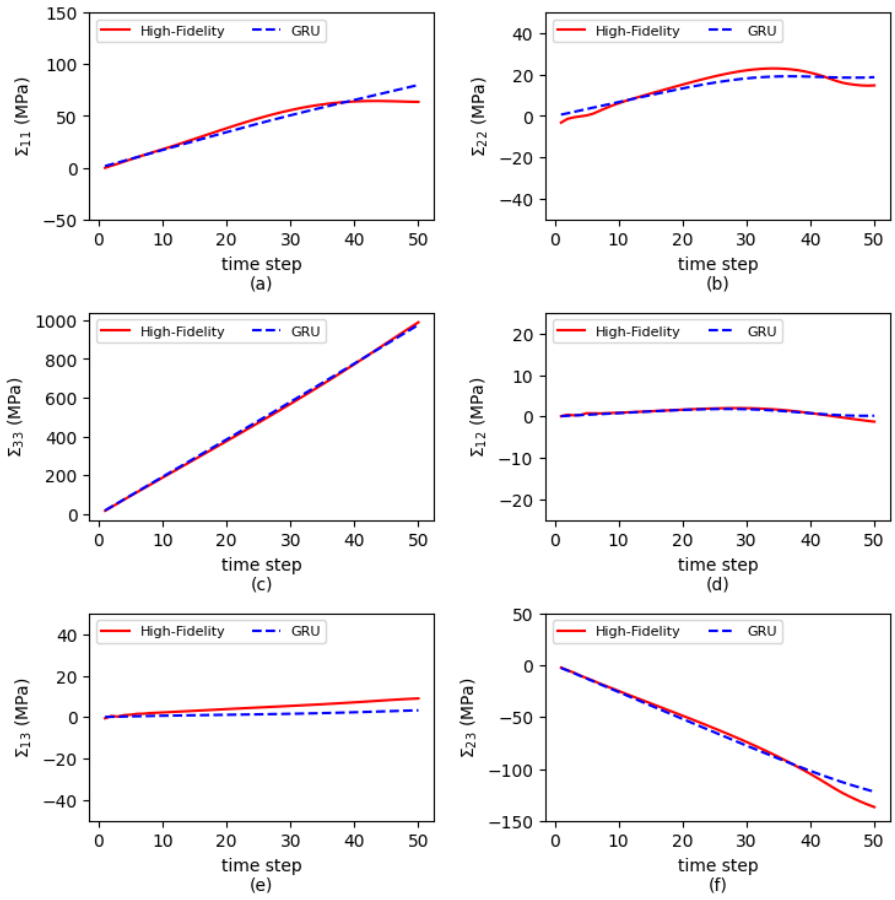


Figure 7.19. The true value and stress prediction at an integration point versus time steps in a cantilever beam (bending).

z-positive plane to keep all nodes on this surface flat. In this case, in order to fit the more complex geometry and internal structure of our coupon model, we use tetrahedral mesh for the high-fidelity case and we still use hexahedral mesh for the multiscale case. In the high-fidelity simulation, we use fine mesh and the mesh size equal to 0.1, while in the multiscale simulation, we use coarse mesh and the mesh size equal to 0.15.

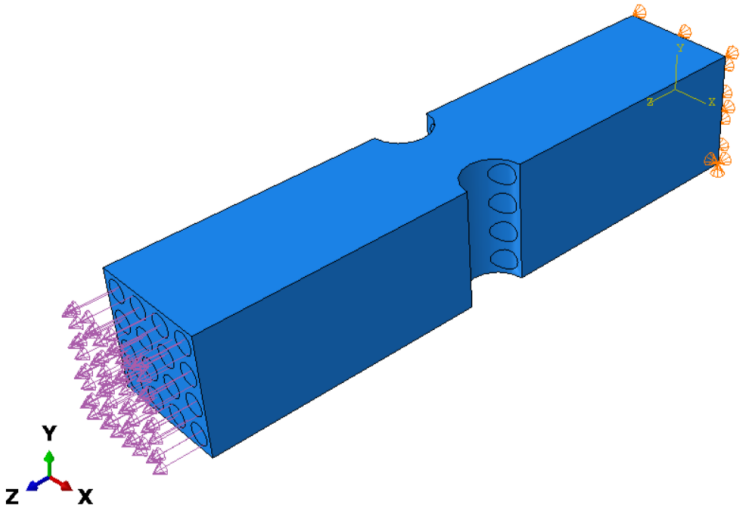
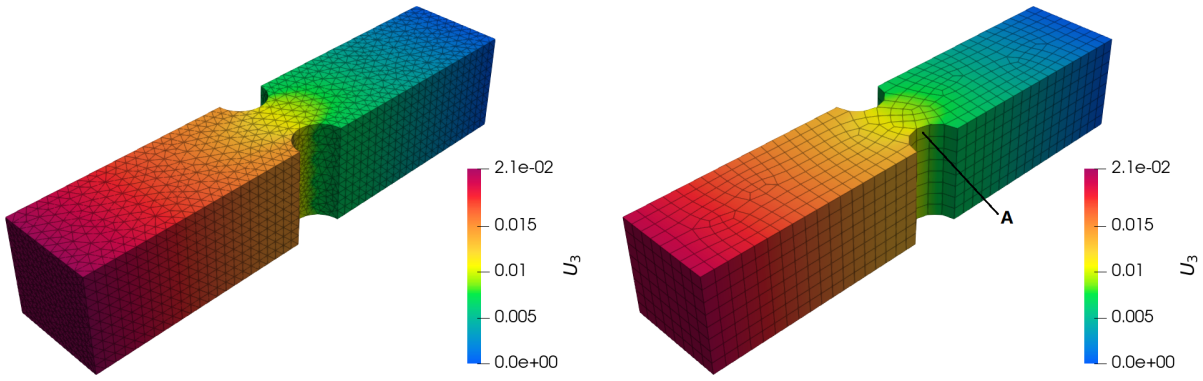


Figure 7.20. The boundary condition of the coupon tensile test ($l/L = 0.25$).

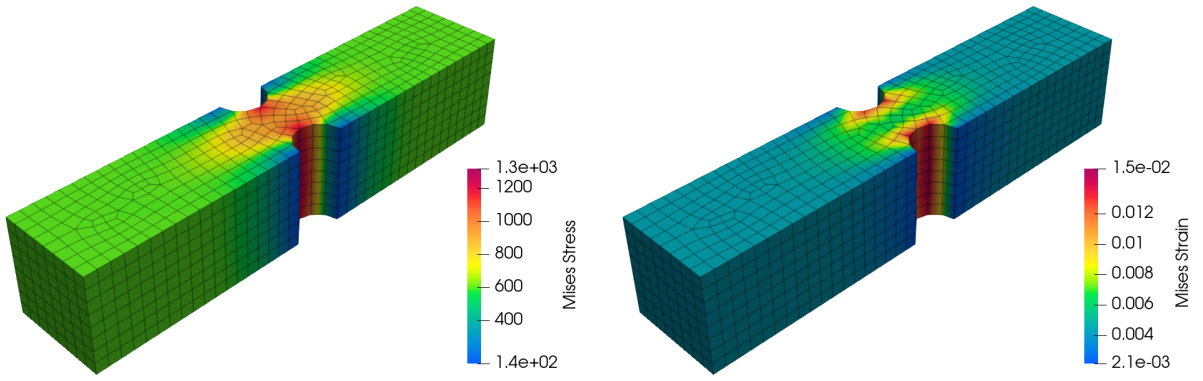
The displacement distribution in the z-direction of high-fidelity simulation is illustrated in Figure 7.21a and the displacement distribution in the z-direction of multiscale simulation is illustrated in Figure 7.21b. Even if we employ a significantly coarser mesh compared to that used in high-fidelity simulations within multiscale simulations, the distribution of displacement in the z-direction is very similar in both simulations. The relative error of maximum displacement between the two simulations is 0.053. The contour plots of von Mises stress and von Mises strain of multiscale simulation of this problem are illustrated in Figure 7.22a and Figure 7.22b, respectively.

In order to compare the true value and prediction of the maximum displacement in the z-direction with changes in load, we plot the maximum displacement in the z-direction of true value and prediction versus the load applied to the coupon, which is illustrated in Figure 7.23. In addition, we select an element in the model and name it “A”, which is shown in Figure 7.21b.



(a) High-Fidelity **(b)** GRU

Figure 7.21. The comparison of displacement fields of high-fidelity and multiscale simulation of a tensile test of a coupon.



(a) von Mises stress **(b)** von Mises strain

Figure 7.22. The von Mises stress and strain fields in multiscale simulation of a tensile test of a coupon.

We extract the historical strain and stress data of integral point 1 of element A. It should be pointed out that the stress data from this integration point is the prediction of the GRU model. Meanwhile, we use strain data to perform RVE high-fidelity simulation and homogenization to obtain the true value of stress. Then we plot the predicted value and true value of the six stress components separately for comparison, which is illustrated in Figure 7.24. At this integration point, the relative error of the model is 0.0145.

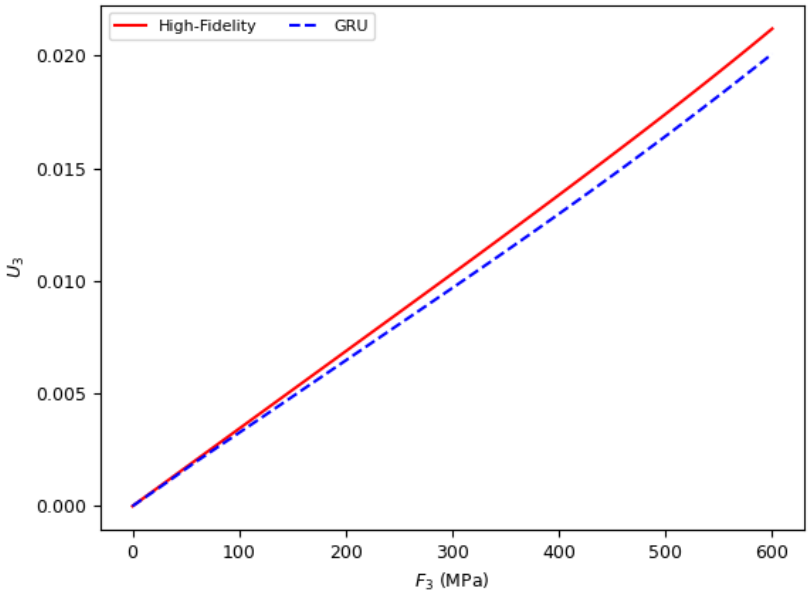


Figure 7.23. The maximum displacement in the z-direction versus the traction applied to the coupon.

7.4 Test in Complex Multiscale Problems

In this section, we test our trained GRU model with a collection of more complex boundary value problems, which is to verify the ability of our model to work in real-world cases with complex geometry and load. We implement a cyclic loading of a long fiber composite thick plate with a circular perforation, the 2D plane projection of the model along the opening direction

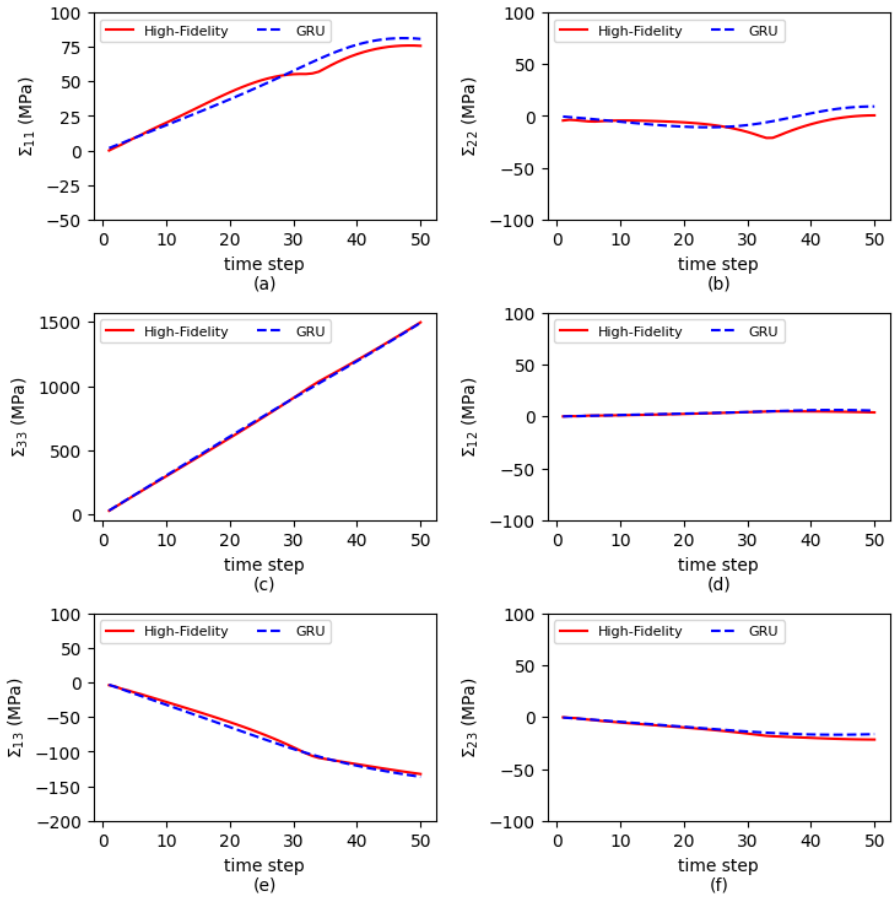
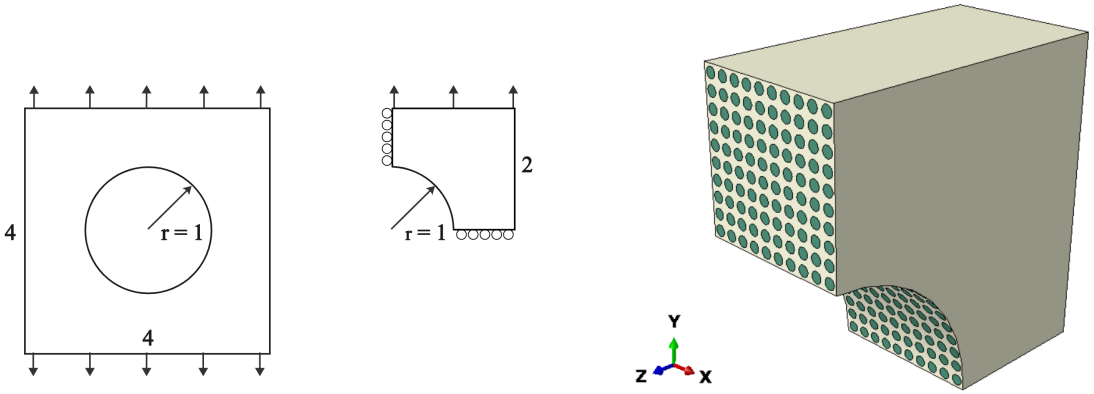


Figure 7.24. The true value and stress prediction at an integration point versus time steps of a coupon.

is illustrated in Figure 7.25a. By symmetry, only one-quarter of the domain is modeled as shown in the right-hand side of Figure 7.25a, which can reduce the computation load in our simulation. The 3D model of the simplified part of this thick plate is illustrated in Figure 7.25b. The length and width of this thick plate are both 4, with a thickness of 1. The circular perforation is located at the center of this plate, with the direction of perforation perpendicular to the direction of fiber extension, and the radius of perforation is 1. The direction of the fibers of this plate is along the z-direction (direction 3).



(a) The 2D plane projection of the thick composite plate model

(b) The 3D model of the one-quarter part of the thick plate ($l/L = 0.1$)

Figure 7.25. Geometry of thick plate with a circular perforation.

The upper and lower parts of this thick plate are subjected to uniform cyclic surface traction force or displacement. The load or displacement changes in the y-direction, similar to the protocol of E_{33} in Figure 7.3b or the protocol of E_{33} in Figure 7.3d, with a positive initial value. In the mesh of the thick plate model, the elements and nodes of interest are illustrated in Figure 7.26. The selection of these elements is based on the locations where stress concentrations occur.

Similar to the previous section, we divide each set of tests into two steps. The first step is to generate one-quarter of the thick plate with the same microstructure as the real material and use a fine mesh to perform high-fidelity traditional finite element simulation on the model, using the simulation results of this step as true value (reference value). In this case, we directly use the

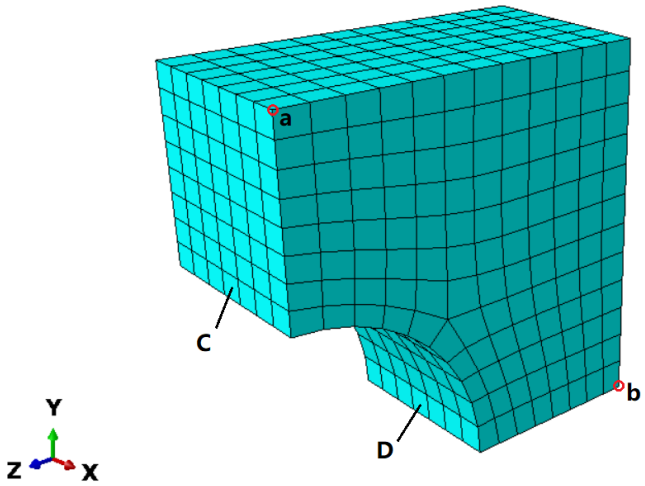


Figure 7.26. The mesh of multiscale simulations, along with the elements and nodes of interest in the thick plate model.

built-in material model of ABAQUS to create material properties for the fibers and matrix of the model, respectively. The second step is to generate one-quarter of the thick plate with only a macroscopic geometric structure and use the UMAT code containing our trained GRU model as a material subroutine for multiscale simulation. In this case, we use a coarser mesh to implement the simulation. Finally, we compare the results of these two simulations and obtain the accuracy of our trained GRU model. In addition, we investigate the performance of our model at the integration point.

7.4.1 Cyclic Surface Traction Force Test

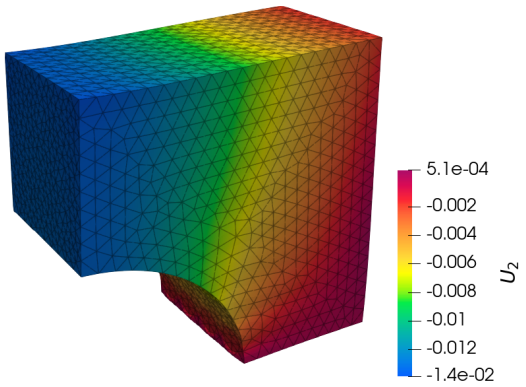
In this subsection, we verify the performance of our trained GRU model in cyclic surface traction load for a thick composite plate with a hole. The boundary condition of high-fidelity simulation and multiscale simulation is the same, which is illustrated in Figure 7.25a. On the upper surface of the model, there is surface traction along the y direction and the maximum and minimum magnitude is equal to ± 200 MPa, and the number of time steps is equal to 100. The pattern of the traction load is the same as the protocol of E_{33} in Figure 7.3b. In this case, we use

tetrahedral mesh for the high-fidelity case and hexahedral mesh for the multiscale case. In the high-fidelity simulation, we use fine mesh, and the mesh size is set to 0.1, while in the multiscale simulation, we use coarse mesh, and the mesh size is set to 0.15.

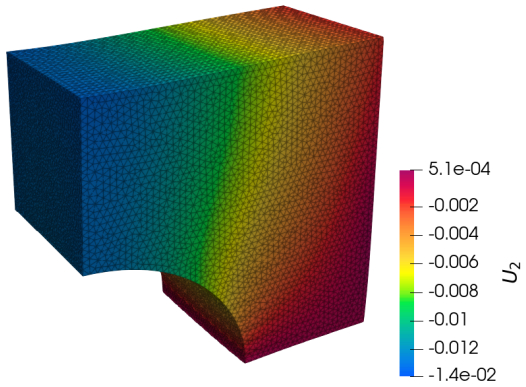
The displacement distribution in the y-direction of high-fidelity simulation and multiscale simulation in the 75th time step is illustrated in Figure 7.27a and Figure 7.27c respectively. It should be pointed out that, in the 75th time step, the traction achieves minimum value. The displacement distribution in the z-direction of high-fidelity simulation and multiscale simulation in the 75th time step is illustrated in Figure 7.28a and Figure 7.28c respectively. Even if we employ a significantly coarser mesh compared to that used in high-fidelity simulations within multiscale simulations, the distribution of displacement in the z-direction is very similar in both simulations. The relative error of maximum displacement in the y-direction between the two simulations is 0.0093 and the relative error of maximum displacement in the z-direction is 0.012. The displacement distribution in the y-direction and z-direction of high-fidelity simulation in the 75th time step is illustrated in Figure 7.27b and Figure 7.28b respectively when l/L is equal to 0.1. The relative errors of maximum displacement in this case are equal to 0.032 and 0.067, respectively.

The distribution of the von Mises stress of multiscale simulation in the 75th time step of this case is illustrated in Figure 7.29a. This figure illustrates the location of stress concentration. The distribution of the von Mises strain of multiscale simulation in the 75th time step of this case is illustrated in Figure 7.29b.

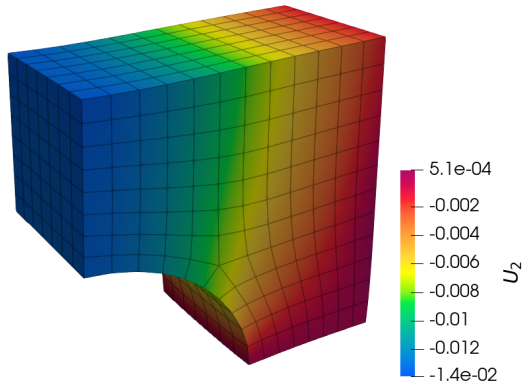
The displacement in the y-direction of node “a” versus the traction applied to the plate and the displacement in the z-direction of node “b” are illustrated in Figure 7.30. In the figure, the result when l/L is equal to 0.25 is shown by a gray line, and the result when l/L is equal to 0.1 is shown by a black line. When l/L equals 0.25, the relative error of U_2 is 0.066, and when l/L equals 0.1, the relative error is 0.070. According to the position of stress concentration, we select integration point 7 of element “C” to plot the true value and prediction of stress components. As before, we use the result of RVE simulation and homogenization as the true value of stress. The



(a) High-Fidelity ($l/L = 0.25$)

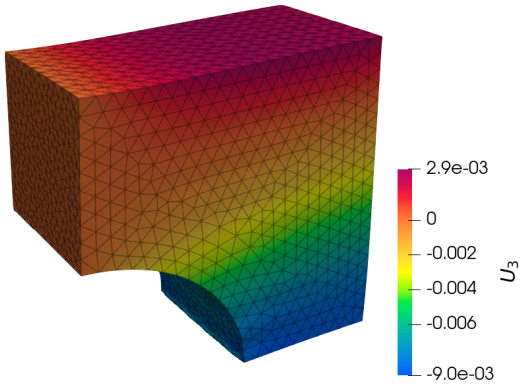


(b) High-Fidelity ($l/L = 0.1$)

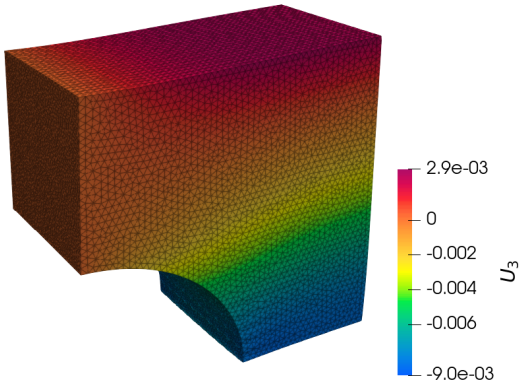


(c) GRU

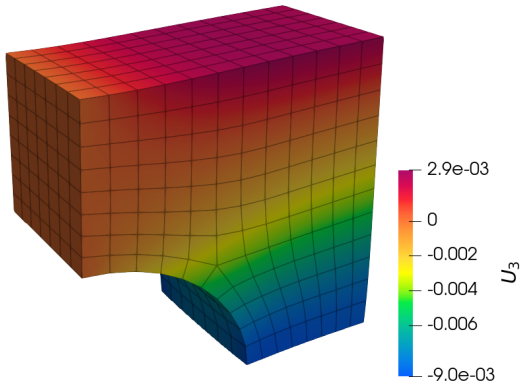
Figure 7.27. The comparison of the displacement distribution in y-direction of high-fidelity and multiscale simulation in the 75th time step (cyclic surface traction force).



(a) High-Fidelity ($l/L = 0.25$)

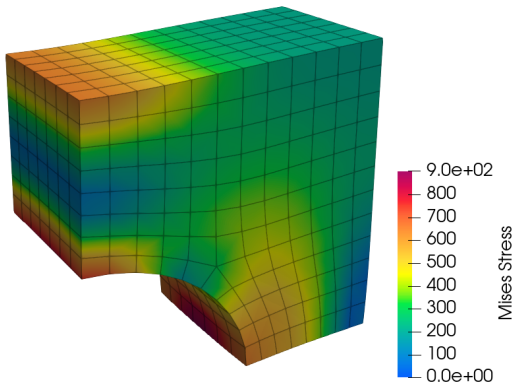


(b) High-Fidelity ($l/L = 0.1$)

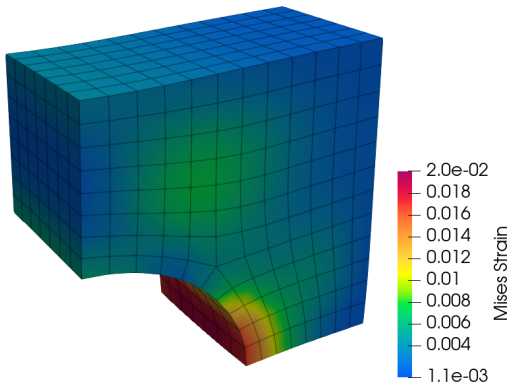


(c) GRU

Figure 7.28. The comparison of the displacement distribution in the z-direction of high-fidelity and multiscale simulation in the 75th time step (cyclic surface traction force).



(a) von Mises stress



(b) von Mises strain

Figure 7.29. The von Mises stress and strain distribution of multiscale simulation in the 75th time step (cyclic surface traction force).

result is illustrated in Figure 7.31. The relative error of our GRU model at this integration point is equal to 0.051. The positions of those nodes and elements are illustrated in Figure 7.26.

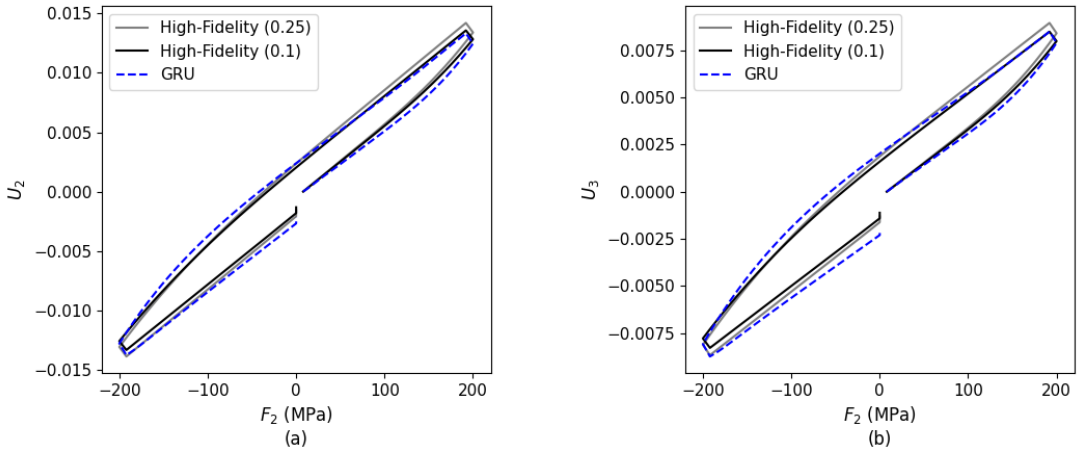


Figure 7.30. The displacement in y-direction of node "a" and the displacement in z-direction of node "b" versus time step (cyclic surface traction force).

7.4.2 Cyclic Surface Displacement Test

In this test, we replace the surface traction load with displacement to test the performance of our model. The boundary condition of high-fidelity simulation and multiscale simulation is the same, which is illustrated in figure 7.25a. On the upper surface of the model, there is uniform displacement along the y direction with the maximum and minimum magnitude equal to ± 0.01. Additionally, the number of time steps is set to 100. The pattern of the load is the same as the protocol of E_{33} in Figure 7.3b. In this case, we use tetrahedral mesh for the high-fidelity case and hexahedral mesh for the multiscale case. In the high-fidelity simulation, we need fine mesh, and the mesh size is set to 0.1, while in the multiscale simulation, we use coarse mesh, and the mesh size is set to 0.15.

The distribution of the von Mises stress and strain of multiscale simulation in the 75th time step of this case is illustrated in Figure 7.32a and Figure 7.32b, respectively. This figure

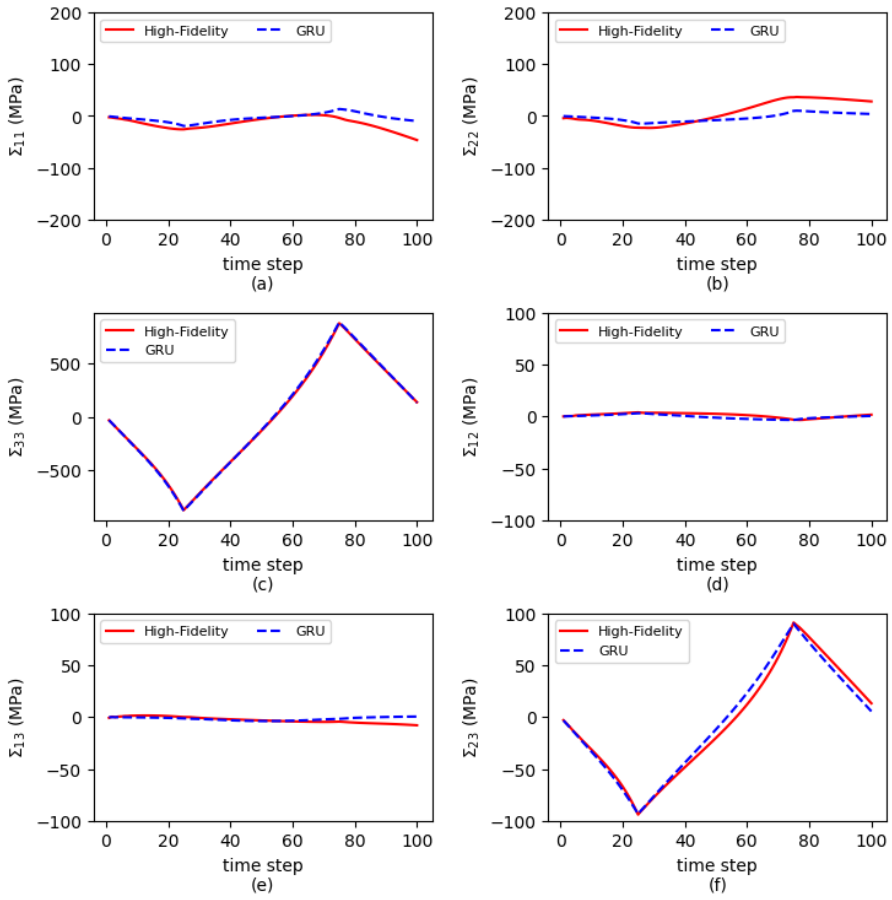
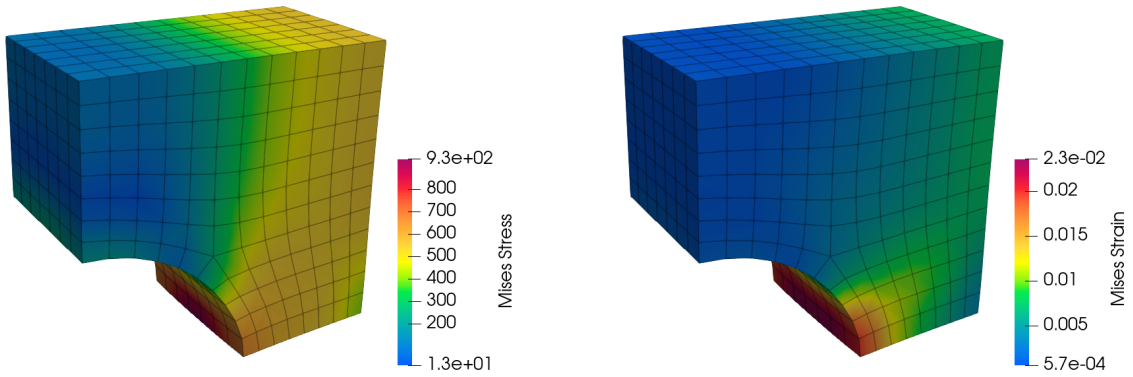


Figure 7.31. The predicted and true stress of integration point 7 of element "C" versus time steps (cyclic surface traction force).

illustrates the location of stress concentration. We don't show the displacement contour plots in this case since the load itself is displacement.



(a) von Mises stress

(b) von Mises strain

Figure 7.32. The von Mises stress and strain distribution of multiscale simulation in the 75th time step (cyclic surface displacement).

In Figure 7.33, we plot the true value and prediction of reaction force in the upper surface of the model. The error between the true value and prediction is 0.082. According to the position of stress concentration, which is illustrated in Figure 7.32a, we select integration point 8 of element "D" to plot the true value and prediction of stress components. As before, we use the result of RVE simulation and homogenization as the true value of stress. The result is illustrated in Figure 7.34. The relative error of our GRU model in this integration point is about 10 %.

It should be pointed out that, the high-fidelity simulation itself introduces some error in our reference value. If we use a finer mesh to do high-fidelity simulation, the error is smaller than the coarse mesh. According to our testing, when we reduce the mesh size to half of its original size, the relative error is reduced by approximately 0.3 % to 0.8 %. Our initial value of mesh size in RVE simulation is 0.075. However, using finer mesh to perform simulation brings a much greater computational cost to our simulation.

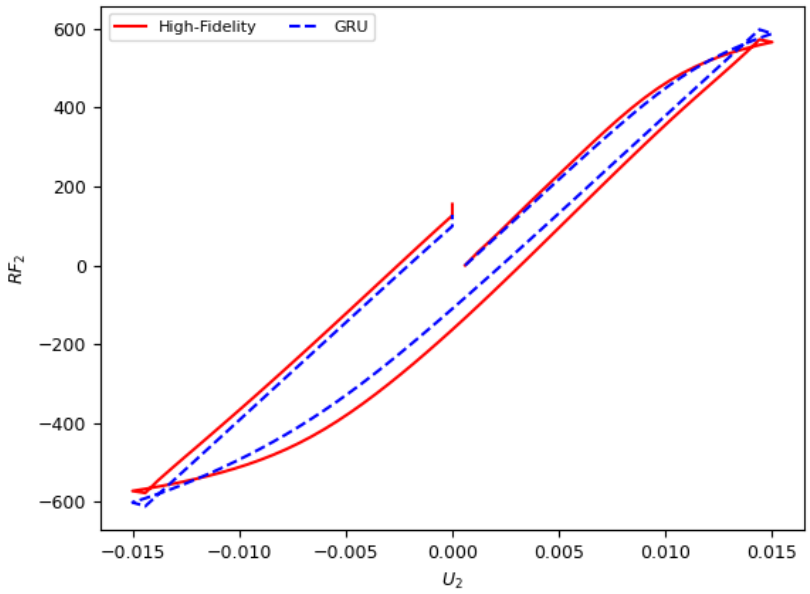


Figure 7.33. The predicted and true reaction force in the upper surface of the model versus cyclic surface displacement.

7.4.3 Sinusoidal Surface Traction Force Test

In the previous tests, our loading pattern is linear. In this subsection, in order to make the problem more complex, we test the performance of our trained GRU model in sinusoidal surface traction load, which validates the ability of our model to deal with nonlinear loading. The boundary condition of high-fidelity simulation and multiscale simulation is the same, which is illustrated in Figure 7.25a. On the upper surface of the model, there is surface traction along the y direction and the maximum and minimum magnitude is equal to $\pm$200 MPa, and the number of time steps is equal to 100. The pattern of the traction load is the same as the protocol of E_{33} in Figure 7.3d. In this case, we use tetrahedral mesh for the high-fidelity case and hexahedral mesh for the multiscale case. In the high-fidelity simulation, we need fine mesh, and the mesh size is set to 0.1, while in the multiscale simulation, we use coarse mesh, and the mesh size is set to 0.15.

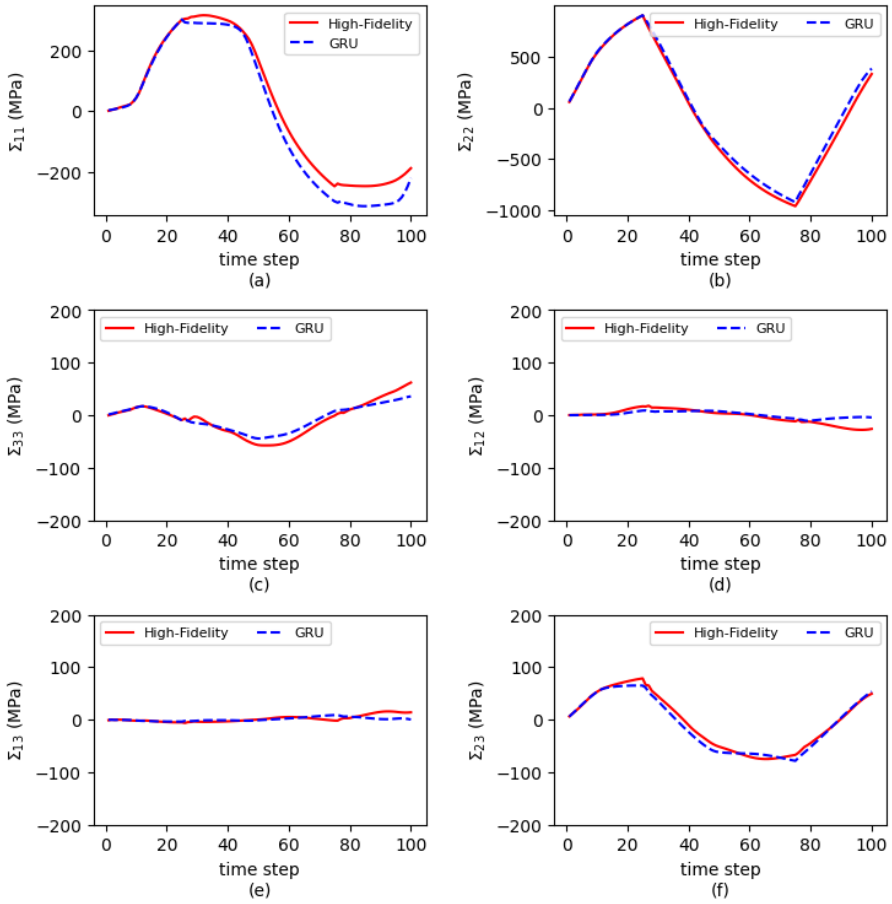
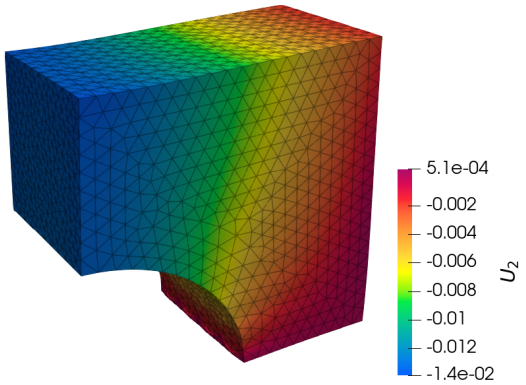
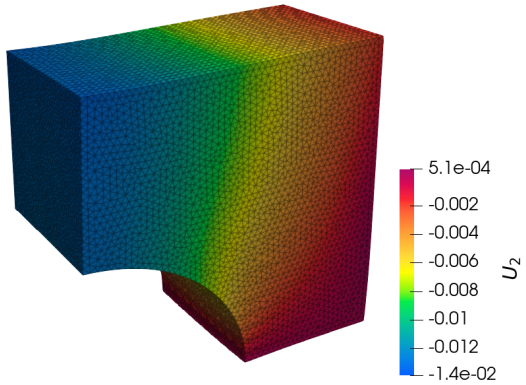


Figure 7.34. The predicted and true stress at integration point 8 of element “D” versus time steps (cyclic surface displacement).

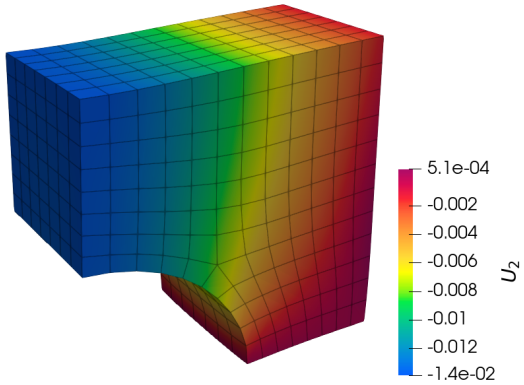
The displacement distributions in the y-direction and z-direction for both high-fidelity and multiscale simulations at the 75th time step are illustrated in Figures 7.35a and 7.35c, and Figures 7.36a and 7.36c, respectively. Despite using a significantly coarser mesh in multiscale simulations compared to high-fidelity simulations, the displacement distribution in the z-direction remains very similar in both cases. The maximum displacement error in the y-direction between the two simulations is 0.0122, while in the z-direction, it is 0.0324. The displacement distribution in the y-direction and z-direction of high-fidelity simulation in the 75th time step is illustrated in Figure 7.35b and Figure 7.36b respectively when l/L is equal to 0.1. The errors of maximum displacement in this case are equal to 0.055 and 0.083, respectively.



(a) High-Fidelity ($l/L = 0.25$)

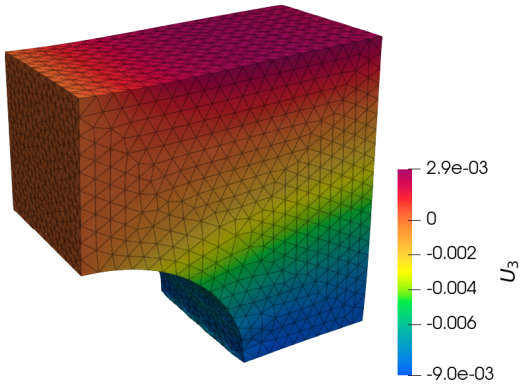


(b) High-Fidelity ($l/L = 0.1$)

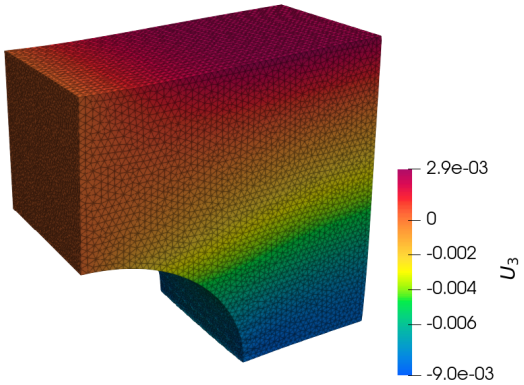


(c) GRU

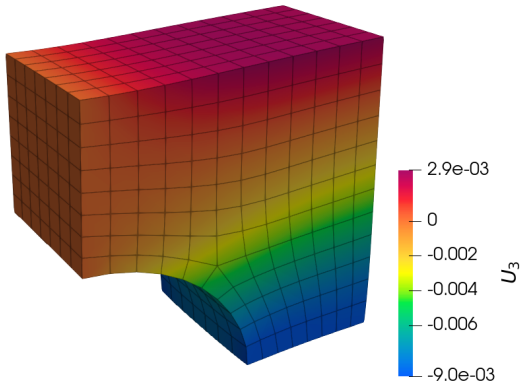
Figure 7.35. The comparison of the displacement distribution in y-direction of high-fidelity and multiscale simulation in the 75th time step (sinusoidal surface traction force).



(a) High-Fidelity ($l/L = 0.25$)

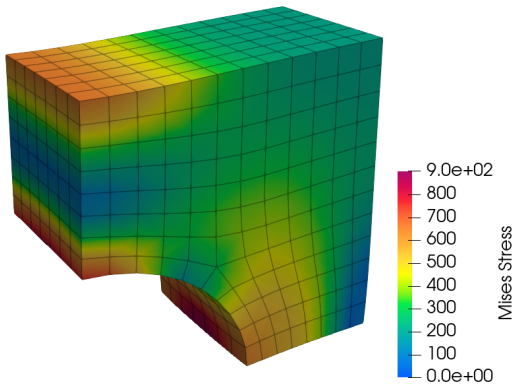


(b) High-Fidelity ($l/L = 0.1$)

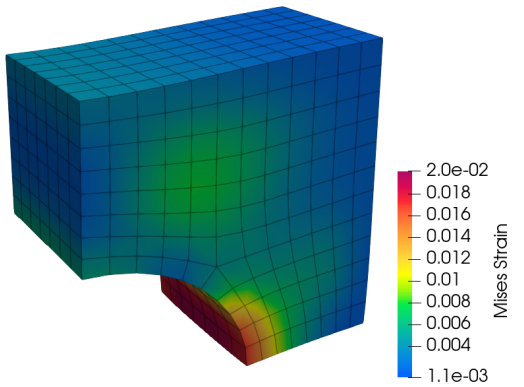


(c) GRU

Figure 7.36. The comparison of the displacement distribution in z-direction of high-fidelity and multiscale simulation in the 75th time step (sinusoidal surface traction force).



(a) von Mises stress



(b) von Mises strain

Figure 7.37. The von Mises stress and strain distribution of multiscale simulation in the 75th time step (sinusoidal surface traction force).

Figure 7.38 illustrates the displacement in the y-direction of node “a” versus the applied traction on the plate, as well as the displacement in the z-direction of node “b”. When l/L equals 0.25, the relative error of U_2 is 0.092, and when l/L equals 0.1, the relative error is 0.076. Based on the stress concentration location, integration point 7 of element “C” was selected to plot the true and predicted values of stress components. As before, the true stress values are derived from RVE simulation and homogenization. This result is illustrated in Figure 7.39. The relative error of our GRU model at this integration point is 0.045, which is even better than the linear case. The positions of these nodes and elements are shown in Figure 7.26. This test demonstrates the performance of our model is similar to the linear case.

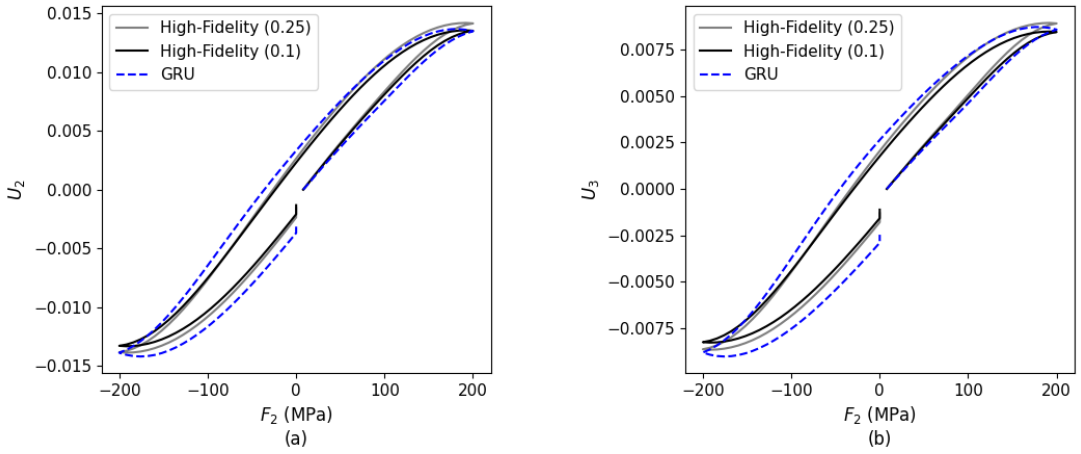


Figure 7.38. The displacement in y-direction of node “a” and the displacement in z-direction of node “b” versus time step (sinusoidal surface traction force).

7.5 Testing in a 2D Problem

In this research, our GRU model was trained using 3D data, but testing shows that it also has good accuracy when dealing with 2D composite material problems. In this section, we use a two-dimensional problem to verify the effectiveness of our surrogate model in this type of problem.

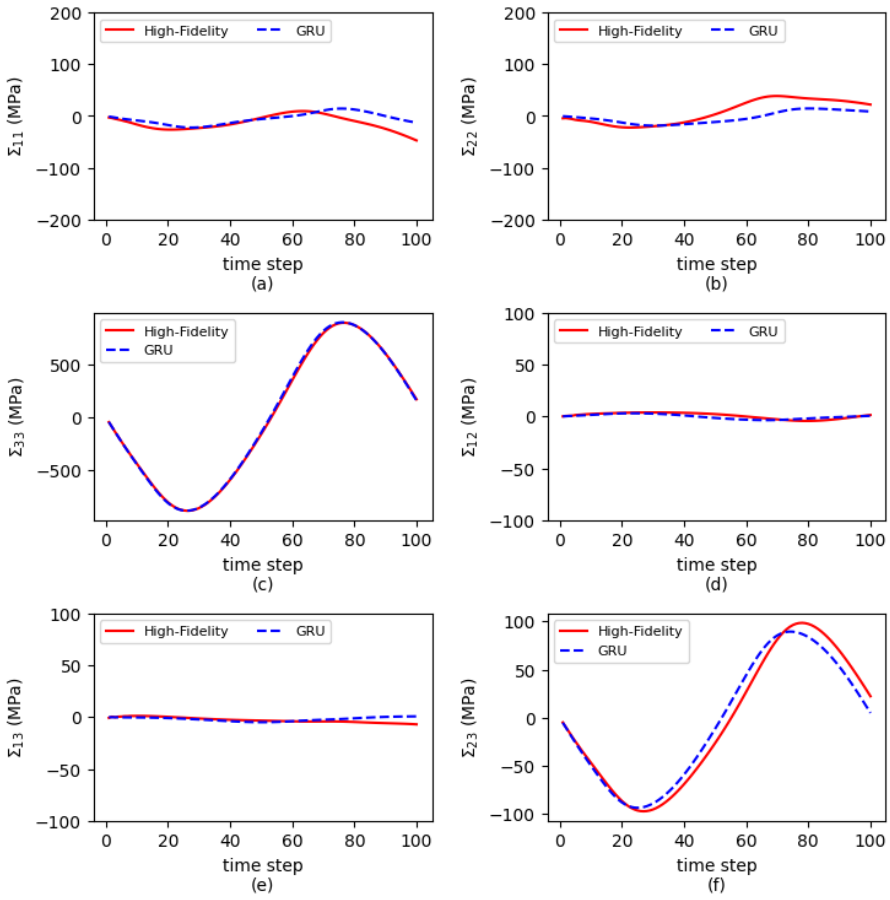


Figure 7.39. The predicted and true stress at integration point 7 of element "C" versus time steps (sinusoidal surface traction force).

The geometry of our two-dimensional problem is the same as the cross-section of the cantilever beam perpendicular to the fibers mentioned earlier. The boundary conditions of this problem are illustrated in Figure 7.40. It should be pointed out that in actual simulation, we need additional constraints to ensure that the displacements of the nodes on the upper and lower surfaces of the model are equal in the y-direction, respectively. These constraints are omitted in the figure to avoid confusion. In the right surface of the model, there is a cyclic uniform displacement along the x direction with maximum and minimum magnitude equal to ± 0.006, the number of time steps is equal to 100. The pattern of the load is the same as the protocol of E_{33} in Figure 7.3b. In this case, we use plane strain elements with the hybrid formulation for the high-fidelity case and the multiscale case. In the high-fidelity simulation, we need fine mesh, and the mesh size is set to 0.025, while in the multiscale simulation, we use coarse mesh, and the mesh size is set to 0.25.

It should be pointed out that although we do not need to modify our trained GRU model, we need to make simple modifications to our UMAT code to adapt to two-dimensional problems. Specifically, in two-dimensional problems, some directions have zero stress and strain, but the input and output of our GRU model still need to include all components of stress and strain. So we need to add zeros to certain components upon input and only transmit theoretically non-zero components to ABAQUS upon output.

In Figure 7.41a, the plot of the true value and prediction of reaction force in the right surface versus U_{11} when l/L is equal to 0,25 is illustrated. The mesh size in this case is 0.025. The relative error in the 75th time step is equal to 6.71%. Through this plot, we can figure out that the performance of our model in the 2D problem is not bad. The increase in the error of the prediction is not significant in the 2D case. Meanwhile, we also set the values of l/L to 0.1, 0.05, and 0.02, and conducted additional high-fidelity simulations. The mesh sizes for these three cases are 0.01, 0.005, and 0.0025, respectively. The results of those simulations are illustrated in Figure 7.41b, Figure 7.41c, and Figure 7.41d. Through these figures, we clearly find that the value of l/L has a low effect on the results of high-fidelity simulations in this problem.

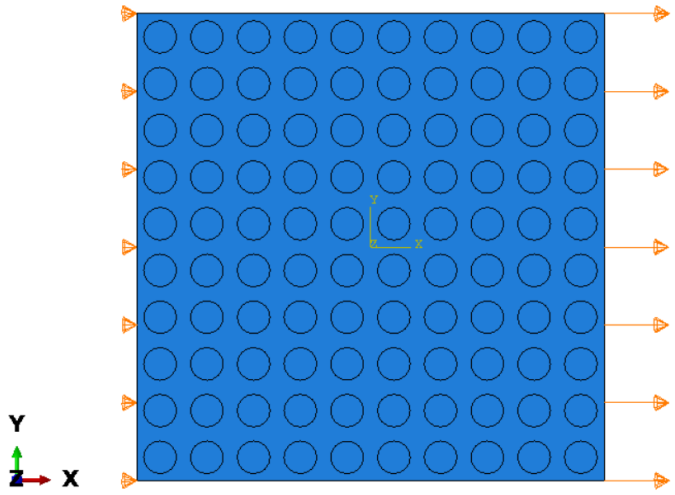
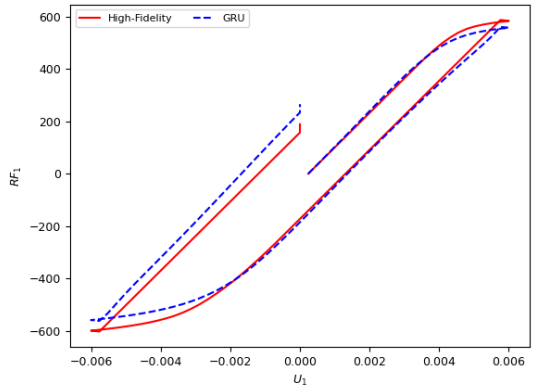
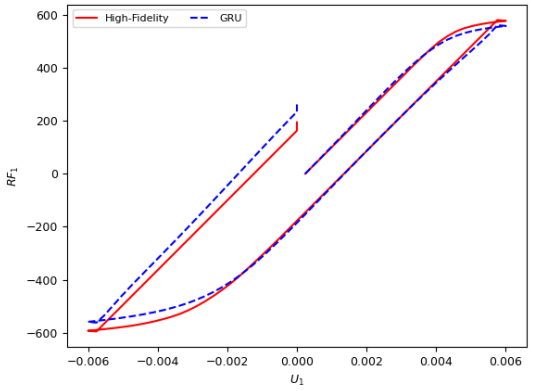


Figure 7.40. The boundary condition of 2D problem ($l/L = 0.1$).

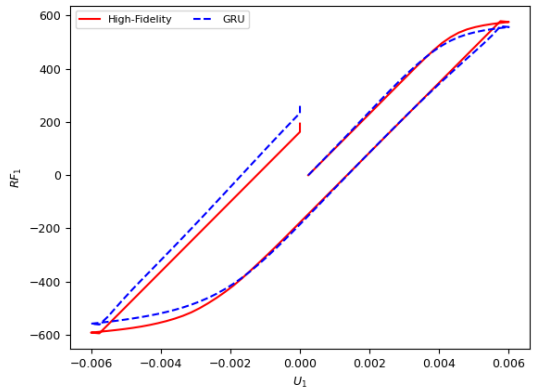
Chapter 7, in full, has been submitted for publication of the material, Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this paper.



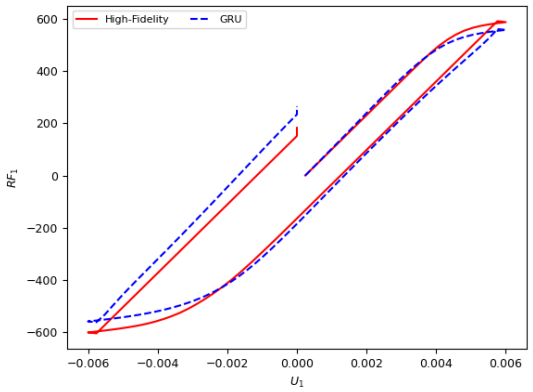
(a) $l/L = 0.25$



(b) $l/L = 0.1$



(c) $l/L = 0.05$



(d) $l/L = 0.02$

Figure 7.41. The predicted and true reaction force on the right surface versus U_1 in the 2D problem.

Chapter 8

Expanding Research

In the previous sections, we introduced the application of a GRU-based surrogate model in nonlinear multiscale simulations. Despite the good performance of the GRU model, it has two significant drawbacks. The first drawback is that the performance of the GRU degrades when the sequence is very long. The second drawback is that when certain strain components are equal to or near zero, the GRU's predicted values exhibit irregular fluctuations. In this chapter, we design a transformer-based surrogate model to learn the constitutive relationships of nonlinear materials and attempt to address the aforementioned shortcomings. The transformer model, proposed in the paper [79] by Vaswani et al., can directly capture dependencies between any two positions, and it has already achieved great success in natural language processing (NLP) and computer vision (CV).

8.1 Transformer Model

8.1.1 Attention Mechanism

Self-Attention Mechanism

Self-Attention Mechanism is the core idea of the transformer model, which is used to learn the dependency relationships of long sequences. The self-attention mechanism can directly model the dependency relationship between any two positions in the input sequence, regardless of the distance between them. Meanwhile, the self-attention mechanism also has many advantages, such as allowing for parallel computing. It has already shown outstanding performance in the fields of NLP and CV. We provide a detailed explanation of it below. Assume the input is a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the embedding (or input) dimension. Each input vector $\mathbf{x}_i$ represents the i-th position in the sequence. The input matrix $\mathbf{X}$ undergoes three different linear transformations to obtain the Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$) matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{Q}, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{K}, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{V} \tag{8.1}$$

Here, $\mathbf{W^Q}, \mathbf{W^K}, \mathbf{W^V} \in \mathbb{R}^{d \times d_k}$ are trainable parameter matrices, and d_k is usually set to $d\frac{1}{h}$ (where h is the number of heads in the multi-head attention mechanism). We can expand the equations in 8.1 into matrix form to obtain Equation 8.2, 8.3, and 8.4.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11}^{Q} & w_{12}^{Q} & \cdots & w_{1d_k}^{Q} \\ w_{21}^{Q} & w_{22}^{Q} & \cdots & w_{2d_k}^{Q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1}^{Q} & w_{d2}^{Q} & \cdots & w_{dd_k}^{Q} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_n \end{bmatrix} \tag{8.2}$$

$$\mathbf{K} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11}^{K} & w_{12}^{K} & \cdots & w_{1d_k}^{K} \\ w_{21}^{K} & w_{22}^{K} & \cdots & w_{2d_k}^{K} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1}^{K} & w_{d2}^{K} & \cdots & w_{dd_k}^{K} \end{bmatrix} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \vdots \\ \mathbf{k}_n \end{bmatrix} \tag{8.3}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11}^V & w_{12}^V & \cdots & w_{1d_k}^V \\ w_{21}^V & w_{22}^V & \cdots & w_{2d_k}^V \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1}^V & w_{d2}^V & \cdots & w_{dd_k}^V \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} \tag{8.4}$$

After obtaining the three matrices **Q**, **K**, and **V**, we can start computing the attention score. First, we compute the dot product of the Query matrix **Q** and the Key matrix **K**. This step aims to compute the similarity between each query and key, which is shown in Equation 8.5.

$$\mathbf{QK}^T = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_n \end{bmatrix} \begin{bmatrix} \mathbf{k}_1^T & \mathbf{k}_2^T & \cdots & \mathbf{k}_n^T \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 \cdot \mathbf{k}_1 & \mathbf{q}_1 \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_1 \cdot \mathbf{k}_n \\ \mathbf{q}_2 \cdot \mathbf{k}_1 & \mathbf{q}_2 \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_2 \cdot \mathbf{k}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{q}_n \cdot \mathbf{k}_1 & \mathbf{q}_n \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_n \cdot \mathbf{k}_n \end{bmatrix} \tag{8.5}$$

Each element $\mathbf{q}_i \cdot \mathbf{k}_j$ in this matrix represents the similarity between the i-th query vector and the j-th key vector. To avoid the problem of large dot product values leading to gradient vanishing or explosion, we divide the dot product result by a scaling factor $\sqrt{d_k}$, which is shown in Equation 8.6

$$\frac{\mathbf{QK}^T}{\sqrt{d_k}} \tag{8.6}$$

Here, $\sqrt{d_k}$ is the square root of the dimension of the key vectors. Scaling helps keep the dot product results within a reasonable range, making the subsequent softmax operation more stable. Next, we apply the softmax function to the scaled result to obtain the attention score matrix in Equation 8.7. The softmax function converts the similarity values into a probability distribution, ensuring that the scores for each query vector sum to 1:

$$\text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right) \tag{8.7}$$

The softmax function is defined in Equation 8.8:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{8.8}$$

where z_i is the attention score for the i-th element.

By applying the softmax function, we obtain the attention weights for each query vector over all key vectors. These weights indicate the degree to which each query vector should focus on each key vector when computing the final output.

Finally, we use the attention score matrix to perform a weighted sum of the value matrix $\mathbf{V}$, yielding the final output, which is shown in Equation 8.9. In Equation 8.10 we show the matrix form of it.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{8.9}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} \alpha_{1j}\mathbf{v}_j \\ \sum_{j=1}^{n} \alpha_{2j}\mathbf{v}_j \\ \vdots \\ \sum_{j=1}^{n} \alpha_{nj}\mathbf{v}_j \end{bmatrix} \tag{8.10}$$

where α_{ij} denotes the softmax-normalized attention scores.

Multi-Head Attention Mechanism

In transformer models, we often introduce multi-head attention mechanism, which are built based on self-attention mechanism. It enables the model to focus on different parts of the input sequence by using multiple attention heads in parallel, thereby capturing richer features and relationships. The single-head attention mechanism only uses one attention function, while the multi-head attention mechanism uses multiple attention functions (i.e. multiple heads), each

with an independent query, key, and value matrix. Through this approach, different heads can focus on input data from different perspectives, thereby improving the model's representation ability.

We extend the self attention mechanism to multiple heads, each with its own set of linear transformations, which is shown in Equation 8.11. The matrix form of it is shown in Equation 8.12

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1,\text{head}_2,\ldots,\text{head}_h)\mathbf{W}^O \tag{8.11}$$

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \begin{bmatrix} \text{head}_1 \\ \text{head}_2 \\ \vdots \\ \text{head}_h \end{bmatrix} \mathbf{W}^O \tag{8.12}$$

where,

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i,\mathbf{K}_i,\mathbf{V}_i) = \text{Attention}(\mathbf{X}\mathbf{W}_i^Q,\mathbf{X}\mathbf{W}_i^K,\mathbf{X}\mathbf{W}_i^V) \tag{8.13}$$

In Equation 8.13, $\mathbf{W}_i^Q$ represents the transformation matrix for the query vector of the i-th head. It is used to convert the input sequence into the query vector $\mathbf{Q}_i$. $\mathbf{W}_i^K$ represents the transformation matrix for the key vector of the i-th head. It is used to convert the input sequence into the key vector $\mathbf{K}_i$. $\mathbf{W}_i^V$ represents the transformation matrix for the value vector of the i-th head. It is used to convert the input sequence into the value vector $\mathbf{V}_i$. $\mathbf{W}^O$ is the linear transformation matrix used after the multi-head attention mechanism. It concatenates the outputs of all heads and then performs this linear transformation to obtain the final output.

8.1.2 Transformer Models

Original Transformer Model

According to the original paper, the transformer model is composed of 6 encoder layers and 6 decoder layers stacked together, the structure of each layer is the same. In each layer of the encoder, there is a multi-head self-attention mechanism and a feedforward neural network, both of which use layer normalization. Layer normalization is a regularization technique aimed at solving the problems of vanishing and exploding gradients when training a deep model. Meanwhile, both multi-head self-attention and feedforward networks have a residual connection. Residual connection is a technique used to alleviate the difficulty of training a deep model, which adds direct connections between the input and output of each layer, making it easier for gradients to propagate to deeper layers of the model. This technology can effectively solve the problem of gradient vanishing and promote the transmission of information in the model.

In each layer of the decoder, the model sequentially connects a masked multi-head self-attention mechanism, a multi-head self-attention mechanism, and a feedforward neural network. After each layer, layer normalization is used, and residual connections are used for each layer. It should be pointed out that, the output of the encoder is used as the Key and Value of the multi-head attention layer in the decoder. The Query of it is from the masked multi-head attention layer of the decoder itself. The complete transformer architecture of the original paper [79] is illustrated in Figure 8.1.

In the transformer model, it is also necessary to include input embedding, output embedding, and positional encoding structures. Input embedding is the process of converting vocabulary in an input sequence into high-dimensional vector representations. These vector representations can facilitate the model in capturing semantic information of vocabulary. Output embedding is similar to input embedding, which is the process of converting vocabulary in the

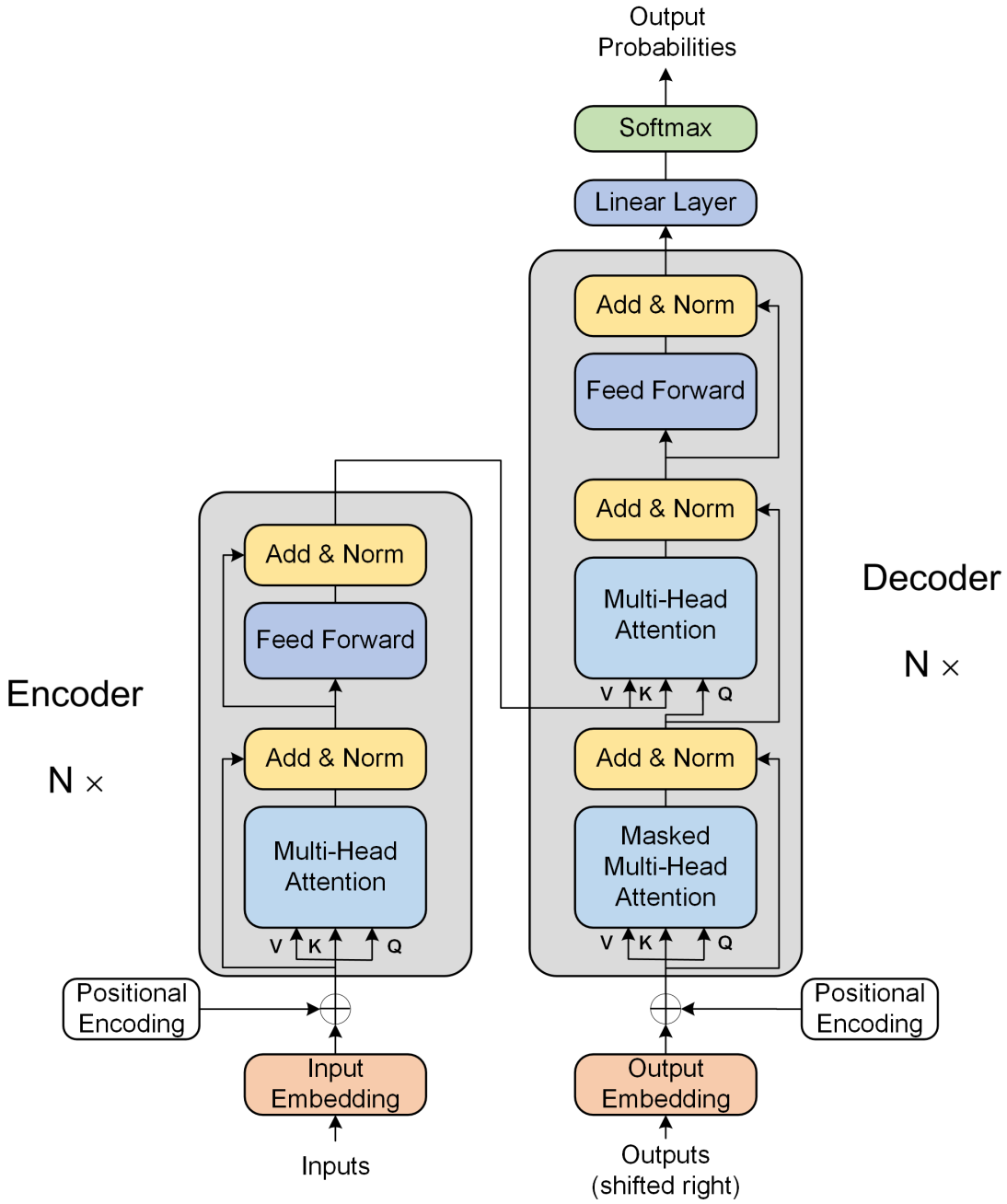


Figure 8.1. Architecture of the transformer model in the original paper [79].

target sequence into high-dimensional vector representations. Used in the decoder. Due to the lack of built-in sequential information in transformers, they also need to explicitly incorporate positional information into vocabulary embeddings through positional encoding. The purpose of positional encoding is to add unique positional information to the embedding of vocabulary at each position, enabling the model to recognize the relative and absolute positions of vocabulary in the sequence.

Now, we define the positional encoding vector PE. For each position *pos* in the input sequence and each dimension index *i* in the embedding dimension, the computation of positional encoding is shown in the Equation 8.14 and 8.15.

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{8.14}$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{8.15}$$

Transformer Surrogate Model

The original transformer model was originally used in the NLP field, and now we redesign this architecture to be suitable for handling regression problems and use it as a constitutive surrogate model.

Firstly, we modify the multi-head attention layer in the encoder to a masked multi-head attention layer. The reason is that in NLP, the input of the encoder is usually the user's input sentence, so the model needs to be able to read the entire sentence, otherwise, it leads to semantic loss. During the training phase, the model should not see its own future output (the entire correct output sequence is saved in the training set), so the decoder needs a mask. However, when we use the transformer model to learn the stress response of materials, the current stress sequence is only related to past steps and completely independent of future steps, so the encoder part

of the model should not see future information. Specifically, in this research, we used strain sequences as inputs to the encoder, so we need a mask to prevent the model from peeking at future information that is completely unrelated to the current step.

In addition, we remove the input embedding layer and output embedding layer of the model, as these layers are used to project words onto a continuous vector space. We use linear layers instead of them. As we use the transformer for regression problems, we also remove the linear layer and Softmax layer after the encoder and replace them with a feedforward neural network with an activation function of LeakyReLU.

The architecture of transformer surrogate model 1 is illustrated in Figure 8.2. The input of the encoder of the model is the strain sequence and the input of the decoder of the model is the strain increment sequence when the model deals with the J2 plasticity constitutive model. The output of the model is stress sequence.

8.2 Training Strategies

8.2.1 Random Extract Training Algorithm

In order to enable the transformer model to process sequences of different lengths, researchers often use datasets containing sequences of different lengths to train the model. In the field of NLP, researchers often use padding methods to solve the problem of unequal sequence lengths during the training process. For sequences within a batch, researchers typically pad all sequences to the same length by appending special padding tokens (commonly [PAD]) to the end of shorter sequences. This ensures that all sequences within a batch have uniform length, facilitating batch processing. To prevent the model from being misled by the padding tokens during training, masking techniques are employed. A masking matrix marks the padding positions as invalid, ensuring that these positions are ignored during the attention mechanism

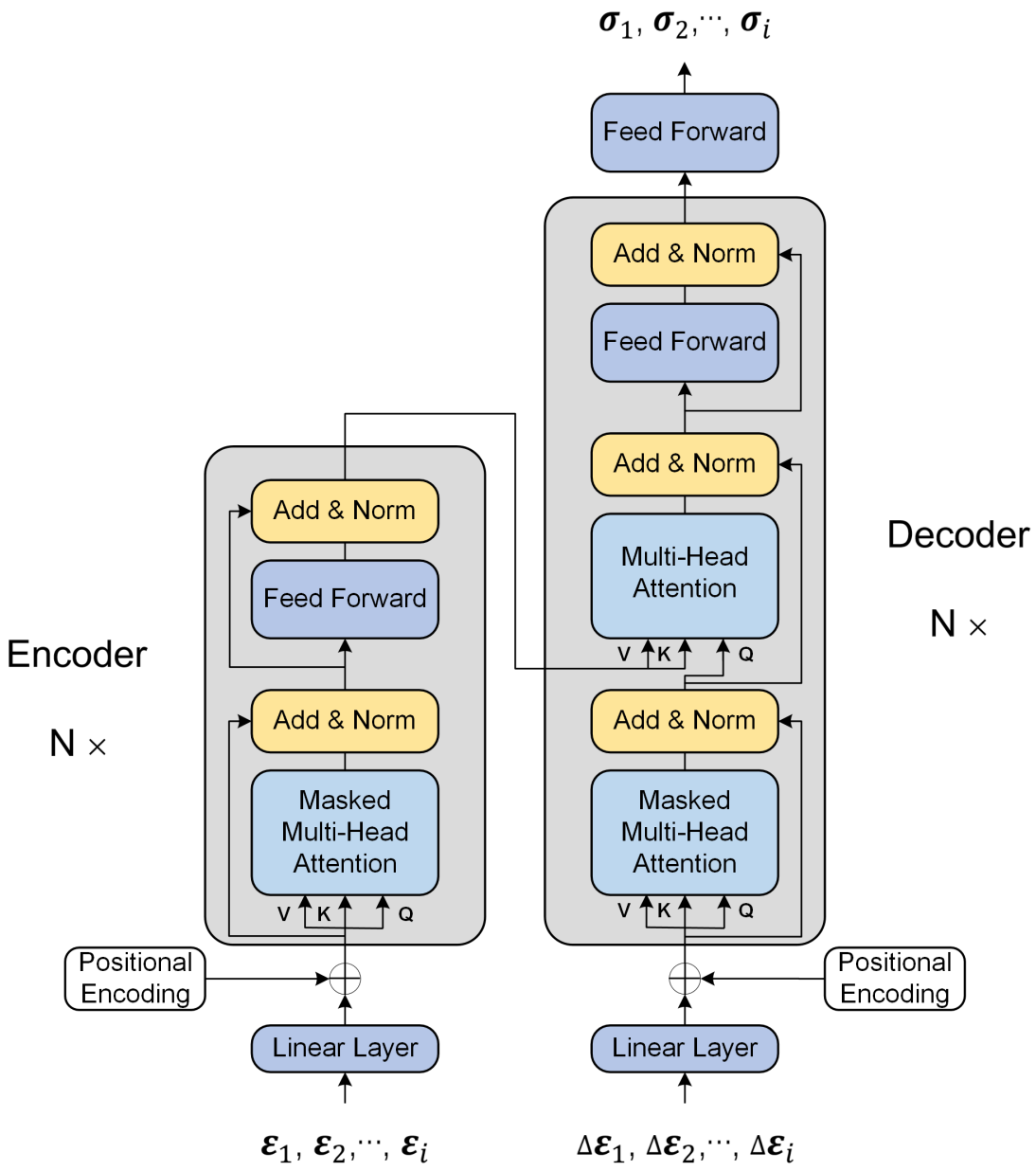


Figure 8.2. Architecture of the transformer surrogate model 1.

and loss calculation. When computing attention weights, the masking matrix assigns a negative infinity weight to the padding positions, effectively nullifying their contribution to the results. Additionally, researchers sometimes place sequences of similar lengths within the same batch and then padding or dynamically adjust the padding length based on the longest sequence in the current batch to make the training process more efficient.

Although the above methods can solve the problem of unequal sequence lengths, the process is relatively cumbersome. In this research, we propose a new training algorithm that can more succinctly solve this problem. Firstly, our data generator generates a set of datasets, including stress, strain, and strain increment datasets. The sequences in these datasets are of equal length, and their length should be equal to or slightly greater than the maximum number of time steps required for our simulation.

When we train the model, we generate a random integer index before the start of each batch in each epoch, which falls between the sequence length in the dataset and a preset minimum length (including endpoints), which should usually be equal to or slightly less than the minimum time steps required for our simulation. Then, when we extract the current batch from the dataset, we only extract the 0th to (n-1)th steps of all sequences in that batch, where n is equal to the value of the random index. For each batch during training, we regenerate the random integer index. In addition, when our training reaches the penultimate epoch, we set the value of the random index to a constant value equal to the length of the sequence in the dataset. This is to prevent the problem of excessive errors in the last few steps when we use the model to process sequences with the maximum length. In the last epoch, the random index is restored and randomly generated. The details of this method are shown in the Algorithm 4. In this algorithm, X_train is a strain sequences dataset, Y_train is a stress dataset, and $X2_train$ is a strain increment dataset.

Training the transformer with Algorithm 4 not only simplifies the training process, but also simplifies the structure of the data generator, as we do not need to generate a set of unequal length sequences. The reason why this algorithm is effective is that the current step of the stress sequence is only related to past information, and in NLP, if you only extract the first half of a

Algorithm 4: Random Extract Training Algorithm

Input: Training datasets, Training parameters, Defined model
Result: Transformer model training completed

Initialize transformer model;
Define the tensors we need;
$l_m \leftarrow$ minimum length;
$l \leftarrow$ length of sequence of training dataset;
for *epoch* $\leftarrow$ *0* ***to*** *number_of_epochs* **do**
  for $i \leftarrow$ *0* ***to*** *(length_of_training_dataset / batch_size)* **do**
    $random_index \leftarrow$ random.randint$(1, l_m)$;
    if $epoch == number_of_epochs - 2$ **then**
      $random_index \leftarrow l$;
    end
    $X_train_i \leftarrow X_train[i * batch_size : (i+1) * batch_size, 0 : random_index, :]$;
    $Y_train_i \leftarrow Y_train[i * batch_size : (i+1) * batch_size, 0 : random_index, :]$;
    $X2_train_i \leftarrow X2_train[i * batch_size : (i+1) * batch_size, 0 : random_index, :]$;
    $prediction \leftarrow$ Model$(X_train_i, X2_train_i)$;
    $loss \leftarrow$ loss_func$(prediction, Y_train_i)$;
    Save training loss;
    Compute and save testing loss;
    Perform backpropagation and update model;
  end
  Adjust the learning rate;
end

sentence, it may lead to semantic loss.

8.2.2 Learning Rate Adjustment Strategy

Dynamically adjusting the learning rate is crucial when training transformer models. This method can accelerate the convergence speed of the model to save computational resources and be more stable in the later stages of training to reduce the risk of overfitting. The warmup adjustment method is a commonly used learning rate adjustment strategy when we train the transformer model. It adopts a method of gradually increasing the learning rate in the initial stage of training, rather than directly using a larger learning rate. This method can help the model train stably in the initial stage and prevent training instability caused by excessive gradients. We design our learning rate adjustment strategy by combining the Warmup method and the learning rate reduction method, which is shown in Algorithm 5.

In the warmup stage of our learning rate adjustment algorithm, we set a small initial learning rate and enlarge it to 100 times its original size within 50 epochs. Then, after n epochs, we multiply the learning rate by the preset reduction coefficient γ_2, where n equals step size 2 until the learning rate is lower than the minimum learning rate 1. Afterwards, whenever the training loss does not decrease for n consecutive epochs, where n equals step size 3 we multiply the learning rate by the preset reduction coefficient γ_3. When our learning rate is less than the minimum learning rate 2, we no longer adjust the learning rate.

8.3 Performance of Transformer Surrogate Model

In this section, we evaluate the performance of transformer surrogate models. We generated a dataset using Algorithm 2, with a dataset size of 10000 and 90% of it for training. Each sequence has a length of 200, and the minimum number of plastic points is set to 55. During data generation, we used the material properties of 7075-T6 aluminum alloy. We tested

Algorithm 5: Learning Rate Adjustment Algorithm

Input: $initial_lr, min_lr_1, min_lr_2, \gamma_2, \gamma_3, step_size_2, step_size_3$
Result: Learning Rate Adjustment

Initialize transformer model;
$\gamma_1 \leftarrow 1.0965$;
$step_size_1 \leftarrow 1$;
$warmup_range \leftarrow 50$;
$lr \leftarrow initial_lr$;
for *epoch* $\leftarrow$ *0* ***to*** *number_of_epochs* **do**
    The training process of current epoch;
    if *epoch* $<$ *warmup_range* **then**
        $lr \leftarrow \gamma_1 * lr$;
    else if *epoch* $>$ *warmup_range* ***and*** $lr >$ *min_lr_1* **then**
        if *epoch - warmup_range % step_size_2 == 0* **then**
            $lr \leftarrow \gamma_2 * lr$;
        end
    else
        if *The training loss does not decrease after step_size_3 consecutive epochs* ***and*** $lr >$ *min_lr_2* **then**
            $lr \leftarrow \gamma_3 * lr$;
        end
    end
end

our models using the strain sequence shown in Figure 7.3d but extended the time steps to 200. Additionally, in some tests, we set the shear strain in the sequence to a non-zero value.

Through tuning, we determine the appropriate model and training parameters for the transformer surrogate model. The transformer model hyperparameters are as follows: the embedding dimension is set to 120, with 6 heads in the multi-head attention models (where the embedding dimension must be divisible by the number of heads). The dimension of the feedforward network model in the transformer encoder and decoder is set to 800. The model includes 6 encoder layers and 6 decoder layers. Both the input size and output size for the entire model are set to 6. For training parameters, the initial learning rate is set between 0.000002 and 0.00001. The number of epochs ranges from 1000 to 2000, and the batch size is set between 50 and 100. The minimum training loss of the transformer model can achieve 5e-8. The hyperparameters of the GRU model are the same as before.

8.3.1 Performance of Random Extract Training Algorithm

In this subsection, we compare the impact of training algorithms on the performance of the transformer model. We train the models using a dataset described previously. One model is trained using the traditional algorithm, where the sequence length remains constant at 200 during training, and the other model is trained using the random extract training (RET) algorithm. After training, we apply both models to sequences of length 100. The results are presented in Figure 8.3. The sequences used to test the models are derived from Figure 7.3d. Meanwhile, we set the value of E_{12} and E_{23} in the sequence equal to E_{11} and E_{22}, respectively.

In Figure 8.3, the red line represents the true stress values, the blue dashed line indicates the predictions of the transformer model trained with the traditional training algorithm, and the green dashed line represents the predictions of the transformer model trained with the RET algorithm. The figure clearly demonstrates that the RET algorithm significantly outperforms the traditional training algorithm. From the perspective of relative error, the model trained with the

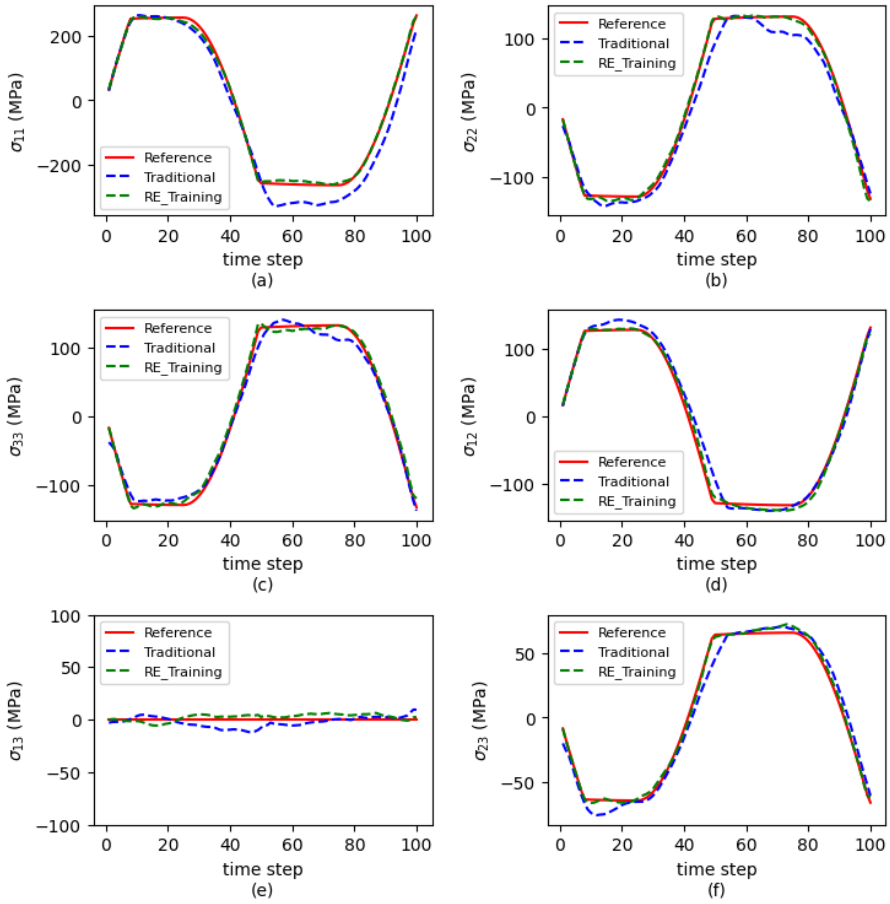


Figure 8.3. Comparison of the prediction of transformers trained by traditional and random extract training algorithm.

traditional algorithm exhibits a relative error of 0.168 when processing sequences of length 100, whereas the model trained with the RET algorithm shows a substantially lower error of only 0.042, indicating a marked improvement over the traditional algorithm. Table 8.1 demonstrates that the performance of the model trained with the RET algorithm remains relatively stable when processing sequences of different lengths. In contrast, the performance of the model trained with the traditional algorithm significantly deteriorates when processing sequences of lengths different from those in the training dataset.

Table 8.1. Comparison of the relative errors of the transformers trained by traditional and RET algorithm.

	Traditional	RET Algorithm
Sequences of length 200	0.063	0.049
Sequences of length 100	0.169	0.044

8.3.2 Comparison Between Transformer and GRU

First, we extend the same strain sequence from Figure 7.3d to 200 steps to test the transformer and GRU surrogate models. In Figure 8.4, we plot the predictions of the transformer and GRU models from step 101 to step 200 with the time steps. In Figure 8.5, we plot the predictions of the transformer and GRU models from step 101 to step 200 for the Σ_{11} and Σ_{33} versus the corresponding strain components, which is used to compare the predictions of those two models more clearly. The red line in the figure represents the reference value, which is the true value, the blue dashed line represents the prediction of GRU, and the green dashed line represents the prediction of the transformer.

From Figure 8.4 and Figure 8.5, we can clearly observe that, in terms of normal stress, the predictions of the transformer surrogate model are more accurate and stable compared to the GRU model. We can also observe that, due to the shear strain being zero, the predictions of the surrogate models fluctuate randomly around zero.

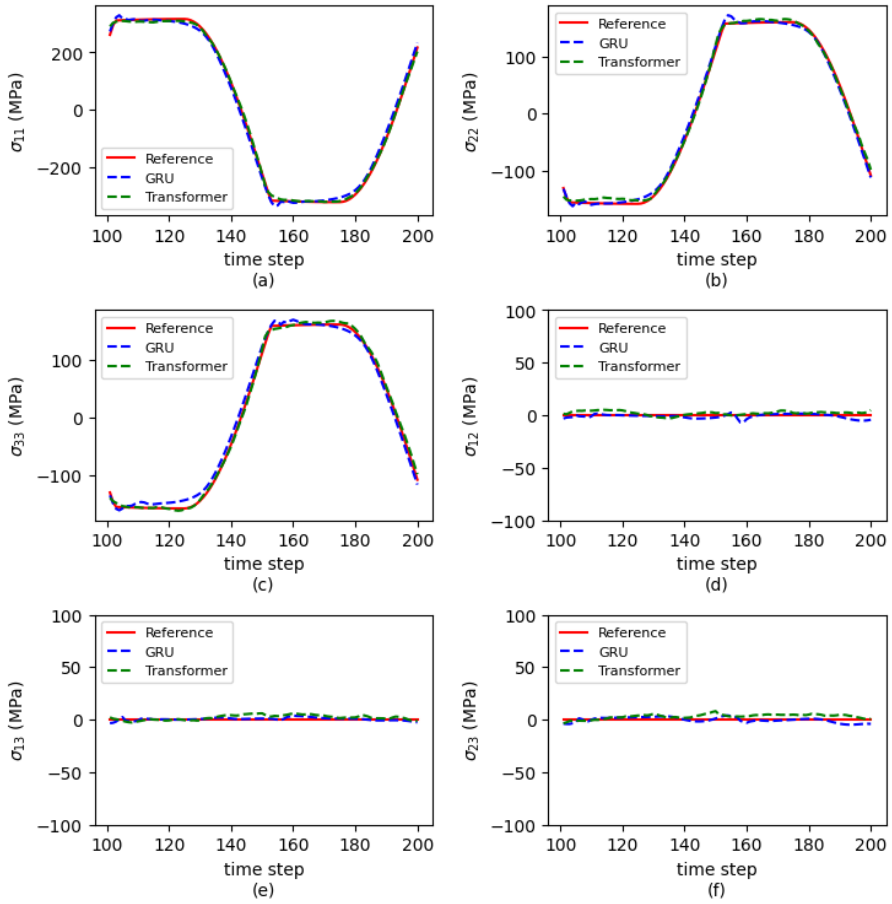


Figure 8.4. Comparison of the prediction of transformer and GRU versus time step under triaxial sinusoidal cyclic strain state with the shear strain of 0 (steps 101 to 200).

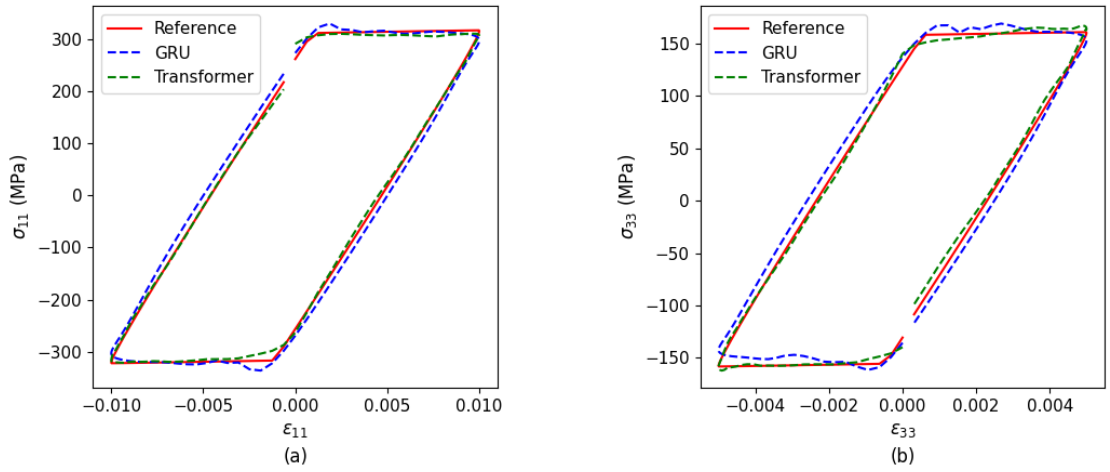


Figure 8.5. Comparison of the prediction of transformer and GRU versus strain under triaxial sinusoidal cyclic strain state with the shear strain of 0 (steps 101 to 200).

In Table 8.2, we present the relative errors of the transformer and GRU models over the entire prediction process and in the latter part of the prediction process. It is clear that the transformer model outperforms the GRU model in terms of accuracy, both over the entire prediction process and in the latter part of the predictions. Additionally, we observe that the GRU model's performance deteriorates in the latter part of the predictions compared to its overall performance, whereas the transformer model does not exhibit this issue.

Table 8.2. Comparison of the relative errors of the transformer and GRU models over the entire prediction process and in the latter part of the predictions (triaxial sinusoidal cyclic strain state with the shear strain of 0).

	GRU Model	Transformer Model
Entire prediction process	0.052	0.036
Step 101 to step 200	0.054	0.035

Secondly, we set the value of E_{12} and E_{13} in the sequence form Figure 7.3d equal to E_{11} and E_{22} of it, respectively, then we extend the strain sequence to 200 steps to test the transformer and GRU surrogate models. In Figure 8.6, we plot the predictions of the transformer and GRU models from step 101 to step 200 versus the time steps. In Figure 8.7, we plot the predictions

of the transformer and GRU models from step 101 to step 200 for the Σ_{12} and Σ_{13} versus the corresponding strain components, which is used to compare the predictions of those two models more clearly. The red line in the figure represents the reference value, which is the true value, the blue dashed line represents the prediction of GRU, and the green dashed line represents the prediction of the transformer.

From Figure 8.6 and Figure 8.7, we can clearly observe that the predictions of the transformer surrogate model are more accurate for both normal stress and shear stress compared to the GRU model. Additionally, we can note that the predictions of the transformer model are more stable, exhibiting fewer fluctuations than the GRU model, which is another advantageous characteristic of the transformer model.

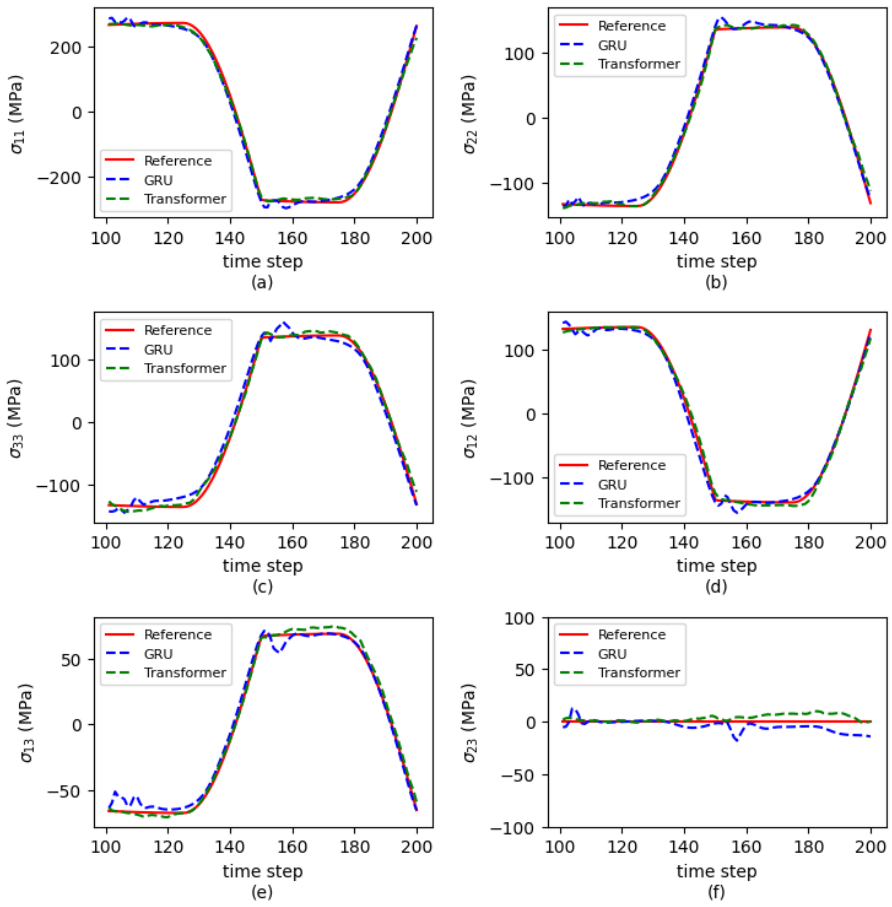


Figure 8.6. Comparison of the prediction of transformer and GRU versus time step under triaxial sinusoidal cyclic strain state with the non-zero shear strain (steps 101 to 200).

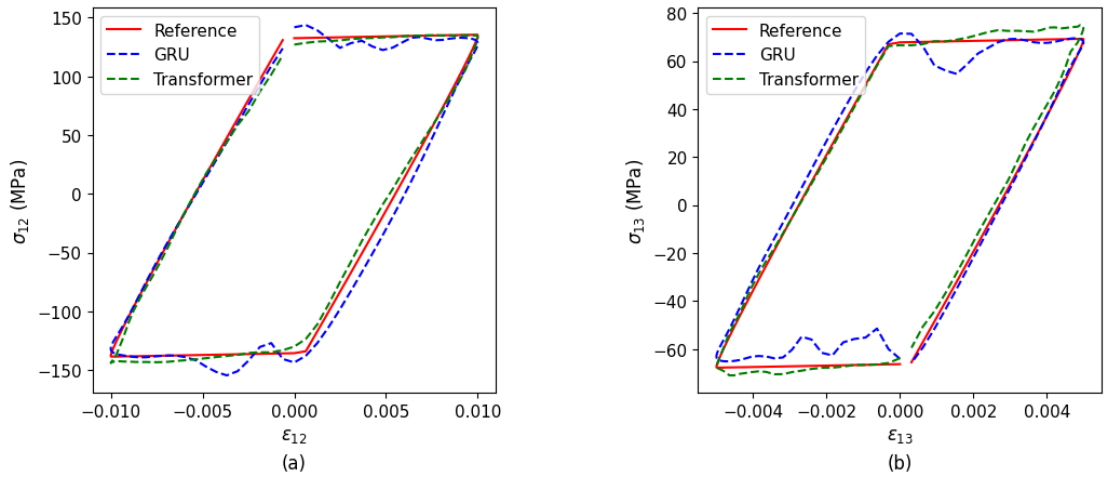


Figure 8.7. Comparison of the prediction of transformer and GRU versus strain under triaxial sinusoidal cyclic strain state with the non-zero shear strain (steps 101 to 200).

In Table 8.3, we present the relative errors of the transformer and GRU models over the entire prediction process and in the latter part of the prediction process. It is clear that the transformer model outperforms the GRU model in terms of accuracy, both over the entire prediction process and in the latter part of the predictions. Additionally, we observe that although the predictive performance of both the GRU and transformer models declines in the latter part of the predictions compared to their overall performance, the decline in the GRU model's performance is significantly more pronounced than that of the transformer model.

Table 8.3. Comparison of the relative errors of the transformer and GRU models over the entire prediction process and in the latter part of the predictions (triaxial sinusoidal cyclic strain state with the non-zero shear strain).

	GRU Model	Transformer Model
Entire prediction process	0.063	0.047
Step 101 to step 200	0.070	0.049

Chapter 8, in part, is currently being prepared for submission for publication of the material. Yijing Zhou; Shabnam J. Semnani. The thesis author was the primary investigator and author of this material.

Chapter 9

Concluding Remarks

Recent research has shown that machine learning based multiscale methods are one of the directions for solving multiscale simulations of heterogeneous materials. In this research, we implemented a 3D multiscale finite element framework of composite materials based on GRU. We proposed several new algorithms in this thesis to achieve more effective sampling, preprocessing, and reconstructing of our trained GRU model in commercial finite element software. Meanwhile, we verify our model in several cases, including multiple problem types and loading types, and the accuracy is overall satisfactory.

We propose a Fortran algorithm used to reconstruct our trained GRU model in ABAQUS UMAT as a surrogate model to implement multiscale simulations. This algorithm enables us to utilize commercial finite element software for machine learning based multiscale modeling, which provides a very convenient option for other researchers to implement their surrogate model in multiscale simulations. In addition, with simple modifications, our UMAT subroutine can apply the GRU model trained on the 3D dataset to 2D planar problems and achieve good accuracy, which verifies that our method has a wider range of applications.

We test the performance of our surrogate model in multiple cases, including 3D and 2D multiscale problems, microscopic problems, and integration point problems under monotonic, cyclic, and sinusoidal loading. According to the test results, our model's error is less than 5% in approximately half of the tests, and in almost all tests, it is less than 10%, indicating

that the accuracy of our model is satisfactory. Our research demonstrates that the multiscale computational framework using GRU as a surrogate model has broad application prospects in the simulation of long fiber composite materials. In addition, we proposed several algorithms to optimize multiple steps in this framework.

We propose a random increment based algorithm to help us generate better training and testing datasets through series selection that contain more information on the plastic range. Through this algorithm, we can confirm that there are enough time steps in our RVE simulations to make the matrix enter the plastic range, which can make our GRU model learn the plastic behavior of composite material better. In addition, we developed a preprocess code to generate many RVE models in ABAQUS CAE. In this code, we proposed a new algorithm to apply periodic boundary conditions to the RVE model more effectively through coordinate transformation and sorting.

We conduct expanding research to address the shortcomings of the GRU model. We proposed a transformer-based surrogate model that leverages the attention mechanism to overcome the accuracy degradation of the GRU model when handling long sequences. Additionally, we introduced a new training method to enhance the generalization capability of the transformer model.

Although our research has made some progress, there is still much work worth completing. We observed that there are still errors in learning the constitutive relationship of composite materials using our surrogate model. In the future, We can conduct more in-depth research on the transformer-based surrogate model or explore the use of other types of models to solve path dependency simulation problems and achieve higher accuracy. Meanwhile, as a black box model, GRU and transformer have poor interpretability. In the future, we can consider using machine learning models based on physical information to improve the accuracy of predictions. Finally, we can expand the application of machine learning to more complex constitutive models in the future, such as rate-dependent materials.

Bibliography

[1] Frédéric Feyel and Jean-Louis Chaboche. Fe2 multiscale approach for modelling the elastoviscoplastic behaviour of long fibre sic/ti composite materials. *Computer methods in applied mechanics and engineering*, 183(3-4):309–330, 2000.

[2] Chin-Teh Sun and Rajesh S Vaidya. Prediction of composite properties from a representative volume element. *Composites science and Technology*, 56(2):171–179, 1996.

[3] JoséMiranda Guedes and Noboru Kikuchi. Preprocessing and postprocessing for materials based on the homogenization method with adaptive finite element methods. *Computer methods in applied mechanics and engineering*, 83(2):143–198, 1990.

[4] Pascale Kanouté, DP Boso, Jean-Louis Chaboche, and BA1170 Schrefler. Multiscale methods for composites: a review. *Archives of Computational Methods in Engineering*, 16:31–75, 2009.

[5] Dana Bishara, Yuxi Xie, Wing Kam Liu, and Shaofan Li. A state-of-the-art review on machine learning-based multiscale modeling, simulation, homogenization and design of materials. *Archives of computational methods in engineering*, 30(1):191–222, 2023.

[6] Frederic E Bock, Roland C Aydin, Christian J Cyron, Norbert Huber, Surya R Kalidindi, and Benjamin Klusemann. A review of the application of machine learning and data mining approaches in continuum materials mechanics. *Frontiers in Materials*, 6:110, 2019.

[7] Xin Liu, Su Tian, Fei Tao, and Wenbin Yu. A review of artificial neural networks in the constitutive modeling of composite materials. *Composites Part B: Engineering*, 224:109152, 2021.

[8] Jacob Fish, Gregory J Wagner, and Sinan Keten. Mesoscopic and multiscale modelling in materials. *Nature materials*, 20(6):774–786, 2021.

[9] Wing Kam Liu, Shaofan Li, and Harold S Park. Eighty years of the finite element method: Birth, evolution, and future. *Archives of Computational Methods in Engineering*, 29(6):4431–4453, 2022.

[10] Christoph Settgast, Martin Abendroth, and Meinhard Kuna. Constitutive modeling of plastic deformation behavior of open-cell foam structures using neural networks. *Mechanics of Materials*, 131:1–10, 2019.

[11] Usman Ali, Waqas Muhammad, Abhijit Brahme, Oxana Skiba, and Kaan Inal. Application of artificial neural networks in micromechanics for polycrystalline metals. *International Journal of Plasticity*, 120:205–219, 2019.

[12] Sunyoung Im, Hyungjun Kim, Wonbae Kim, and Maenghyo Cho. Neural network constitutive model for crystal structures. *Computational Mechanics*, 67:185–206, 2021.

[13] Dong Phill Jang, Piemaan Fazily, and Jeong Whan Yoon. Machine learning-based constitutive model for j2-plasticity. *International Journal of Plasticity*, 138:102919, 2021.

[14] Tomonari Furukawa and Genki Yagawa. Implicit constitutive modelling for viscoplasticity using neural networks. *International Journal for Numerical Methods in Engineering*, 43(2):195–219, 1998.

[15] Lahouari Benabou. Implementation of a neural network into a user-material subroutine for finite element simulation of material viscoplasticity. *Journal of Engineering Materials and Technology*, 143(4):041001, 2021.

[16] Sungmoon Jung and Jamshid Ghaboussi. Neural network constitutive model for rate-dependent materials. *Computers & Structures*, 84(15-16):955–963, 2006.

[17] Sungmoon Jung and Jamshid Ghaboussi. Characterizing rate-dependent material behaviors in self-learning simulation. *Computer methods in applied mechanics and engineering*, 196(1-3):608–619, 2006.

[18] Xueyang Li, Christian C Roth, and Dirk Mohr. Machine-learning based temperature-and rate-dependent plasticity model: Application to analysis of fracture experiments on dp steel. *International Journal of Plasticity*, 118:320–344, 2019.

[19] M Mozaffar, R Bostanabad, W Chen, K Ehmann, Jian Cao, and MA Bessa. Deep learning predicts path-dependent plasticity. *Proceedings of the National Academy of Sciences*, 116(52):26414–26420, 2019.

[20] Yangzi He and Shabnam J Semnani. Machine learning based modeling of path-dependent materials for finite element analysis. *Computers and Geotechnics*, 156:105254, 2023.

[21] Colin Bonatti and Dirk Mohr. On the importance of self-consistency in recurrent neural network models representing elasto-plastic solids. *Journal of the Mechanics and Physics of Solids*, 158:104697, 2022.

[22] Ning Zhang, Shui-Long Shen, Annan Zhou, and Yin-Fu Jin. Application of lstm approach for modelling stress–strain behaviour of soil. *Applied Soft Computing*, 100:106959, 2021.

[23] Sunyoung Im, Jonggeon Lee, and Maenghyo Cho. Surrogate modeling of elasto-plastic problems via long short-term memory neural networks and proper orthogonal decomposition. *Computer Methods in Applied Mechanics and Engineering*, 385:114030, 2021.

[24] Julian N Heidenreich and Dirk Mohr. Recurrent neural network plasticity models: Unveiling their common core through multi-task learning. *Computer Methods in Applied Mechanics and Engineering*, 426:116991, 2024.

[25] Shaoheng Guan, Xue Zhang, Sascha Ranftl, and Tongming Qu. A neural network-based material cell for elastoplasticity and its performance in fe analyses of boundary value problems. *International Journal of Plasticity*, 171:103811, 2023.

[26] Thomas Tancogne-Dejean, Maysam B Gorji, Juner Zhu, and Dirk Mohr. Recurrent neural network modeling of the large deformation of lithium-ion battery cells. *International Journal of Plasticity*, 146:103072, 2021.

[27] Wulf G Dettmer, Eugenio J Muttio, Reem Alhayki, and Djordje Perić. A framework for neural network based constitutive modelling of inelastic materials. *Computer Methods in Applied Mechanics and Engineering*, 420:116672, 2024.

[28] Ling Wu, Van Dung Nguyen, Nanda Gopala Kilingar, and Ludovic Noels. A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths. *Computer Methods in Applied Mechanics and Engineering*, 369:113234, 2020.

[29] F Ghavamian and A Simone. Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network. *Computer Methods in Applied Mechanics and Engineering*, 357:112594, 2019.

[30] Hernan J Logarzo, German Capuano, and Julian J Rimoli. Smart constitutive laws: Inelastic homogenization through machine learning. *Computer methods in applied mechanics and engineering*, 373:113482, 2021.

[31] Kun Wang and WaiChing Sun. A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. *Computer Methods in Applied Mechanics and Engineering*, 334:337–380, 2018.

[32] Ehsan Motevali Haghighi and SeonHong Na. A single long short-term memory network for enhancing the prediction of path-dependent plasticity with material heterogeneity and anisotropy. *arXiv preprint arXiv:2204.01466*, 2022.

[33] Juan Manuel Calleja Vazquez, Van Dung Nguyen, Ling Wu, Syed Mohib Mustafa, and Ludovic Noels. Development of history-dependent surrogate models in the context of stochastic multi-scale simulations for elasto-plastic composites. In *5th International Conference on Uncertainty Quantification in Computational Science and Engineering (UNCECOMP2023)*, 2023.

[34] Ling Wu and Ludovic Noels. Self-consistency reinforced minimal gated recurrent unit for surrogate modeling of history-dependent non-linear problems: Application to history-dependent homogenized response of heterogeneous materials. *Computer Methods in Applied Mechanics and Engineering*, 424:116881, 2024.

[35] Hyejin Kim, Inho Jeong, Haeseong Cho, and Maenghyo Cho. Surrogate model based on data-driven model reduction for inelastic behavior of composite microstructure. *International Journal of Aeronautical and Space Sciences*, 24(3):732–752, 2023.

[36] Soumianarayanan Vijayaraghavan, Ling Wu, Ludovic Noels, SPA Bordas, Sundar Natarajan, and Lars AA Beex. A data-driven reduced-order surrogate model for entire elastoplastic simulations applied to representative volume elements. *Scientific Reports*, 13(1):12781, 2023.

[37] Maysam B Gorji, Mojtaba Mozaffar, Julian N Heidenreich, Jian Cao, and Dirk Mohr. On the potential of recurrent neural networks for modeling path dependent plasticity. *Journal of the Mechanics and Physics of Solids*, 143:103972, 2020.

[38] J Friemann, B Dashtbozorg, M Fagerström, and SM Mirkhalaf. A micromechanics-based recurrent neural networks model for path-dependent cyclic deformation of short fiber composites. *International Journal for Numerical Methods in Engineering*, 124(10):2292–2314, 2023.

[39] Guang Chen. Recurrent neural networks (rnns) learn the constitutive law of viscoelasticity. *Computational Mechanics*, 67(3):1009–1019, 2021.

[40] Saurabh Balkrishna Tandale, Franz Bamer, Bernd Markert, and Marcus Stoffel. Physics-based self-learning recurrent neural network enhanced time integration scheme for computing viscoplastic structural finite element response. *Computer Methods in Applied Mechanics and Engineering*, 401:115668, 2022.

[41] MA Maia, IBCM Rocha, Pierre Kerfriden, and FP van der Meer. Physically recurrent neural networks for path-dependent heterogeneous materials: Embedding constitutive models in a data-driven surrogate. *Computer Methods in Applied Mechanics and Engineering*, 407:115934, 2023.

[42] Shiguang Deng, Shirin Hosseinmardi, Libo Wang, Diran Apelian, and Ramin Bostanabad. Data-driven physics-constrained recurrent neural networks for multiscale damage modeling of metallic alloys with process-induced porosity. *Computational Mechanics*, pages 1–31, 2024.

[43] L Borkowski, C Sorini, and A Chattopadhyay. Recurrent neural network-based multiaxial plasticity model with regularization for physics-informed constraints. *Computers & Structures*, 258:106678, 2022.

[44] Tongming Qu, Shaocheng Di, YT Feng, Min Wang, and Tingting Zhao. Towards data-driven constitutive modelling for granular materials via micromechanics-informed deep learning. *International Journal of Plasticity*, 144:103046, 2021.

[45] MA Maia, Iuri BCM Rocha, D Kovačević, and FP van der Meer. Physically recurrent neural network for rate and path-dependent heterogeneous materials in a finite strain framework. *arXiv preprint arXiv:2404.17583*, 2024.

[46] Jacob Fish and Yang Yu. Data-physics driven reduced order homogenization. *International Journal for Numerical Methods in Engineering*, 124(7):1620–1645, 2023.

[47] Miguel A Bessa, Ramin Bostanabad, Zeliang Liu, Anqi Hu, Daniel W Apley, Catherine Brinson, Wei Chen, and Wing Kam Liu. A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. *Computer Methods in Applied Mechanics and Engineering*, 320:633–667, 2017.

[48] Sadik L Omairey, Peter D Dunning, and Srinivas Sriramula. Development of an abaqus plugin tool for periodic rve homogenisation. *Engineering with Computers*, 35:567–577, 2019.

[49] Julian N Heidenreich, Colin Bonatti, and Dirk Mohr. Transfer learning of recurrent neural network-based plasticity models. *International Journal for Numerical Methods in Engineering*, 125(1):e7357, 2024.

[50] Tongming Qu, Jidong Zhao, Shaoheng Guan, and YT Feng. Data-driven multiscale modelling of granular materials via knowledge transfer and sharing. *International Journal of Plasticity*, 171:103786, 2023.

[51] Saurabh Balkrishna Tandale and Marcus Stoffel. Recurrent and convolutional neural networks in structural dynamics: a modified attention steered encoder–decoder architecture versus lstm versus gru versus tcn topologies to predict the response of shock wave-loaded plates. *Computational Mechanics*, 72(4):765–786, 2023.

[52] Anindya Bhaduri, Ashwini Gupta, and Lori Graham-Brady. Stress field prediction in fiber-reinforced composite materials using a deep learning approach. *Composites Part B: Engineering*, 238:109879, 2022.

[53] Ashwini Gupta, Anindya Bhaduri, and Lori Graham-Brady. Accelerated multiscale mechanics modeling in a deep learning framework. *Mechanics of Materials*, page 104709, 2023.

[54] Indrashish Saha, Ashwini Gupta, and Lori Graham-Brady. Prediction of local elasto-plastic stress and strain fields in a two-phase composite microstructure using a deep convolutional neural network. *Computer Methods in Applied Mechanics and Engineering*, 421:116816, 2024.

[55] Mohammad S Khorrami, Jaber R Mianroodi, Nima H Siboni, Pawan Goyal, Bob Svendsen, Peter Benner, and Dierk Raabe. An artificial neural network for surrogate modeling of stress fields in viscoplastic polycrystalline materials. *npj Computational Materials*, 9(1):37, 2023.

[56] Charles Yang, Youngsoo Kim, Seunghwa Ryu, and Grace X Gu. Using convolutional neural networks to predict composite properties beyond the elastic limit. *MRS Communications*, 9(2):609–617, 2019.

[57] Hao Su, TianYuan Guan, and Yan Liu. A three-dimensional prediction method of stiffness properties of composites based on deep learning. *Computational Mechanics*, 71(3):583–597, 2023.

[58] Hao-Syuan Chang, Jou-Hua Huang, and Jia-Lin Tsai. Predicting mechanical properties of unidirectional composites using machine learning. *Multiscale Science and Engineering*, 4(4):202–210, 2022.

[59] Jaber Rezaei Mianroodi, Shahed Rezaei, Nima H Siboni, Bai-Xiang Xu, and Dierk Raabe. Lossless multi-scale constitutive elastic relations with artificial intelligence. *npj Computational Materials*, 8(1):67, 2022.

[60] Mohammed Abir Mahdi, Christopher Crick, and Wei Zhao. Lattice structure design using machine learning and homogenization approach. In *ASME Aerospace Structures, Structural Dynamics, and Materials Conference*, volume 87745, page V001T01A015. American Society of Mechanical Engineers, 2024.

[61] Charles Yang, Youngsoo Kim, Seunghwa Ryu, and Grace X Gu. Prediction of composite microstructure stress-strain curves using convolutional neural networks. *Materials & Design*, 189:108509, 2020.

[62] Chengping Rao and Yang Liu. Three-dimensional convolutional neural network (3d-cnn) for heterogeneous material homogenization. *Computational Materials Science*, 184:109850, 2020.

[63] Hao Peng, An Liu, Jingcheng Huang, Lingxin Cao, Jikai Liu, and Lin Lu. Ph-net: Parallelepiped microstructure homogenization via 3d convolutional neural networks. *Additive Manufacturing*, 60:103237, 2022.

[64] Rutwik Gulakala, Bernd Markert, and Marcus Stoffel. Graph neural network enhanced finite element modelling. *PAMM*, 22(1):e202200306, 2023.

[65] Chunhao Jiang and Nian-Zhong Chen. Graph neural networks (gnns) based accelerated numerical simulation. *Engineering Applications of Artificial Intelligence*, 123:106370, 2023.

[66] Marco Maurizi, Chao Gao, and Filippo Berto. Predicting stress, strain and deformation fields in materials and structures with graph neural networks. *Scientific reports*, 12(1):21834, 2022.

[67] Minyi Dai, Mehmet F Demirel, Yingyu Liang, and Jia-Mian Hu. Graph neural networks for an accurate and interpretable prediction of the properties of polycrystalline materials. *npj Computational Materials*, 7(1):103, 2021.

[68] Roberto Perera, Davide Guzzetti, and Vinamra Agrawal. Graph neural networks for simulating crack coalescence and propagation in brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 395:115021, 2022.

[69] Konstantinos Karapiperis and Dennis M Kochmann. Prediction and control of fracture paths in disordered architected materials using graph neural networks. *Communications Engineering*, 2(1):32, 2023.

[70] Akhil Thomas, Ali Riza Durmaz, Mehwish Alam, Peter Gumbsch, Harald Sack, and Chris Eberl. Materials fatigue prediction using graph neural networks on microstructure representations. *Scientific Reports*, 13(1):12562, 2023.

[71] Emil Pitz and Kishore Pochiraju. A neural network transformer model for composite microstructure homogenization. *Engineering Applications of Artificial Intelligence*, 134:108622, 2024.

[72] Michael Okereke, Simeon Keates, M Okereke, and S Keates. Boundary conditions. *Finite Element Applications: A Practical Guide to the FEM Process*, pages 243–297, 2018.

[73] Scott Pecullan, LV Gibiansky, and Salvatore Torquato. Scale effects on the elastic behavior of periodic andhierarchical two-dimensional composites. *Journal of the Mechanics and Physics of Solids*, 47(7):1509–1542, 1999.

[74] M Ozgur Seydibeyoglu, Amar K Mohanty, and Manjusri Misra. *Fiber technology for fiber-reinforced composites*. Woodhead Publishing, 2017.

[75] Woei-Shyan Lee, Wu-Chung Sue, and Chi-Feng Lin. The effects of temperature and strain rate on the properties of carbon-fiber-reinforced 7075 aluminum alloy metal-matrix composite. *Composites Science and Technology*, 60(10):1975–1983, 2000.

[76] Vineet Chak, Himadri Chattopadhyay, and TL Dora. A review on fabrication methods, reinforcements and mechanical properties of aluminum matrix composites. *Journal of manufacturing processes*, 56:1059–1074, 2020.

[77] E Tikarrouchine, George Chatzigeorgiou, Francis Praud, Boris Piotrowski, Yves Chemisky, and Fodil Meraghni. Three-dimensional fe2 method for the simulation of non-linear, rate-dependent response of composite structures. *Composite Structures*, 193:165–179, 2018.

[78] Yang Shen, Robert H Moore, and Ankit Deo. Visualizing abaqus output database in paraview: A universal converter in python and c++. *SoftwareX*, 22:101331, 2023.

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.