Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Fast Prediction of Hoop Stress and Radial Displacement of Large-Diameter Pipe Jacking Segments Using Fourier Neural Operator

Fuhua ZHANG

tfuhuazhang@163.com

Shiqing HUANG

Research Article

Keywords: Large-diameter pipe jacking, Fourier neural operator, Hoop stress, Radial displacement, Fast prediction, Neural operator

Posted Date: November 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11247580/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Fast Prediction of Hoop Stress and Radial Displacement of Large-Diameter Pipe Jacking Segments Using Fourier Neural Operator

ZHANG Fuhua[1], HUANG Shiqing[1]

(1. *School of Mechanics and Civil Engineering, Jinan University*, *Guangzhou* 510630, *China*)

Abstract: The hoop stress and radial displacement of large-diameter prefabricated circular pipe jacking segments are key control indicators in structural design. Although analytical solutions can provide precise closed-form results, each parameter combination requires recalculation, making it difficult to support rapid large-scale comparison in parametric design. Finite element methods offer high accuracy but single solutions take minutes to hours. This paper introduces the Fourier Neural Operator (FNO) to learn the mapping from the design parameter space to the full-field solution function space (solution operator), achieving a fast surrogate model with "train once, millisecond-level inference for new cases". Based on the analytical solution of thick-walled cylinders under n=0~3 harmonic loads, 1500 training samples were generated in batches, and two independent FNO models were constructed for hoop stress prediction and radial displacement prediction, respectively. The results show that FNO predictions agree well with analytical solutions in circumferential distribution, full-wall contour plots, and parameter sensitivity. The mean RMSE of inner-wall hoop stress over 100 random test samples is 0.0212 MPa (max 0.1037 MPa), and the mean RMSE of radial displacement is 0.0209 mm (max 0.0650 mm), without the high-frequency spurious oscillations observed in the deep energy method. FNO provides an efficient tool for parametric design and rapid multi-scheme comparison of pipe jacking segments.

Key words: Large-diameter pipe jacking; Fourier neural operator; Hoop stress; Radial displacement; Fast prediction; Neural operator

First author: ZHANG Fuhua(1993–), male, currently a graduate student, registered structural engineer , is engaged in building structures and geotechnical engineering, E-mail: tfuhuazhang@163.com

***Corresponding author:** ZHANG Fuhua(1993–), male, currently a graduate student, registered structural engineer , is engaged in building structures and geotechnical engineering, E-mail: tfuhuazhang@163.com. ORCID: https://orcid.org/0009-0003-4702-0762

1 Introduction

As China's urbanization enters the stage of stock upgrading and quality improvement, the development and utilization of underground space is moving toward networking, deepening, and intensification. The construction demand for large-scale linear underground projects such as urban utility tunnels, underground logistics tunnels, and deep drainage systems is increasingly urgent. Pipe jacking, as the core trenchless construction method, has become the preferred solution for crossing urban core areas, traffic arteries, and sensitive structures due to its significant advantage of minimal interference with ground traffic, the environment, and existing facilities [1]. Currently, super-large-diameter pipe jacking projects of 10 m class and above are increasing. Such projects feature large segment cross-sections, wide variation in overburden thickness, high requirements for surrounding environmental protection [2], and frequent construction correction. The pipe-soil interaction mechanism is complex [3, 4], and problems such as significant jacking force fluctuations during construction, uneven structural loading of segments [5], and prominent risk of joint cracking in segmented segments pose higher requirements for design and calculation theory.

When traditional cast-in-place or monolithic prefabricated pipe jacking technology evolves toward super-large diameters (typically referring to inner diameter $\geqslant$3.5 m) and long distances, it faces two core bottlenecks: first, super-large-diameter prefabricated segments have "excessively wide and excessively high" transportation limitations, restricting project economy and feasibility; second,

the complex pipe-soil interaction and structural loading problems brought by large cross-sections make traditional design methods based on experience and simplified assumptions tend to be conservative, leading to material waste and high costs.

To solve the above problems, prefabricated pipe jacking technology has emerged. This technology prefabricates entire segments into several segments in the factory, transports them to the site for assembly, and then jacks them in, effectively solving the transportation problem and conforming to the development trend of building industrialization and green construction. However, the prefabricated structure introduces the weak link of "segment joints", whose mechanical properties (stiffness, waterproofing, durability) are directly related to the safety of the overall structure. At present, domestic research in this field is still in its infancy, lacking a complete set of design theories, stress analysis methods, and construction process standards for super-large-diameter, circular-cross-section, prefabricated pipe jacking.

Therefore, systematically conducting research on the structural design and stress analysis of super-large-diameter prefabricated pipe jacking has important theoretical and practical significance: theoretically, it can improve the mechanical analysis system of prefabricated underground structures and develop design methods considering the discontinuous characteristics of joints; practically, it can provide key technical support for the emerging deep underground space projects in Guangdong Province and even the whole country (such as deep drainage tunnels and large utility tunnels), promoting the development of China's pipe jacking technology toward larger diameters, higher prefabrication, and greater intelligence, serving the high-quality construction and renewal of urban infrastructure.

The above research mostly focuses on analytical derivation and finite element numerical simulation, while research on the application of data-driven neural operator methods in the rapid prediction of full-field stress and displacement of pipe jacking segments is still scarce. The authors previously used the Deep Energy Method Neural Network (DEM-NN) and Physics-Informed Neural Network (PINN) to solve this problem for single cases, verifying the feasibility of physics-driven neural networks, but both require retraining for each parameter combination, with prominent efficiency bottlenecks.

To address the above problems, this paper introduces the Fourier Neural Operator (FNO) to learn the mapping from the parameter space to the full-field solution function space (i.e., the solution operator), achieving a fast surrogate model with "train once, millisecond-level inference for new cases". FNO was proposed by Li Zongyi et al. [6]. Its core innovation lies in parameterizing the integral kernel in Fourier space and achieving efficient computation of global convolution through Fast Fourier Transform (FFT). It possesses two key properties: resolution invariance and universal operator approximation [7], and is currently the most widely used neural operator architecture in the field of scientific machine learning [8].

2 Analytical Solution Benchmark for Thick-Walled Cylinder

To verify the solution accuracy of FNO, this paper uses the plane-strain thick-walled cylinder analytical solution as the training data generation benchmark and comparison benchmark. This analytical solution is based on the polar coordinate theory of elasticity, expanding the non-uniform earth pressure on the outer wall into a Fourier series, solving order by order, and then linearly superimposing.

The example is a plane-strain thick-walled cylinder with inner radius $r_i = 5.0$ m, outer radius $R = 5.5$ m, and wall thickness 0.5 m, corresponding to an 11 m-class large-diameter pipe jacking segment. The material is C50 concrete with elastic modulus $E = 3.45 \times 10^{10}$ Pa and Poisson's ratio $\nu = 0.2$. Tensile stress is positive and compressive stress is negative; radial displacement outward is positive. The inner wall is completely free ($\sigma_r = 0$， $\tau_{r\theta} = 0$); only normal distributed pressure acts on the outer wall, and the circumferential shear stress is 0.

Soil parameters are taken as: $\gamma = 19.0$ kN/m³, $K_0 = 0.626$， $\varphi = 22°$， $K_p = 2.198$， $c = 18.0$ kPa. The plastic radius ratio is taken as 1.22 at the pipe crown, 1.20 at the pipe side, and 1.11 at the pipe invert, respectively. The normal pressure on the outer wall adopts the form of a 0~3 order cosine Fourier series:

$$p(\theta) = p_0 + p_1\cos\theta + p_2\cos 2\theta + p_3\cos 3\theta$$

The coefficients of each order are $p_0 = 213.82$ kPa, $p_1 = -35.53$ kPa, $p_2 = -5.87$ kPa, $p_3 = -0.017$ kPa, and the radial stress boundary condition on the outer wall is $\sigma_r(R,\theta) = -p(\theta)$.

For n-order harmonic loading, the general stress formula for the thick-walled cylinder is:

$n = 0$-order (axisymmetric):

$$\sigma_{r0} = A_0 + \frac{B_0}{r^2}, \sigma_{\theta 0} = A_0 - \frac{B_0}{r^2}$$

$n = 1$-order:

$$\sigma_{r1} = \left(2A_1 r + \frac{2D_1}{r^3}\right)\cos\theta, \sigma_{\theta 1} = \left(6A_1 r - \frac{2D_1}{r^3}\right)\cos\theta$$

$n = 2$-order:

$$\sigma_{r2} = \left(-2B_2 - \frac{4C_2}{r^2} - \frac{6D_2}{r^4}\right)\cos 2\theta$$

$$\sigma_{\theta 2} = \left(12A_2 r^2 + 2B_2 + \frac{6D_2}{r^4}\right)\cos 2\theta$$

$$\tau_{r\theta 2} = \left(6A_2 r^2 + 2B_2 - \frac{2C_2}{r^2} - \frac{6D_2}{r^4}\right)\sin 2\theta$$

$n = 3$-order:

$$\sigma_{r3} = \left(-4A_3 r^3 - 6B_3 r - \frac{10C_3}{r^3} - \frac{12D_3}{r^5}\right)\cos 3\theta$$

$$\sigma_{\theta 3} = \left(20A_3 r^3 + 6B_3 r + \frac{2C_3}{r^3} + \frac{12D_3}{r^5}\right)\cos 3\theta$$

The undetermined coefficients of each order are determined by the boundary conditions (inner wall $\sigma_r = 0$、$\tau_{r\theta} = 0$; outer wall $\sigma_r = -p_n$、$\tau_{r\theta} = 0$). Substituting the inner-wall free and outer-wall known loading boundary conditions, after solving each order and linear superposition, the analytical solution distribution of inner-wall hoop stress is obtained:

Inner-wall hoop stress ($r = 5.0$ m):

$$\sigma_\theta(5.0,\theta) = (-2464.02 + 407.6\cos\theta - 1423.76\cos 2\theta - 1.64\cos 3\theta) \text{ kPa}$$

This analytical solution provides an accurate, controllable, and noise-free sample source for generating FNO training data.

3 Neural Operator and Fundamentals of FNO

3.1 Mathematical Foundation of Operator Learning

Traditional neural networks learn mappings between finite-dimensional Euclidean spaces $\mathbb{R}^n \to \mathbb{R}^m$, while the solution operator of partial differential equations (PDEs) is a mapping between function spaces:

$$\mathcal{G}: \mathcal{A} \to \mathcal{U}$$

where $\mathcal{A}$ is the input function space (e.g., parameter field, initial conditions, boundary conditions) and $\mathcal{U}$ is the output solution function space (e.g., stress field, displacement field). Chen T and Chen H proved in 1995 that neural networks can uniformly approximate any nonlinear continuous operator, laying the theoretical foundation for operator learning [9].

Kovachki et al. established a general mathematical framework for neural operators in 2023 [7]: by alternately stacking linear integral operator layers and nonlinear activation functions, constructing a network layer of the form

$$v_{l+1}(x) = \sigma\left(W_l v_l(x) + \int_{\mathcal{D}} \kappa_l(x,y) v_l(y) dy\right)$$

network layer, and proves that this framework simultaneously possesses resolution invariance (the network can directly infer at another resolution after training at one resolution without retraining) and universal approximation (it can approximate any nonlinear continuous operator), making it the only deep learning model that currently possesses both properties [7].

3.2 FNO Architecture

The core idea of FNO is to parameterize the above integral kernel $\kappa_l(x, y)$ in Fourier space, using the convolution theorem to convert spatial-domain integration into frequency-domain element-wise multiplication, and computing efficiently through FFT. The single-layer transformation of FNO is:

$$v_{l+1}(x) = \sigma(W_l v_l(x) + \mathcal{F}^{-1}(R_l \cdot \mathcal{F}(v_l))(x))$$

where $\mathcal{F}$ is the Fast Fourier Transform, R_l is the frequency-domain learnable parameter (only the first k low-frequency modes are retained, achieving frequency-domain low-rank approximation), W_l is the point-wise linear transformation, and σ is the nonlinear activation function.

The complete FNO network consists of three parts:

1. **Lifting layer (P): lifts the input function to a high-dimensional feature space;**
2. **Fourier layer stack: multiple layers of the above Fourier integral operator layers to extract global spatial dependencies;**
3. **Projection layer (Q): projects high-dimensional features back to the target output field.**

The key advantages of FNO are [6, 7]:

Resolution-independent: training and inference can use different resolutions, achieving zero-shot super-resolution;

Global receptive field: Fourier transform naturally captures global dependencies without deep stacking;

Computationally efficient: FFT complexity is $\mathcal{O}(N \log N)$, much lower than direct integration $\mathcal{O}(N^2)$.

3.3 Essential Differences from DEM-NN and PINN

FNO, Deep Energy Method Neural Network (DEM-NN), and Physics-Informed Neural Network (PINN) represent two fundamentally different technical routes for artificial intelligence to solve mechanics problems. Their core differences are shown in Table 1.

Table 1 Comparison of FNO with DEM-NN and PINN

Comparison Dimension	DEM-NN	PINN	FNO (this paper)
Driving Method	Physics-driven (energy functional minimization)	Physics-driven (PDE residual constraint)	Data-driven (analytical solution samples)
Solution Scope	Single case	Single case	Multiple cases (solution operator)
Cost for New Case	Retraining	Retraining	One forward propagation (millisecond-level)
Network Output	Displacement → stress by differentiation	Mixed displacement + stress output	Direct stress/displacement field output
Resolution	Fixed collocation points	Fixed collocation points	Resolution-independent
Running Variability	Yes	Yes	No (deterministic inference after training)
Physical Constraint	Variational principle	Equilibrium equation + constitutive relation	Physics implicit in training data

The advantage of DEM-NN and PINN is that they can solve without training data, applicable to scenarios without analytical solutions or numerical simulation data; however, their single-case

characteristic determines that they cannot directly support rapid iteration in parametric design. FNO sacrifices the ability to "solve without data" in exchange for the efficiency advantage of "train once, rapid inference for multiple cases", and the two are complementary.

4 Development History and Research Frontiers

4.1 Theoretical Foundation Stage (1995–2019)

The theoretical roots of operator learning can be traced back to the universal approximation theorem for operators proved by Chen T and Chen H in 1995 [9]. This theorem shows that neural networks with arbitrary nonlinear activation functions can uniformly approximate any nonlinear continuous operator, laying the mathematical foundation for the subsequent development of neural operators.

From 2019 to 2020, breakthrough progress was made in the field of neural operators. Lu et al. proposed the DeepONet architecture in 2021 [10], adopting a dual-network structure of "Branch Net + Trunk Net" to encode the input function and evaluation coordinates respectively, realizing operator learning in practice for the first time and proving its universal approximation based on the Chen-Chen theorem. Li et al. proposed FNO in 2021 [6], parameterizing the integral kernel in Fourier space and using FFT for efficient computation, achieving state-of-the-art accuracy on benchmark problems such as the Navier-Stokes equations, with inference speed improved by approximately 1000 times compared to traditional numerical solvers.

4.2 Theoretical Refinement and Architecture Expansion Stage (2021–2023)

Kovachki et al. published a review paper in the Journal of Machine Learning Research in 2023 [7], establishing a general mathematical framework for neural operators, rigorously proving resolution invariance and universal approximation, and systematically reviewing the theoretical foundations and applicable conditions of various architectures such as FNO, DeepONet, and Graph Neural Operator.

During this period, various improved FNO architectures were proposed successively: Wen et al. proposed U-FNO in 2022 [11], combining the multi-scale structure of U-Net with FNO to improve accuracy in multiphase flow problems; Li et al. proposed Geo-FNO in 2022 [12], enabling FNO to handle arbitrary geometric domains through geometric mapping, breaking the limitation of original FNO to regular rectangular grids; Li et al. proposed the Physics-Informed Neural Operator (PINO) in 2021 [13], combining physical constraints with FNO to improve generalization ability in data-scarce scenarios.

4.3 Large-Scale Application and Frontier Exploration Stage (2022–Present)

FNO has achieved significant application results in multiple scientific and engineering fields. Pathak et al. proposed FourCastNet in 2022 [14], realizing global high-resolution weather forecasting based on adaptive FNO, with inference speed improved by approximately 45,000 times compared to traditional numerical weather forecasting, marking the successful application of neural operators in large-scale real physical systems. Rashid et al. applied FNO to stress-strain field prediction in digital composites in 2022 [15], verifying the effectiveness of FNO in full-field quantity prediction in solid mechanics. Kaewnuratchadasorn et al. applied FNO to vehicle-bridge coupled vibration simulation and bridge health monitoring in 2024 [16]. In the field of solid mechanics, physics-informed neural operators have been used to solve elasticity equations [17], local stress field prediction [18], and other problems.

In the field of underground engineering, the application of neural operators is still in its infancy. Tristani et al. used bagging ensemble and artificial neural networks to predict tunnel support pressure and radial displacement in 2024 [19], but employed traditional machine learning methods rather than

neural operators. Xu et al. proposed a multi-fidelity DeepONet framework in 2024 [20], fusing numerical simulation and monitoring data to achieve real-time prediction of ground settlement during tunnel construction, which is an early application of neural operators in tunnel engineering, but the research object was the ground settlement field rather than the internal forces of segment structures.

Azizzadenesheli et al. published a review in Nature Reviews Physics in 2024 [8], systematically summarizing the applications of neural operators in scientific simulation and design, pointing out that operator learning is expanding from benchmark PDE problems to real engineering systems, with data efficiency, physical consistency, and complex geometry handling as current frontier directions.

Through Chinese and English literature searches, the application of FNO to the rapid prediction of full-field hoop stress and radial displacement of large-diameter circular pipe jacking segments has not been reported in the literature.

5 Application Value of FNO in Pipe Jacking Engineering

5.1 Necessity of Introducing FNO under the Analytical Solution Framework

Chapter 2 has established the complete analytical solution for stress and displacement of thick-walled rings under n=0~3 harmonic loads. Given any geometric and loading parameters, precise closed-form solutions can be obtained. On this basis, the necessity of introducing the data-driven FNO method needs to be elucidated from three levels: method positioning, scope of application, and computational efficiency.

(1) Method positioning: proof of concept based on analytical solutions. This paper uses analytical solutions to generate training data in batches. Its primary purpose is to utilize the accurate, controllable, and noise-free characteristics of analytical solutions to verify the feasibility and accuracy upper limit of FNO as a solution operator learning method in thick-walled ring mechanics problems. This is proof-of-concept research—only by fully verifying the learning ability and prediction accuracy of FNO on simplified problems with precise benchmark solutions can we lay the foundation for subsequently extending this method to complex engineering problems without analytical solutions.

(2) Scope of application: expansion from ideal assumptions to complex engineering conditions. The analytical solution in Chapter 2 is established on strict assumptions such as homogeneous elastic thick-walled cylinder, ideal circular cross-section, homogeneous isotropic stratum, finite-order Fourier loading, and neglect of joint effects and construction disturbance. In actual large-diameter pipe jacking projects, the above assumptions are difficult to fully satisfy: strata often exhibit heterogeneous or layered characteristics, the stiffness of prefabricated segment joints is significantly lower than that of the segment body, effects such as grouting pressure, jacking deviation, and stratum loss during construction are difficult to incorporate into the analytical framework, and elastic solutions no longer apply after segments locally enter the plastic stage. When the problem complexity exceeds the applicable scope of analytical solutions, numerical methods such as the finite element method become the main analysis tools. As a data-driven operator learning method, FNO's training data can come from numerical simulation, field monitoring, or model tests, unrestricted by the existence of analytical solutions, and thus has a much wider scope of application than analytical solutions.

(3) Computational efficiency: advantages in large-scale batch inference scenarios. Even within the applicable scope of analytical solutions, when tasks requiring tens of thousands or more repeated solutions such as parameter optimization design, parameter inversion, reliability analysis, or full-parameter-space sensitivity scanning are needed, point-by-point invocation of analytical solution functions may still constitute a computational bottleneck. FNO's forward propagation is pure matrix operations, enabling batch parallel inference on GPUs, with total time significantly lower than point-by-point analytical computation in large-scale solving scenarios. In addition, FNO possesses resolution invariance [7], enabling direct inference at

high-resolution grids after training at low-resolution grids, achieving zero-shot super-resolution, a feature not available in analytical solution programming.

5.2 Advantages over the Finite Element Method

In complex engineering problems without analytical solutions, the finite element method is the current mainstream analysis tool. FNO has the following essential advantages over the finite element method:

(1) Inference speed improved by several orders of magnitude. Finite element solution requires assembling the stiffness matrix and solving large-scale linear equations, with a single computation taking from minutes to hours. After FNO training is completed, a new case only requires one forward propagation to obtain the full-field solution, with inference time on the order of milliseconds [6], and the speed improvement can reach more than three orders of magnitude, shortening parametric design comparison that originally took days to the minute level.

(2) Direct full-field output. The finite element method uses nodal displacement as the basic unknown, and stress needs to be obtained through post-processing of strain-displacement relations and constitutive relations, with discontinuous stress at element boundaries. FNO can directly output continuous and smooth stress and displacement fields without additional post-processing.

(3) Differentiability supports gradient-based optimization and inversion. The FNO network is differentiable throughout and can be directly embedded in the gradient descent optimization framework to achieve tasks such as segment wall thickness optimization and stratum parameter inversion. Although finite element models can calculate gradients through the adjoint method, the implementation is complex and computationally expensive.

5.3 Advantages over DEM-NN and PINN

Both DEM-NN and PINN are physics-driven single-case solving methods, requiring training a new network from scratch for each parameter combination, and there is variability in running results due to random initialization. As a data-driven operator learning method, FNO has the following distinguishing features:

(1) Train once, infer for multiple cases. DEM-NN and PINN require retraining for thousands of steps when changing parameter combinations, while FNO only requires one forward propagation, with computational costs differing by several orders of magnitude.

(2) Deterministic inference results. After FNO training is completed, it constitutes a deterministic mapping from the parameter space to the solution space. Multiple inferences for the same parameter combination yield completely consistent results, without the running variability problem of physics-driven methods.

(3) One-time coverage of the parameter space. After using analytical solutions or numerical simulations to batch generate training data covering the target parameter space, FNO can learn the solution operator within the entire parameter space at once, rather than solving each case independently.

5.4 Positioning and Innovation of This Work

In summary, the positioning of this work is: for the first time, verifying the feasibility and accuracy of FNO as a fast surrogate model in the mechanical analysis of large-diameter pipe jacking segments. Using the analytical solution from Chapter 2 to generate high-quality training data, it is proven that FNO can accurately learn the solution operator from design parameters to full-field hoop stress and radial displacement, providing a methodological foundation for subsequently extending this method to complex engineering problems without analytical solutions such as heterogeneous strata, prefabricated joints, and plastic analysis.

6 Case Setup, Complete Code, and Result Analysis

6.1 Training Data Generation

Based on the analytical solution in Chapter 2, training samples are generated by random sampling within the following parameter space:

Segment outer radius $R \in [4.0,7.0]$ m

Segment wall thickness $t \in [0.3,0.8]$ m

Pipe crown burial depth $H \in [15.0,30.0]$ m

Soil parameters are: $\gamma = 19.0\ \mathrm{kN/m^3}$, $K_0 = 0.626$, $\varphi = 22°$, $K_p = 2.198$, $c = 18.0$ kPa. The segment material is C50 concrete, $E = 3.45 \times 10^7$ kPa, $\nu = 0.2$. The plastic radius ratio is taken as 1.22 at the pipe crown, 1.20 at the pipe side, and 1.11 at the pipe invert, respectively.

A total of 1500 valid samples were generated (excluding unreasonable combinations of $r_i <$ 3.0 m), each containing full-field data of hoop stress and radial displacement on a grid (73 circumferential points × 20 radial points). Four representative parameter sets were selected as test cases, as shown in Table 2.

Table 2 Parameters of Test Cases

Case	R (m)	t (m)	H (m)	r_i (m)
Case 1	4.0	0.4	28	3.6
Case 2	5.0	0.6	30	4.4
Case 3	6.0	0.5	18	5.5
Case 4	7.0	0.8	25	6.2

6.2 Model Setup

Two independent FNO 2D models are constructed for hoop stress prediction and radial displacement prediction, respectively. The input is a 5-channel field. The network adopts 4 Fourier layers, retaining (R, t, H, θ, r) modes in the frequency domain, with a feature width of 64 and GELU activation function. Training for 80 epochs, Adam optimizer with initial learning rate 12×8, StepLR decaying by 0.5 every 20 epochs, batch size 20, GPU acceleration (NVIDIA RTX 3080). Weight decay 10^{-3}. Input features are normalized by mean and variance, stress output scaling factor 5000, displacement output scaling factor 0.01.

6.3 Complete Implementation Code

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as \(\mathcal{F}\)
import matplotlib.pyplot as plt
import matplotlib as mpl
import time
import os

mpl.rcParams.update({
    'font.size': 16, 'axes.labelsize': 18, 'axes.titlesize': 18,
    'xtick.labelsize': 16, 'ytick.labelsize': 16, 'legend.fontsize': 15,
    'figure.dpi': 120, 'savefig.dpi': 300, 'savefig.bbox': 'tight',
})
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'[Device] {device}')
O\(\mathcal{U}\)T_DIR = './fno_811_output'
os.makedirs(O\(\mathcal{U}\)T_DIR, exist_ok=True)
```

```
# ======================================================================
# 1. Soil parameters (thesis 7.1.1 / Table 7-1, directly used)
# ======================================================================
gamma = 19.0          # kN/m^3
K0 = 0.626
phi = np.deg2rad(22.0)
sin_phi = np.sin(phi)
cos_phi = np.cos(phi)
Kp = 2.198
c = 18.0         # kPa
B_coh = 2 * c * np.sqrt(Kp) / (Kp - 1)
Rp_ratio = {'top': 1.22, 'side': 1.20, 'bottom': 1.11}
# Segment parameters (C50 concrete)
E_pipe = 3.45e7 # kPa
nu  pipe = 0.2

# ======================================================================
# 2. Reaming pressure + Fourier coefficients (thesis 7.2 / 7.4 / 8.7)
# ======================================================================
def get  coefficients(R, t, H):
    ri = R - t
    h = {'top': H - R, 'side': H, 'bottom': H + R}
    p = {}
    for key in ['top', 'side', 'bottom']:
        sigma_h0 = K0 * gamma * h[key]
        sigma_rp = sigma_h0 * (1 - sin_phi) + c * cos_phi
        p[key] = (sigma_rp + B_coh) * (Rp_ratio[key])**(Kp - 1) - B_coh
    p0_2 = (p['top'] + p['bottom']) / 4.0 + p['side'] / 2.0
    p1_2 = (p['top'] - p['bottom']) / 2.0
    p2_2 = (p['top'] + p['bottom']) / 4.0 - p['side'] / 2.0
    p_60 = p0_2 + p1_2 * np.cos(np.deg2rad(60)) + p2_2 * np.cos(np.deg2rad(120))
    p_120 = p0_2 + p1_2 * np.cos(np.deg2rad(120)) + p2_2 * np.cos(np.deg2rad(240))
    \(\mathcal{A}\)_mat = np.array([
        [1, 0.5, -0.5, -1.0],
        [1, 1.0, 1.0, 1.0],
        [1, -0.5, -0.5, 1.0],
        [1, -1.0, 1.0, -1.0],
    ])
    p0, p1, p2, p3 = np.linalg.solve(
        \(\mathcal{A}\)  mat, np.array([p['top'], p  60, p  120, p['bottom']]))
    # \(n=0\)
    sigma_r0_R = -p0
    \(\mathcal{A}\)0 = sigma_r0_R * R**2 / (R**2 - ri**2)
    B0 = sigma_r0_R * R**2 * ri**2 / (R**2 - ri**2)
    # \(n=1\)
    sigma_r1_R = -p1
    \(\mathcal{A}\)1 = sigma_r1_R * R**3 / (2.0 * (R**4 - ri**4))
    D1 = \(\mathcal{A}\)1 * ri**4
    # \(n=2\) (correct formula: inner wall sigma_r=0, tau=0; outer wall sigma_r=-p2, tau=0)
    sigma_r2_R = -p2
    M2 = np.array([
        [0,    -2, -4/ri**2, -6/ri**4],
        [6*ri**2, 2, -2/ri**2, -6/ri**4],
        [0,    -2, -4/R**2, -6/R**4],
        [6*R**2, 2, -2/R**2, -6/R**4],
    ])
    \(\mathcal{A}\)2, B2, C2, D2 = np.linalg.solve(M2, np.array([0, 0, sigma_r2_R, 0]))
    # \(n=3\)
    sigma_r3_R = -p3
    M3 = np.array([
        [-4*ri**3, -6*ri, -10/ri**3, -12/ri**5],
```

```
        [12*ri**3, 6*ri, -6/ri**3, -12/ri**5],
        [-4*R**3, -6*R, -10/R**3, -12/R**5],
        [12*R**3, 6*R, -6/R**3, -12/R**5],
    ])
    \(\mathcal{A}\)3, B3, C3, D3 = np.linalg.solve(M3, np.array([0, 0, sigma_r3_R, 0]))
    return {'\(\mathcal{A}\)0': \(\mathcal{A}\)0, 'B0': B0, '\(\mathcal{A}\)1': \(\mathcal{A}\)1, 'D1': D1,
            '\(\mathcal{A}\)2': \(\mathcal{A}\)2, 'B2': B2, 'C2': C2, 'D2': D2,
            '\(\mathcal{A}\)3': \(\mathcal{A}\)3, 'B3': B3, 'C3': C3, 'D3': D3}

def sigma_at_r(coef, r, theta_deg):
    th = np.deg2rad(theta_deg)
    s0 = coef['\(\mathcal{A}\)0'] + coef['B0'] / r**2
    s1 = 6.0 * coef['\(\mathcal{A}\)1'] * r + 2.0 * coef['D1'] / r**3
    s2 = 12.0 * coef['\(\mathcal{A}\)2'] * r**2 + 2.0 * coef['B2'] + 6.0 * coef['D2'] / r**4
    s3 = (20.0 * coef['\(\mathcal{A}\)3'] * r**3 + 6.0 * coef['B3'] * r
          + 2.0 * coef['C3'] / r**3 + 12.0 * coef['D3'] / r**5)
    return s0 + s1*np.cos(th) + s2*np.cos(2*th) + s3*np.cos(3*th)

def displacement_at_r(coef, r, theta_deg, E=E_pipe, nu=nu_pipe):
    th = np.deg2rad(theta_deg)
    f = (1 + nu) / E
    d0 = f * ((1 - 2*nu) * coef['\(\mathcal{A}\)0'] * r + coef['B0'] / r)
    d1 = f * ((1 - 3*nu) * coef['\(\mathcal{A}\)1'] * r**2
              + (1 + nu) * coef['D1'] / r) * np.cos(th)
    d2 = f * (-4*nu/(1-nu) * coef['\(\mathcal{A}\)2'] * r**3 - 2*(1+nu) * coef['B2'] * r
              + 4*(1-2*nu)/(1-nu) * coef['C2'] / r
              + 2*(1+nu)/(1-nu) * coef['D2'] / r**3) * np.cos(2*th)
    d3 = f * (-(1+4*nu) * coef['\(\mathcal{A}\)3'] * r**4 - 3 * coef['B3'] * r**2
              + (5-4*nu) * coef['C3'] / r + 3 * coef['D3'] / r**3) * np.cos(3*th)
    return d0 + d1 + d2 + d3

def analytical_field(R, t, H, theta_arr, r_arr, field_type='stress'):
    coef = get_coefficients(R, t, H)
    field = np.zeros((len(theta_arr), len(r_arr)))
    for j, th in enumerate(theta_arr):
        for k, rr in enumerate(r_arr):
            if field_type == 'stress':
                field[j, k] = sigma_at_r(coef, rr, th)
            else:
                field[j, k] = displacement_at_r(coef, rr, th)
    return field

# ============================================================
# 3. \(\mathcal{F}\)NO2d network
# ============================================================
class SpectralConv2d(nn.Module):
    def __init__(self, in_ch, out_ch, modes1, modes2):
        super().__init__()
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.modes1 = modes1
        self.modes2 = modes2
        scale = 1.0 / (in_ch * out_ch)
        self.weights1 = nn.Parameter(
            scale * torch.randn(in_ch, out_ch, modes1, modes2, dtype=torch.cfloat))
        self.weights2 = nn.Parameter(
            scale * torch.randn(in_ch, out_ch, modes1, modes2, dtype=torch.cfloat))
    def forward(self, x):
        B = x.shape[0]
        x_ft = torch.fft.rfft2(x)
        out_ft = torch.zeros(B, self.out_ch, x.size(-2), x.size(-1)//2 + 1,
                             device=x.device, dtype=torch.cfloat)
```

```
        out_ft[:, :, :self.modes1, :self.modes2] = torch.einsum(
            'bixy,ioxy->boxy',
            x_ft[:, :, :self.modes1, :self.modes2], self.weights1)
        out_ft[:, :, -self.modes1:, :self.modes2] = torch.einsum(
            'bixy,ioxy->boxy',
            x_ft[:, :, -self.modes1:, :self.modes2], self.weights2)
        return torch.fft.irfft2(out_ft, s=(x.size(-2), x.size(-1)))

class \(\mathcal{F}\)NO2d(nn.Module):
    def __init__(self, in_ch=5, width=64, modes1=12, modes2=8, depth=4):
        super().__init__()
        self.fc_in = nn.Linear(in_ch, width)
        self.convs = nn.ModuleList(
            [SpectralConv2d(width, width, modes1, modes2) for _ in range(depth)])
        self.skips = nn.ModuleList(
            [nn.Conv2d(width, width, 1) for    in range(depth)])
        self.fc_out1 = nn.Linear(width, 128)
        self.fc_out2 = nn.Linear(128, 1)
    def forward(self, x):
        x = self.fc_in(x)
        x = x.permute(0, 3, 1, 2)
        for conv, skip in zip(self.convs, self.skips):
            x = \(\mathcal{F}\).gelu(conv(x) + skip(x))
        x = x.permute(0, 2, 3, 1)
        x = \(\mathcal{F}\).gelu(self.fc_out1(x))
        x = self.fc_out2(x)
        return x.squeeze(-1)

# ============================================================
# 4. Generate dataset (1500 samples)
# ============================================================
N_TR\(\mathcal{A}\)IN = 1500
N_THET\(\mathcal{A}\) = 73
N_R = 20
theta_grid = np.linspace(0, 360, N_THET\(\mathcal{A}\))
R_MIN, R_M\(\mathcal{A}\)X = 4.0, 7.0
T_MIN, T_M\(\mathcal{A}\)X = 0.3, 0.8
H_MIN, H_M\(\mathcal{A}\)X = 15.0, 30.0
X_mean = np.array([5.5, 0.55, 22.5, 180.0, 5.0], dtype=np.float32)
X_std = np.array([1.5, 0.25, 7.5, 180.0, 2.0], dtype=np.float32)
Y_SC\(\mathcal{A}\)LE_S = 5000.0 # Stress scaling (kPa)
Y_SC\(\mathcal{A}\)LE_D = 0.01 # Displacement scaling (m)

def generate_dataset(N=N_TR\(\mathcal{A}\)IN, seed=42):
    rng = np.random.RandomState(seed)
    R_arr = rng.uniform(R_MIN, R_M\(\mathcal{A}\)X, N)
    t_arr = rng.uniform(T_MIN, T_M\(\mathcal{A}\)X, N)
    H_arr = rng.uniform(H_MIN, H_M\(\mathcal{A}\)X, N)
    X = np.zeros((N, N_THET\(\mathcal{A}\), N_R, 5), dtype=np.float32)
    Ys = np.zeros((N, N_THET\(\mathcal{A}\), N_R), dtype=np.float32)
    Yd = np.zeros((N, N_THET\(\mathcal{A}\), N_R), dtype=np.float32)
    valid = 0
    t0 = time.time()
    for i in range(N):
        R, t, H = R_arr[i], t_arr[i], H_arr[i]
        ri = R - t
        if ri < 3.0:
            continue
        r_grid = np.linspace(ri, R, N_R)
        TH, RR = np.meshgrid(theta_grid, r_grid, indexing='ij')
        X[valid, :, :, 0] = R
        X[valid, :, :, 1] = t
```

```
        X[valid, :, :, 2] = H
        X[valid, :, :, 3] = TH
        X[valid, :, :, 4] = RR
        Ys[valid, :, :] = analytical_field(R, t, H, theta_grid, r_grid, 'stress')
        Yd[valid, :, :] = analytical_field(R, t, H, theta_grid, r_grid, 'disp')
        valid += 1
        if (i+1) % 200 == 0:
            print(f' Generated {i+1}/{N}, valid {valid}, elapsed {time.time()-t0:.1f}s')
    X = X[:valid]
    Ys = Ys[:valid]
    Yd = Yd[:valid]
    Xn = (X - X_mean) / X_std
    Yns = Ys / Y_SC\(\mathcal{A}\)LE_S
    Ynd = Yd / Y_SC\(\mathcal{A}\)LE_D
    return Xn, Yns, Ynd, X

# ============================================================
# 5. Training (80 epochs, step_size=20, batch=20, print every epoch)
# ============================================================
def train_fno(model, Xtr, Ytr, Xte, Yte, epochs=80, lr=1e-3, name='model'):
    model = model.to(device)
    opt = torch.optim.\(\mathcal{A}\)dam(model.parameters(), lr=lr, weight_decay=1e-5)
    sched = torch.optim.lr_scheduler.StepLR(opt, step_size=20, gamma=0.5)
    Xtr_t = torch.tensor(Xtr, dtype=torch.float32).to(device)
    Ytr_t = torch.tensor(Ytr, dtype=torch.float32).to(device)
    Xte_t = torch.tensor(Xte, dtype=torch.float32).to(device)
    Yte_t = torch.tensor(Yte, dtype=torch.float32).to(device)
    bs = 20
    t0 = time.time()
    for ep in range(epochs):
        ep_t0 = time.time()
        model.train()
        idx = torch.randperm(len(Xtr_t))
        total = 0
        for i in range(0, len(Xtr_t), bs):
            b = idx[i:i+bs]
            pred = model(Xtr_t[b])
            loss = \(\mathcal{F}\).mse_loss(pred, Ytr_t[b])
            opt.zero_grad()
            loss.backward()
            opt.step()
            total += loss.item()
        sched.step()
        train_loss = total * bs / len(Xtr_t)
        ep_time = time.time() - ep_t0
        total  time = time.time() - t0
        if (ep+1) % 10 == 0 or ep == 0:
            model.eval()
            with torch.no_grad():
                te_loss = \(\mathcal{F}\).mse_loss(model(Xte_t), Yte_t).item()
            print(f' [{name}] Ep {ep+1:3d}/{epochs} '
                f'train={train_loss:.6f} test={te_loss:.6f} '
                f'epoch={ep_time:.2f}s total={total_time:.1f}s')
        else:
            print(f' [{name}] Ep {ep+1:3d}/{epochs} '
                f'train={train_loss:.6f} '
                f'epoch={ep_time:.2f}s total={total_time:.1f}s')
    model = model.cpu()
    return model

# ============================================================
# 6. Prediction
```

```
# ================================================================
def fno_predict(model, R, t, H, scale):
    model = model.to(device)
    model.eval()
    with torch.no_grad():
        ri = R - t
        r_grid = np.linspace(ri, R, N_R)
        TH, RR = np.meshgrid(theta_grid, r_grid, indexing='ij')
        x = np.zeros((1, N_THET\(\mathcal{A}\), N_R, 5), dtype=np.float32)
        x[0, :, :, 0] = R
        x[0, :, :, 1] = t
        x[0, :, :, 2] = H
        x[0, :, :, 3] = TH
        x[0, :, :, 4] = RR
        x_in = torch.tensor((x - X_mean) / X_std, dtype=torch.float32).to(device)
        y = model(x  in).cpu().numpy()[0] * scale
    return y, r_grid

# ================================================================
# 7. Visualization
# ================================================================
cases = [
    (4.0, 0.4, 28, 'Case 1: R=4.0 m, t=0.4 m, H=28 m'),
    (5.0, 0.6, 30, 'Case 2: R=5.0 m, t=0.6 m, H=30 m'),
    (6.0, 0.5, 18, 'Case 3: R=6.0 m, \(t=0.5\) m, H=18 m'),
    (7.0, 0.8, 25, 'Case 4: R=7.0 m, t=0.8 m, \(H=25\) m'),
]
theta_plot = np.linspace(0, 360, 200)

def plot_inner_wall_curves(model_s, model_d):
    # Stress curves
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    for ax, (R, t, H, title) in zip(axes.flatten(), cases):
        ri = R - t
        coef = get_coefficients(R, t, H)
        sa = np.array([sigma_at_r(coef, ri, th) for th in theta_plot]) / 1000.0
        sf_field, _ = fno_predict(model_s, R, t, H, Y_SC\(\mathcal{A}\)LE_S)
        sf = np.interp(theta_plot, theta_grid, sf_field[:, 0]) / 1000.0
        ax.plot(theta_plot, sa, 'r--', linewidth=2.2, label='\(\mathcal{A}\)nalytical solution')
        ax.plot(theta_plot, sf, 'b-', linewidth=1.8, label='\(\mathcal{F}\)NO prediction')
        ax.set  xlabel(r'Circumferential angle $\theta$ ($^\circ$)')
        ax.set_ylabel(r'Hoop stress $\sigma_\theta$ (MPa)')
        ax.set_title(f'{title} (inner wall)', fontsize=18)
        ax.set_xticks([0, 60, 120, 180, 240, 300, 360])
        ax.grid(True, alpha=0.3)
        ax.legend(loc='upper right', framealpha=0.9)
        rmse = np.sqrt(np.mean((sa - sf)**2))
        ax.text(0.03, 0.05, f'RMSE={rmse:.4f} MPa',
            transform=ax.trans\(\mathcal{A}\)xes, fontsize=15,
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
    plt.tight_layout()
    plt.savefig(f'{O\(\mathcal{U}\)T_DIR}/curves_stress.png')
    plt.show()
    plt.close()
    # Displacement curves
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    for ax, (R, t, H, title) in zip(axes.flatten(), cases):
        ri = R - t
        coef = get_coefficients(R, t, H)
        da = np.array([displacement_at_r(coef, ri, th) for th in theta_plot]) * 1000.0
        df_field, _ = fno_predict(model_d, R, t, H, Y_SC\(\mathcal{A}\)LE_D)
        df = np.interp(theta_plot, theta_grid, df_field[:, 0]) * 1000.0
```

```
        ax.plot(theta_plot, da, 'r--', linewidth=2.2, label='\(\mathcal{A}\)nalytical solution')
        ax.plot(theta_plot, df, 'b-', linewidth=1.8, label='\(\mathcal{F}\)NO prediction')
        ax.set_xlabel(r'Circumferential angle $\theta$ ($^\circ$)')
        ax.set_ylabel(r'Radial displacement $u_r$ (mm)')
        ax.set_title(f'{title} (inner wall)', fontsize=18)
        ax.set_xticks([0, 60, 120, 180, 240, 300, 360])
        ax.grid(True, alpha=0.3)
        ax.legend(loc='upper right', framealpha=0.9)
        rmse = np.sqrt(np.mean((da - df)**2))
        ax.text(0.03, 0.05, f'RMSE={rmse:.4f} mm',
            transform=ax.trans\(\mathcal{A}\)xes, fontsize=15,
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
    plt.tight_layout()
    plt.savefig(f'{O\(\mathcal{U}\)T_DIR}/curves_disp.png')
    plt.show()
    plt.close()

def plot_ring_clouds(model_s, model_d):
    for Rc, tc, Hc, ctitle in cases:
        ric = Rc - tc
        r  cld = np.linspace(ric, Rc, N  R)
        TH, RR = np.meshgrid(np.deg2rad(theta_grid), r_cld, indexing='ij')
        case_tag = ctitle.split(':')[0].replace(' ', '')
        # Stress contour plots
        s_ana = analytical_field(Rc, tc, Hc, theta_grid, r_cld, 'stress') / 1000.0
        s_fno, _ = fno_predict(model_s, Rc, tc, Hc, Y_SC\(\mathcal{A}\)LE_S)
        s_fno = s_fno / 1000.0
        s_err = np.abs(s_ana - s_fno)
        fig, axes = plt.subplots(1, 3, figsize=(15, 6.2),
                    subplot_kw=dict(projection='polar'))
        titles = [r'\(\mathcal{A}\)nalytical solution $\sigma_\theta$ (MPa)',
            r'\(\mathcal{F}\)NO prediction $\sigma_\theta$ (MPa)',
            r'\(\mathcal{A}\)bsolute error (MPa)']
        datas = [s_ana, s_fno, s_err]
        cmaps = ['jet', 'jet', 'hot_r']
        for ax, ti, da, cm in zip(axes, titles, datas, cmaps):
            ax.set_theta_zero_location('N')
            ax.set_theta_direction(-1)
            ax.pcolormesh(TH, RR, da, cmap=cm, shading='auto', edgecolors='none')
            ax.set_ylim(ric * 0.5, Rc * 1.08)
            ax.grid(\(\mathcal{F}\)alse)
            ax.spines['polar'].set_visible(\(\mathcal{F}\)alse)
            ax.set_yticklabels([])
            ax.set_xticklabels([])
            ax.set_title(f'{ti}\nR={Rc} m, t={tc} m, H={Hc} m',
                fontsize=16, pad=18)
            plt.colorbar(ax.collections[0], ax=ax, shrink=0.78, pad=0.03)
        plt.tight_layout(pad=1.0)
        plt.savefig(f'{O\(\mathcal{U}\)T_DIR}/ring_stress_{case_tag}.png')
        plt.show()
        plt.close()
        # Displacement contour plots
        d_ana = analytical_field(Rc, tc, Hc, theta_grid, r_cld, 'disp') * 1000.0
        d_fno, _ = fno_predict(model_d, Rc, tc, Hc, Y_SC\(\mathcal{A}\)LE_D)
        d_fno = d_fno * 1000.0
        d_err = np.abs(d_ana - d_fno)
        fig, axes = plt.subplots(1, 3, figsize=(15, 6.2),
                    subplot_kw=dict(projection='polar'))
        titles = [r'\(\mathcal{A}\)nalytical solution $u_r$ (mm)',
            r'\(\mathcal{F}\)NO prediction $u_r$ (mm)',
            r'\(\mathcal{A}\)bsolute error (mm)']
        datas = [d_ana, d_fno, d_err]
```

```
    for ax, ti, da, cm in zip(axes, titles, datas, cmaps):
        ax.set_theta_zero_location('N')
        ax.set_theta_direction(-1)
        ax.pcolormesh(TH, RR, da, cmap=cm, shading='auto', edgecolors='none')
        ax.set_ylim(ric * 0.5, Rc * 1.08)
        ax.grid(\(\mathcal{F}\)alse)
        ax.spines['polar'].set_visible(\(\mathcal{F}\)alse)
        ax.set_yticklabels([])
        ax.set_xticklabels([])
        ax.set_title(f'{ti}\nR={Rc} m, t={tc} m, H={Hc} m',
                fontsize=16, pad=18)
        plt.colorbar(ax.collections[0], ax=ax, shrink=0.78, pad=0.03)
    plt.tight_layout(pad=1.0)
    plt.savefig(f'{O\(\mathcal{U}\)T_DIR}/ring_disp_{case_tag}.png')
    plt.show()
    plt.close()

def plot_sensitivity(model_s, model_d):
    R_base, t_base, H_base = 5.5, 0.5, 25.0
    R_arr = np.linspace(4.0, 7.0, 25)
    t  arr = np.linspace(0.3, 0.8, 25)
    H_arr = np.linspace(15.0, 30.0, 25)
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    for row, (model, scale, ylabel, unit) in enumerate([
        (model_s, Y_SC\(\mathcal{A}\)LE_S, r'$\sigma_\theta$', 'MPa'),
        (model_d, Y_SC\(\mathcal{A}\)LE_D, r'$u_r$', 'mm')]):
        for col, (param_arr, param_name, fixed) in enumerate([
            (R_arr, 'R', {'t': t_base, 'H': H_base}),
            (t_arr, 't', {'R': R_base, 'H': H_base}),
            (H_arr, 'H', {'R': R_base, 't': t_base})]):
            ax = axes[row, col]
            ana_vals, fno_vals = [], []
            for pv in param_arr:
                if param_name == 'R':
                    R, t, H = pv, fixed['t'], fixed['H']
                elif param_name == 't':
                    R, t, H = fixed['R'], pv, fixed['H']
                else:
                    R, t, H = fixed['R'], fixed['t'], pv
                ri = R - t
                coef = get  coefficients(R, t, H)
                if row == 0:
                    ana_vals.append(sigma_at_r(coef, ri, 90) / 1000.0)
                    fld, _ = fno_predict(model, R, t, H, scale)
                    fno_vals.append(np.interp(90, theta_grid, fld[:, 0]) / 1000.0)
                else:
                    ana_vals.append(displacement_at_r(coef, ri, 90) * 1000.0)
                    fld, _ = fno_predict(model, R, t, H, scale)
                    fno_vals.append(np.interp(90, theta_grid, fld[:, 0]) * 1000.0)
            ax.plot(param_arr, ana_vals, 'r-o', linewidth=2, markersize=5,
                    label='\(\mathcal{A}\)nalytical solution')
            ax.plot(param_arr, fno_vals, 'b--s', linewidth=1.8, markersize=5,
                    label='\(\mathcal{F}\)NO prediction')
            if param_name == 'R':
                ax.set_xlabel('Pipe radius R (m)')
                title = f'R sensitivity\n(side, 90°, inner wall; t={t_base} m, H={H_base} m fixed)'
            elif param_name == 't':
                ax.set_xlabel('Wall thickness t (m)')
                title = f't sensitivity\n(side, 90°, inner wall; R={R_base} m, H={H_base} m fixed)'
            else:
                ax.set_xlabel('Buried depth H (m)')
                title = f'H sensitivity\n(side, 90°, inner wall; R={R_base} m, t={t_base} m fixed)'
```

```
        ax.set_ylabel(f'{ylabel} ({unit})')
        ax.set_title(title, fontsize=16)
        ax.grid(True, alpha=0.3)
        ax.legend(loc='best', framealpha=0.9)
    plt.tight_layout()
    plt.savefig(f'{O\(\mathcal{U}\)T_DIR}/sensitivity.png')
    plt.show()
    plt.close()

def plot_error_stats(model_s, model_d, N_test=100, seed=123):
    rng = np.random.RandomState(seed)
    s_rmse, d_rmse = [], []
    for _ in range(N_test):
        R = rng.uniform(R_MIN, R_M\(\mathcal{A}\)X)
        t = rng.uniform(T_MIN, T_M\(\mathcal{A}\)X)
        H = rng.uniform(H_MIN, H_M\(\mathcal{A}\)X)
        ri = R - t
        if ri < 3.0:
            continue
        coef = get_coefficients(R, t, H)
        sa = np.array([sigma_at_r(coef, ri, th) for th in theta_grid])
        sf, _ = fno_predict(model_s, R, t, H, Y_SC\(\mathcal{A}\)LE_S)
        s_rmse.append(np.sqrt(np.mean((sa - sf[:, 0])**2)) / 1000.0)
        da = np.array([displacement_at_r(coef, ri, th) for th in theta_grid])
        df, _ = fno_predict(model_d, R, t, H, Y_SC\(\mathcal{A}\)LE_D)
        d_rmse.append(np.sqrt(np.mean((da - df[:, 0])**2)) * 1000.0)
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
    axes[0].hist(s_rmse, bins=20, color='steelblue', edgecolor='black', alpha=0.8)
    axes[0].set_xlabel(r'RMSE of $\sigma_\theta$ (MPa)', fontsize=16)
    axes[0].set_ylabel('Count', fontsize=16)
    axes[0].set_title('Inner wall stress RMSE distribution', fontsize=18)
    axes[0].text(0.65, 0.85,
            f'Mean={np.mean(s_rmse):.4f} MPa\nMax={np.max(s_rmse):.4f} MPa',
            transform=axes[0].trans\(\mathcal{A}\)xes, fontsize=15,
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
    axes[0].grid(True, alpha=0.3)
    axes[1].hist(d_rmse, bins=20, color='coral', edgecolor='black', alpha=0.8)
    axes[1].set_xlabel(r'RMSE of $u_r$ (mm)', fontsize=16)
    axes[1].set_ylabel('Count', fontsize=16)
    axes[1].set_title('Inner wall displacement RMSE distribution', fontsize=18)
    axes[1].text(0.65, 0.85,
            f'Mean={np.mean(d_rmse):.4f} mm\nMax={np.max(d_rmse):.4f} mm',
            transform=axes[1].trans\(\mathcal{A}\)xes, fontsize=15,
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
    axes[1].grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(f'{O\(\mathcal{U}\)T_DIR}/error_stats.png')
    plt.show()
    plt.close()
    print(f' Stress RMSE mean={np.mean(s_rmse):.4f} MPa, '
        f'max={np.max(s_rmse):.4f} MPa')
    print(f' Displacement RMSE mean={np.mean(d_rmse):.4f} mm, '
        f'max={np.max(d_rmse):.4f} mm')

# ============================================================
# 8. Main program
# ============================================================
if __name__ == '__main__':
    print('[1] Generating dataset (1500 samples)...')
    Xn, Yns, Ynd, Xraw = generate_dataset(N=N_TR\(\mathcal{A}\)IN, seed=42)
    N_total = len(Xn)
    N_tr = int(0.9 * N_total)
```

```
idx_perm = np.random.RandomState(0).permutation(N_total)
Xtr, Xte = Xn[idx_perm[:N_tr]], Xn[idx_perm[N_tr:]]
Ystr, Yste = Yns[idx_perm[:N_tr]], Yns[idx_perm[N_tr:]]
Ydtr, Ydte = Ynd[idx_perm[:N_tr]], Ynd[idx_perm[N_tr:]]
print(f' Training set {len(Xtr)}, test set {len(Xte)}')
print('\n[2] Training stress \(\mathcal{F}\)NO (80 epochs)...')
model_s = \(\mathcal{F}\)NO2d(in_ch=5, width=64, modes1=12, modes2=8, depth=4)
model_s = train_fno(model_s, Xtr, Ystr, Xte, Yste, epochs=80, lr=1e-3, name='stress')
print('\n[3] Training displacement \(\mathcal{F}\)NO (80 epochs)...')
model_d = \(\mathcal{F}\)NO2d(in_ch=5, width=64, modes1=12, modes2=8, depth=4)
model_d = train_fno(model_d, Xtr, Ydtr, Xte, Ydte, epochs=80, lr=1e-3, name='disp')
print('\n[4] Plotting inner-wall curves...')
plot_inner_wall_curves(model_s, model_d)
print('\n[5] Plotting ring contour plots...')
plot_ring_clouds(model_s, model_d)
print('\n[6] Plotting parameter sensitivity...')
plot_sensitivity(model_s, model_d)
print('\n[7] Error statistics (100 random samples)...')
plot_error_stats(model_s, model_d, N_test=100, seed=123)
print('\n Done.')
```

6.4 Comparison of Hoop Stress Distribution

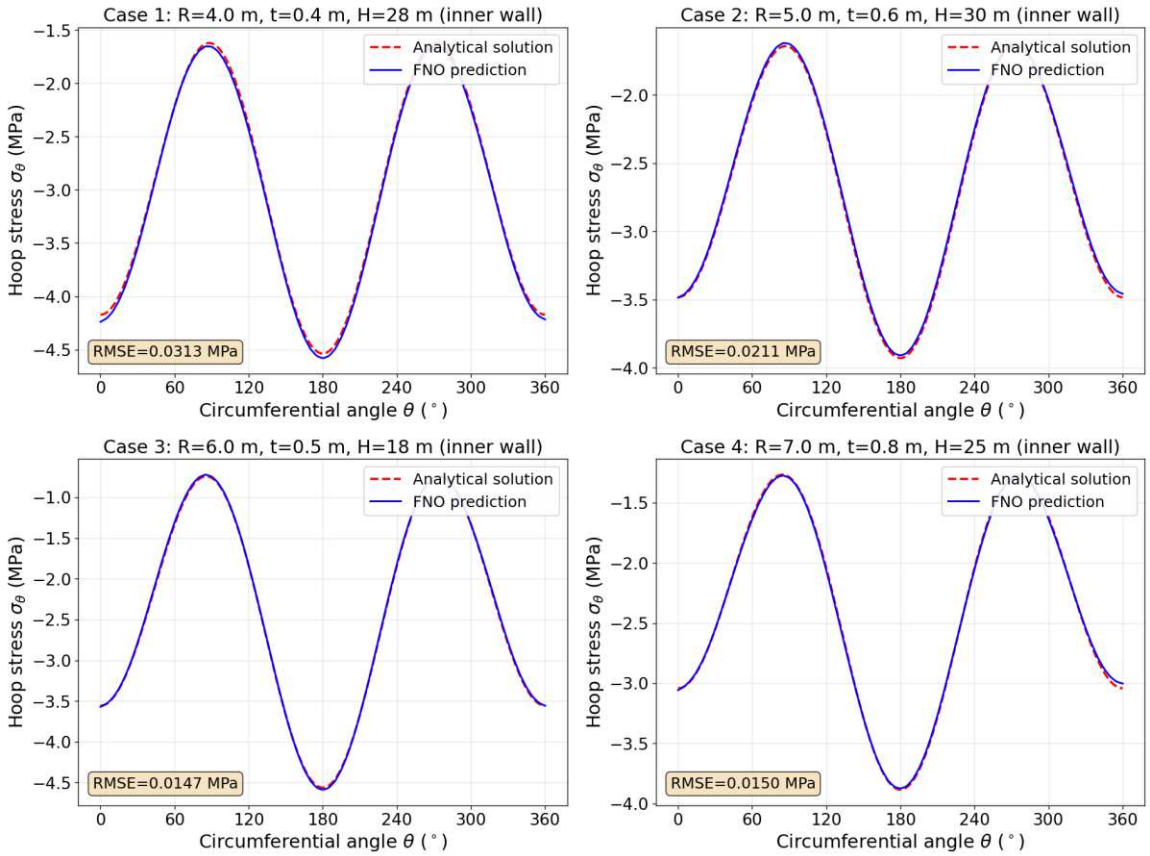


Fig. 1 Inner-Wall Hoop Stress

Figure 1 presents the distribution curves of inner-wall hoop stress σ_θ along the segment circumference (0°~360°) for four cases, where the red dashed line is the analytical solution result and the blue solid line is the FNO prediction result.

As can be seen from the figure, the hoop stress curves predicted by FNO are in high agreement with the analytical solutions, and the phase, amplitude, and variation trend of stress are accurately reproduced. All cases show the distribution pattern of stress concentration at the pipe crown (0°) and

pipe invert (180°) and smaller stress at the pipe sides (90°/270°), consistent with the mechanical characteristics of thick-walled rings under non-uniform loading. FNO can accurately capture the complex stress waveform after superposition of n=0~3 harmonic components, without the high-frequency spurious oscillations observed in DEM-NN. The full-circumference inner-wall stress RMSE for each case is 0.0313, 0.0211, 0.0147, and 0.0150 MPa, respectively, all within 0.04 MPa.

6.5 Comparison of Radial Displacement Distribution

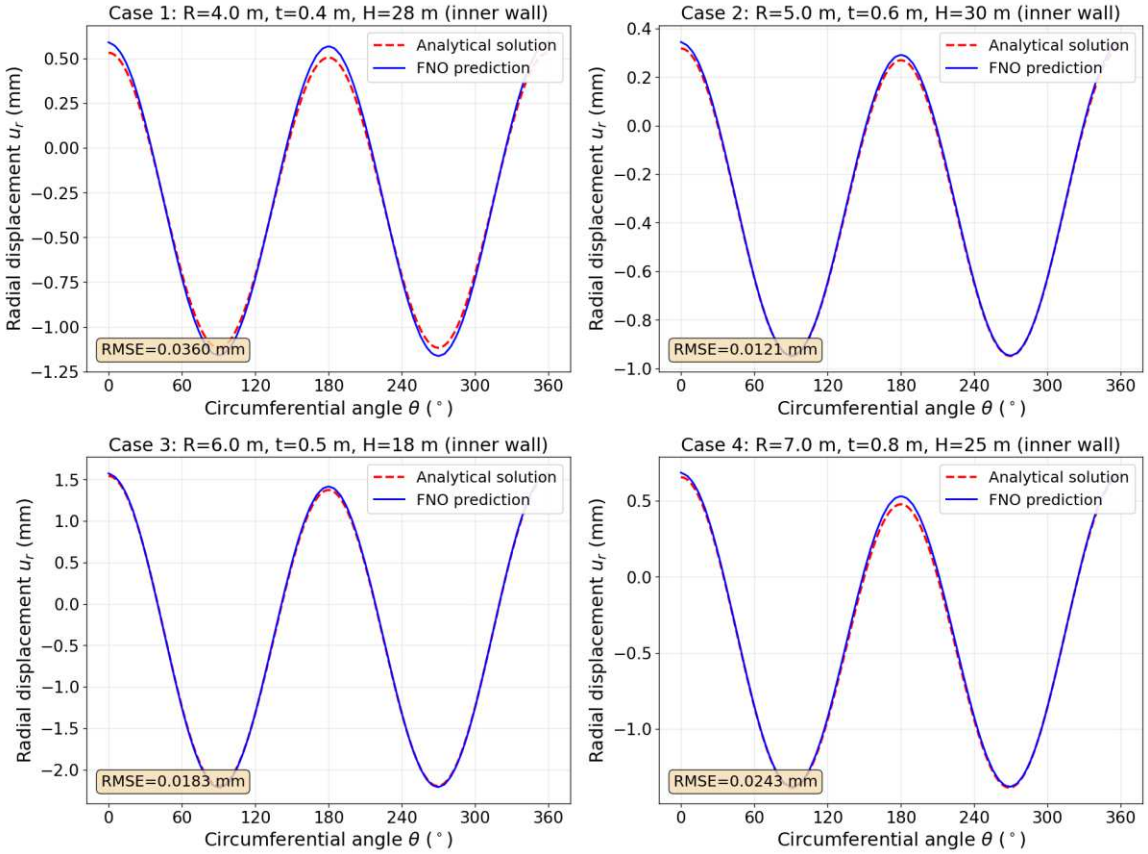
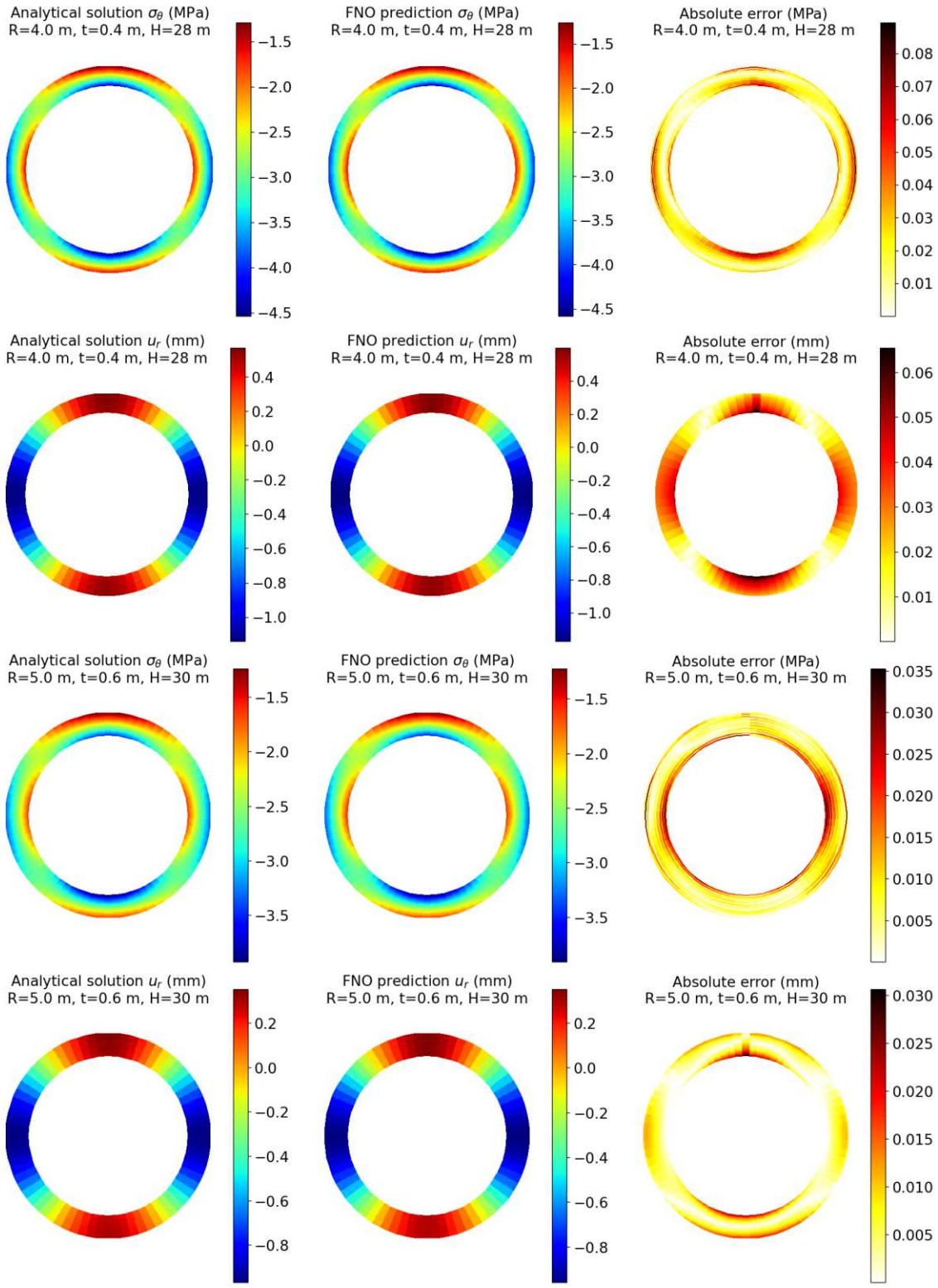


Fig. 2 Inner-Wall Radial Displacement

Figure 2 presents the distribution curves of radial displacement u_r along the segment circumference for four cases, comparing FNO predictions with analytical solutions.

The distribution pattern of radial displacement corresponds to that of hoop stress, with larger displacement at the pipe crown and invert (converging inward) and smaller displacement at the pipe sides. The displacement curves predicted by FNO are in high agreement with the analytical solutions, and the displacement amplitude and phase of each case are accurately reproduced. Since the displacement field itself is smoother than the stress field (stress is the derivative of displacement), FNO's prediction accuracy for displacement is comparable to that for stress. The full-circumference inner-wall displacement RMSE for each case is 0.0360, 0.0121, 0.0183, and 0.0243 mm, respectively, all within 0.04 mm.

6.6 Full-Wall Thickness Stress and Displacement Ring Contour Plots



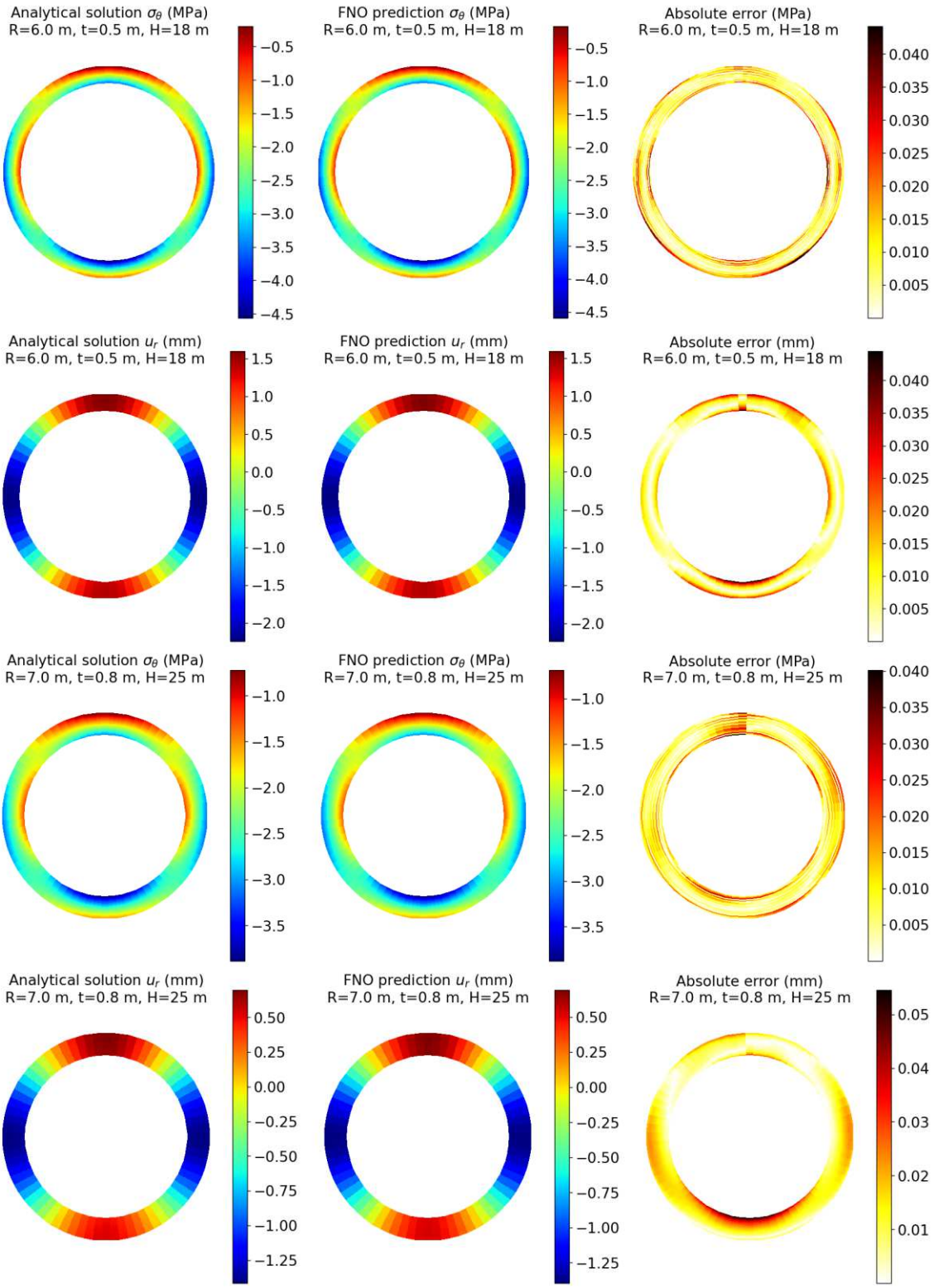


Fig. 3 Contour Plots of Hoop Stress and Radial Displacement across Full Wall Thickness

Figure 3 presents the full-wall-thickness ring contour plots of hoop stress and radial displacement for four cases, with each field quantity containing three subplots: analytical solution, FNO prediction, and absolute error.

The ring contour plots cover the entire wall thickness range from the inner wall to the outer wall, with colors indicating the magnitude of stress/displacement. The following can be intuitively observed from the contour plots: (1) hoop stress varies continuously throughout the wall thickness, with the absolute value of inner-wall stress greater than that of the outer wall, consistent with the basic law of the Lame solution for thick-walled cylinders; (2) the circumferential distribution of stress and displacement shows obvious non-uniform characteristics, with the pipe crown and invert as control sections; (3) the contour plots predicted by FNO are consistent with the analytical solution contour plots in spatial distribution morphology, with smooth color transitions and no obvious numerical artifacts; (4) the absolute error contour plots show that errors are mainly concentrated in the pipe crown and invert regions with large stress gradients, but the overall error magnitude is much smaller than the magnitude of stress itself (maximum absolute stress error is approximately 0.08 MPa, maximum absolute displacement error is approximately 0.05 mm).

Unlike DEM-NN, which only outputs inner-wall stress curves, and PINN, which also mainly uses curve comparison, FNO can directly output full-field contour plots, providing designers with more intuitive visualization of the stress state.

6.7 Parameter Sensitivity Analysis

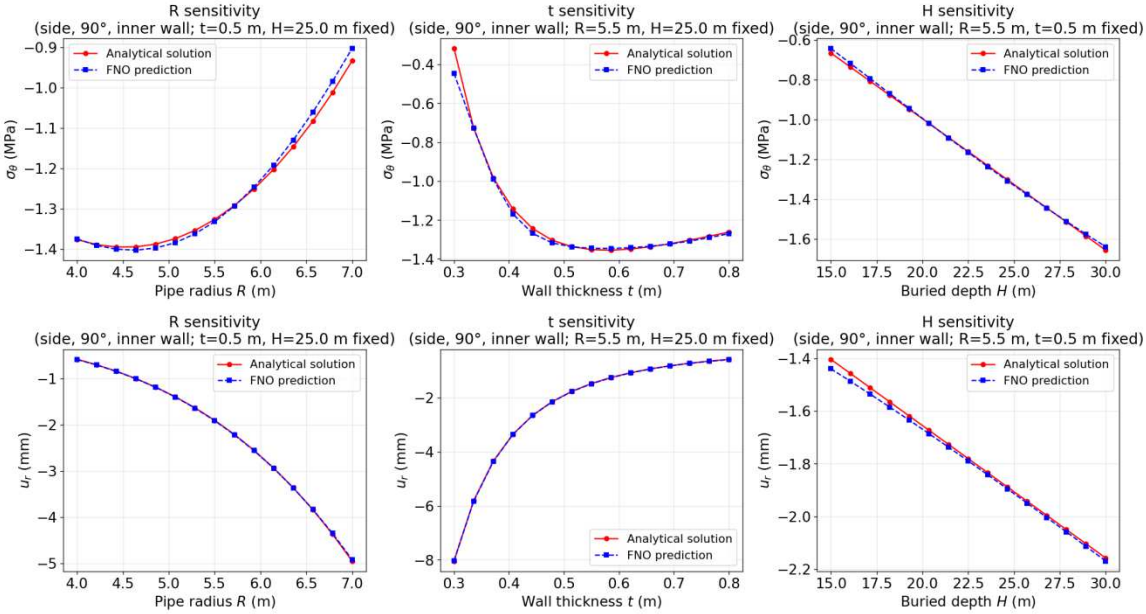


Fig. 4 Sensitivity Curves of Hoop Stress and Radial Displacement at Inner-Wall Pipe Side

Figure 4 presents the sensitivity analysis curves of hoop stress and radial displacement at the pipe side (90°) position as key parameters vary, comparing FNO prediction results with analytical solutions. The baseline parameters are $R = 5.5$ m, $t = 0.5$ m, $H = 25$ m.

The sensitivity analysis selects three main design parameters—segment radius R, wall thickness t, and burial depth H—and examines the response patterns of pipe-side hoop stress and radial displacement when each varies independently. As can be seen from the figure: (1) the sensitivity curves predicted by FNO almost completely overlap with the analytical solutions, correctly reflecting the direction and magnitude of the influence of each parameter on stress and displacement; (2) wall thickness has the most significant influence on stress, and increasing wall thickness can effectively reduce hoop stress; (3) increasing burial depth leads to increased loading, and both the absolute value of stress and displacement increase accordingly; (4) the influence of segment radius is between the two.

The fast inference feature of FNO makes large-scale parameter sensitivity analysis efficient—traditional methods require re-solving for each parameter set, while FNO can complete inference for

any parameter combination in milliseconds after training, facilitating sensitivity scanning and reliability analysis over the full parameter space.

6.8 Prediction Error Statistics

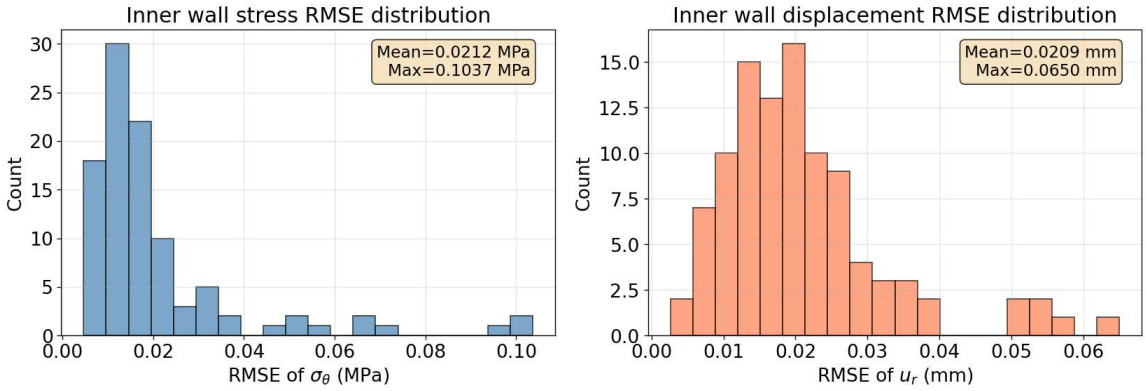


Fig. 5 Histogram of RMSE Distribution for Inner-Wall Hoop Stress and Radial Displacement

Figure 5 presents the histogram of RMSE distribution for full-circumference inner-wall hoop stress and radial displacement of 100 random test samples.

From the statistical results, the mean stress RMSE is 0.0212 MPa, with a maximum of 0.1037 MPa; the mean displacement RMSE is 0.0209 mm, with a maximum of 0.0650 mm. The error distribution is right-skewed, with most sample errors much smaller than the mean, indicating that FNO has stable high-precision prediction ability within the training parameter space. As a data-driven method, the accuracy of FNO is constrained by the quality and coverage of training data. It can achieve high accuracy within the training parameter space, but errors may increase when extrapolating to parameter combinations far from the training distribution.

7 Comparison and Summary of Three AI Methods

7.1 Systematic Comparison of DEM-NN, PINN, and FNO

Deep Energy Method Neural Network (DEM-NN), Physics-Informed Neural Network (PINN), and Fourier Neural Operator (FNO) respectively represent three technical routes for artificial intelligence to solve the mechanics problems of large-diameter pipe jacking segments. Their systematic comparison is shown in Table 3.

Table 3 Systematic Comparison of Three AI Methods

Comparison Dimension	DEM-NN	PINN	FNO
Physical Basis	Minimum total potential energy principle	Equilibrium equation + constitutive relation	Data-driven (analytical solution)
Network Output	Displacement (stress by differentiation)	Mixed displacement + stress output	Stress/displacement field
σ_θ Inner-Wall Accuracy	Trend correct, high-frequency spurious oscillations, max err≈2.3 MPa	Smooth curve, max err≈1.71 MPa	RMSE≈0.02 MPa
σ_r Outer-Wall Accuracy	max err≈1.25 MPa	max err≈9.3 kPa	Consistent across full field
Cost for New Case	Retraining (200 steps L-BFGS)	Retraining (8000 steps Adam)	Forward propagation (millisecond-level)

Running Variability	Yes	Yes	No
Training Data Requirement	None	None	Required (analytical/numerical solution)
Full-Field Output	Requires post-processing	Requires post-processing	Direct output
Applicable Scenarios	Complex constitutive, no analytical solution	Known PDE, high-precision single case	Parametric design, rapid comparison

The advantage of DEM-NN is that it does not require writing the strong form of the governing equations, only constructing the energy functional, and has potential advantages for complex constitutive problems; its limitation lies in the zero-energy mode problem under pure Neumann boundaries and high-frequency spurious stress oscillations. The advantage of PINN is that it can directly utilize the known governing equation structure, with flexible boundary condition application, and the mixed variable method effectively overcomes the stress degradation problem of the pure displacement method; its limitation is likewise that single cases require retraining, and the inner-wall stress accuracy is limited due to the lack of direct boundary constraints. The advantage of FNO lies in operator-level fast inference and full-field output, which can efficiently support parametric design; its limitation is that it requires training data, and extrapolation accuracy is constrained by the coverage of the training distribution.

7.2 Summary

This paper introduces the Fourier Neural Operator (FNO) into the rapid prediction of hoop stress and radial displacement of large-diameter circular pipe jacking segments, establishing an operator learning framework from design parameters to full-field solutions. Based on the analytical solution in Chapter 2, 1500 training samples were generated in batches, and two independent FNO models for stress prediction and displacement prediction were constructed respectively. Systematic verification was carried out through four dimensions: circumferential distribution curves, full-wall-thickness ring contour plots, parameter sensitivity analysis, and error statistics.

The results show that the hoop stress and radial displacement predicted by FNO agree well with the analytical solutions in terms of circumferential distribution, full-wall-thickness contour plots, and parameter sensitivity. The mean RMSE of inner-wall full-circumference stress over 100 random test samples is 0.0212 MPa (max 0.1037 MPa), and the mean RMSE of displacement is 0.0209 mm (max 0.0650 mm), without the high-frequency spurious oscillations in DEM-NN; after training is completed, inference for new cases only requires one forward propagation, improving by several orders of magnitude compared to the single-case retraining methods of DEM-NN and PINN. FNO provides an efficient tool for parametric design and rapid multi-scheme comparison of pipe jacking segments.

FNO is complementary to DEM-NN and PINN: physics-driven methods are suitable for high-precision single-case solving in data-free scenarios, while data-driven FNO is suitable for fast multi-case inference in scenarios with data. Future work can be further expanded in the following directions: (1) combining PINO [13] to embed physical constraints into the FNO training process to improve generalization ability in data-scarce scenarios; (2) extending to heterogeneous strata and prefabricated joint effects to make FNO applicable to more complex engineering conditions; (3) utilizing the differentiability of FNO to conduct research on segment wall thickness optimization and parameter inversion.

8 Conclusions

Aiming at the rapid prediction problem of hoop stress and radial displacement of large-diameter prefabricated circular pipe jacking segments, this paper introduces the Fourier Neural Operator (FNO)

method and establishes an operator learning framework from design parameters to full-field solutions. The main conclusions are as follows:

(1) FNO can accurately learn the solution operator from design parameters (segment radius R, wall thickness t, burial depth H) to full-field hoop stress and radial displacement. The prediction results are in high agreement with the analytical solutions in terms of circumferential distribution, full-wall-thickness contour plots, and parameter sensitivity. The mean RMSE of inner-wall full-circumference stress is 0.0212 MPa, and the mean RMSE of displacement is 0.0209 mm, without the high-frequency spurious oscillations in the deep energy method.

(2) FNO possesses the efficiency advantage of "train once, millisecond-level inference for multiple cases". A new case only requires one forward propagation to obtain the full-field solution, improving by several orders of magnitude compared to the single-case retraining methods of DEM-NN and PINN, and can efficiently support parametric design and rapid multi-scheme comparison.

(3) FNO is complementary to DEM-NN and PINN: physics-driven methods are suitable for high-precision single-case solving in data-free scenarios, while data-driven FNO is suitable for fast multi-case inference in scenarios with data. Future work can combine the Physics-Informed Neural Operator (PINO) to embed physical constraints into the FNO training process, improve generalization ability in data-scarce scenarios, and extend to more complex engineering conditions such as heterogeneous strata and prefabricated joint effects.

Funding

The authors declare that no funds, grants, or other financial support were received for this study.

Ethics, Consent to Participate, and Consent to Publish

Ethics, Consent to Participate, and Consent to Publish declarations: not applicable.

Conflict of Interest

The authors declare that they have no conflict of interest.

Availability of Data and Materials

The data used to support the findings of this study are available from the corresponding author upon reasonable request.

References:

[1]. Zeng Jianjun, et al. Research status and development trend of pipe jacking tunnel. Modern Tunnelling Technology, 2024. 61(S1): 1-16.

[2]. Yu Yingxia, et al. Influence of super-large diameter double-round pipe jacking construction on settlement deformation of underground pipelines. Journal of Yangtze River Scientific Research Institute, 2026. 43(02): 130-139+147.

[3].Tang, L., Z. Zhang and J. Wei, Evaluation of Deformation Mechanisms and Application Effects of Pipe Roofing in Ultra-Shallow Buried Large-Section Rectangular Pipe Jacking Projects. International Journal of Civil Engineering, 2026. 24(4-5): p. 1-18.

[4].Geng, Z., et al., Study on ground deformation and settlement caused by the construction of pipe jacking tunnel in multi-layer strata. Tunnelling and Underground Space Technology incorporating Trenchless Technology Research, 2025. 165: p. 106856-106856.

[5]. Zhang Zhongjie, et al. Stress analysis and field measurement of deeply buried large-section quasi-rectangular pipe jacking structure in soft soil. Chinese Journal of Geotechnical Engineering, 2025. 47(06): 1190-1198.

[6].Li, Z., et al., Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895, 2020.

[7].Kovachki, N., et al., Neural operator: Learning maps between function spaces with applications to pdes. Journal of Machine Learning Research, 2023. 24(89): p. 1-97.

[8].Azizzadenesheli, K., et al., Neural operators for accelerating scientific simulations and design. Nature Reviews Physics, 2024. 6(5): p. 320-328.

[9].Chen, T. and H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks, 1995. 6(4): p. 911-917.

[10].Lu, L., et al., Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature machine intelligence, 2021. 3(3): p. 218-229.

[11].Wen, G., et al., U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow.

Advances in Water Resources, 2022. 163: p. 104180-104180.
[12].Li, Z., et al., Fourier neural operator with learned deformations for pdes on general geometries. Journal of Machine Learning Research, 2023. 24(388): p. 1-26.
[13].Li, Z., et al., Physics-informed neural operator for learning partial differential equations. ACM/IMS Journal of Data Science, 2024. 1(3): p. 1-27.
[14].Pathak, J., et al., Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. arXiv preprint arXiv:2202.11214, 2022.
[15].Rashid, M.M., et al., Learning the stress-strain fields in digital composites using Fourier neural operator. Iscience, 2022. 25(11).
[16].Kaewnuratchadasorn, C., J. Wang and C.W. Kim, Neural operator for structural simulation and bridge health monitoring. Computer‐Aided Civil and Infrastructure Engineering, 2024. 39(6): p. 872-890.
[17].Kaewnuratchadasorn, C., J. Wang and C.W. Kim, Physics‐informed neural operator solver and super‐resolution for solid mechanics. Computer‐Aided Civil and Infrastructure Engineering, 2024. 39(22): p. 3435-3451.
[18].Shin, M., et al., Local Stress Fields Prediction using Global Displacement through Fourier Neural Operators. Journal of Computational Design and Engineering, 2025.
[19].Tristani, A., L.M.G. Carrillo and J. Sulem, Data‐Driven Tools to Evaluate Support Pressure, Radial Displacements, and Face Extrusion for Tunnels Excavated in Elastoplastic Grounds. International Journal for Numerical and Analytical Methods in Geomechanics, 2024.
[20].Xu, C., et al., Fusing simulation and monitoring data for real-time settlement prediction during tunnel construction: A multi-fidelity deep operator network (deeponet). arXiv preprint arXiv:2310.00057, 2023.