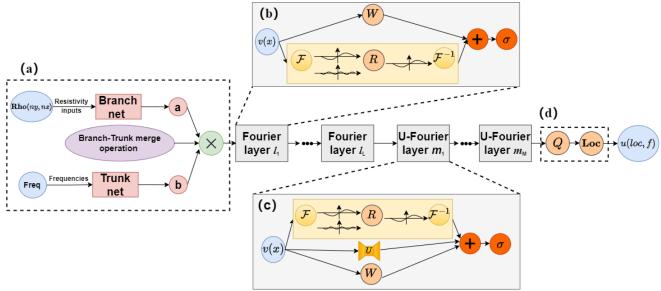
Extended Fourier DeepONet Neural Operator Network (EFDO) 🚀

If you find this work helpful for your research, please consider citing:

Citation:

```
@article{liao2025fast,
1
      title={Fast forward modeling of magnetotelluric data in complex continuous media
    using an extended Fourier DeepONet architecture},
      author={Liao, Weiyang and Peng, Ronghua and Hu, Xiangyun and Zhang, Yue and Zhou,
    Wenlong and Fu, Xiaonian and Lin, Haikun},
4
      journal={Geophysics},
5
      volume={90},
      number={2},
6
      pages={F11--F25},
8
      year = \{2025\},\
      publisher={Society of Exploration Geophysicists}
9
10
    }
```

W. Liao, R. Peng, X. Hu, Y. Zhang, W. Zhou, X. Fu, et al., GEOPHYSICS 2025 Vol. 90 Issue 2 Pages F11-F25, DOI: 10.1190/geo2023-0613.1



The innovative architecture of our Extended Fourier DeepONet (EFDO) neural operator network, combining the power of Fourier transforms with DeepONet for superior performance in geophysical modeling.

1. 6 Why EFDO?

EFDO is a cutting-edge neural operator network designed specifically for geophysical modeling tasks. It combines the efficiency of Fourier transforms with the flexibility of DeepONet architecture, offering:

- ii Better accuracy
- Les Robust performance
- Enhanced generalization

2. Hardware Requirements

Before you dive in, make sure you have:

- M NVIDIA GPU (compute capability ≥ 6.0)
- 💾 GPU Memory: 16GB minimum (24GB recommended)
- § System Memory: 64GB minimum (256GB recommended)
- S Disk Space: 50GB minimum (512GB recommended)

3. Software Requirements

You'll need these tools in your arsenal:

- **Q** Python ≥ 3.7
- | PyTorch ≥ 1.8.0
- Lessential packages:
 - o torchinfo
 - o yaml
 - o numpy
 - o scipy
 - o pandas
 - o matplotlib
 - jupyter notebook

4. X Installation

Pro tip: We recommend using Anaconda with Mamba for lightning-fast package installation!

4.1 Step 1: Get Mamba Up and Running

First, grab Mamba from the Mambaforge download page:

1 | bash Miniforge3-Linux-x86_64.sh -b -p \${HOME}/miniforge

4.2 Step 2: Set Up Your Environment

Add these magic lines to your ~/.bashrc:

```
# conda
if [ -f "${HOME}/miniforge/etc/profile.d/conda.sh" ]; then
source "${HOME}/miniforge/etc/profile.d/conda.sh"

fi

mamba
if [ -f "${HOME}/miniforge/etc/profile.d/mamba.sh" ]; then
source "${HOME}/miniforge/etc/profile.d/mamba.sh"

alias conda=mamba
```

4.3 Step 3: Create Your EFDO Environment

```
1 conda create -n efdo python=3.8
2 conda activate efdo
```

4.4 Step 4: Install Dependencies

```
# Install PyTorch with CUDA support

conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia

# Install other required packages

conda install torchinfo pyyaml numpy scipy pandas matplotlib jupyter notebook

pip install ray
```

4.5 Step 5: Get the Code

```
1 | git clone https://github.com/CUG-EMI/EFDO
```

5. Dataset Generation

5.1 Option 1: Python Method

Quick and parallel dataset generation:

```
1 | python model_gen.py 100 50 train_gen
```

- 100: Number of datasets
- 50: Parallel processes
- train_gen: Output filename

5.2 Option 2: Julia Method

```
1 \mid \mathsf{julia} juliaCallGRF.jl
```

In this command, the generated datasets will be saved in the data directory. But it is worth noting that the julia code only generates the Gaussian random field models, and you need to calculate the forward modeling results using other forward modeling codes. In this research, we use our own MT2D julia forward modeling code to calculate the forward modeling results. And the forward modeling code

is not open source.

Note: For Julia users, set up PyCall first:

```
1 ENV["PYTHON"] = "Path of the python environment"
2 using Pkg
3 Pkg.add("PyCall")
4 Pkg.build("PyCall")
```

5.3 **A** Important Note on Forward Modeling

The Python forward modeling code included in this repository (which is not our original work, for more details see <u>EFNO_GRF</u>) has some limitations and potential issues. Therefore, we recommend:

- 1. Use only the Gaussian random field part of this code to generate the conductivity models
- 2. Employ other well-established forward modeling codes to calculate the MT responses
- 3. This approach ensures more reliable and accurate dataset generation

This separation of model generation and forward computation allows for:

- Better quality control of the synthetic data
- More flexibility in choosing appropriate forward modeling algorithms
- Increased reliability of the training dataset

If you need recommendations for alternative forward modeling codes, please refer to established MT modeling software in the geophysical community.

5.4 **!** Pre-generated Datasets

Don't want to generate data? No problem! Download our pre-generated datasets from <u>Google Drive</u> <u>datasets</u>

6. Training Your Network

6.1 Configuration

- 1. Navigate to the code directory
- 2. Edit config_EFDO.yaml with your paths and parameters

6.2 Training Commands

```
# change to the code directory
cd code
# activate environment
conda activat efdo
# Train EFDO
python EFDO_main.py EFDO_config

# Train EFNO
python EFNO_main.py EFNO_config

# Train UFNO3d
python UFNO3d_main.py UFNO3d_config
```

6.3 🔬 Visualization and Analysis

Check out our Jupyter notebooks (*.ipynb) in the code directory for:

- Result visualization
- Performance analysis
- Model comparisons

6.4 **Pre-trained Models**

Get a head start with our pre-trained models from <u>Google Drive pre-trained models</u>, you can download them and put them in the <u>temp_model</u> directory, and then you can use the trained models in the <u>jupyter</u> notebook file to predict the forward responses.

Ready to revolutionize your geophysical modeling? Let's get started! 🚀

For questions and support, open an issue in our GitHub repository or contact our team.

EFDO Neural Network Visualization Guide

1. 🌞 Introduction

Welcome to the visualization guide for the EFDO Neural Network! To help researchers in geophysics better understand and utilize our open-source neural operator network, we're excited to share all the plotting scripts used in our paper.

1.1 | Data Visualization Tools

All figures in this study were created using Generic Mapping Tools (GMT). To promote broader adoption and understanding of GMT within the research community, we provide scripts in three different formats:

- **Traditional GMT Scripts**: For command-line interface enthusiasts
- **Q PyGMT Scripts**: For Python users seeking a more familiar syntax
- 🐈 Julia GMT Scripts: For Julia users preferring GMT functionality in Julia

Pro Tip: Choose the format that matches your programming style - they all produce identical, publication-quality results!

2. **©** Example: Reproducing Paper Figures

Let's take a look at one of our key figures and how you can reproduce it using any of the three supported approaches:

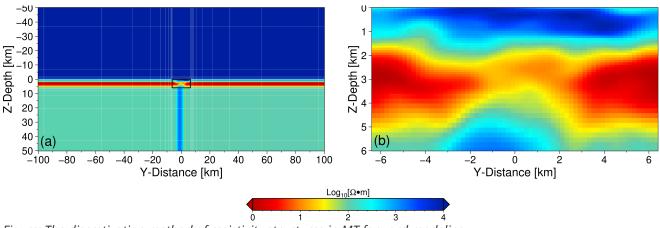


Figure: The discretization method of resistivity structures in MT forward modeling.

2.1 **%** Three Implementation Approaches

GMT Script

```
#!/usr/bin/env bash
 2
    function preset1()
 3
        ymin = -100000
 4
 5
        ymax=100000
        zmin = -50000
 6
 7
        zmax=50000
8
        scale=1000
        ymin=`echo $ymin $scale | awk '{print ($1/$2)}'`
9
10
        ymax=`echo $ymax $scale | awk '{print ($1/$2)}'`
11
        zmin=`echo $zmin $scale | awk '{print ($1/$2)}'`
        zmax=`echo $zmax $scale | awk '{print ($1/$2)}'`
12
13
        echo $ymin $ymax $zmin $zmax
14
        range_yz1=$ymin/$ymax/$zmin/$zmax
15
    }
16
    preset1
17
18
    gmt set FONT_ANNOT_PRIMARY 12p
    gmt set FONT_LABEL 14p
19
20
    gmt set MAP_FRAME_PEN 1p,black
21
22
    # x=0 slice
23
    gmt begin Figure5 pdf,png
    gmt makecpt -Cthermal.cpt -T0/4.0/0.01 -Z -H > rbow.cpt
24
25
    gmt grdconvert wholeRangeGrids.grd out.grd
26
    gmt grdedit out.grd -R$range_yz1 -Gresult.nc
27
    gmt basemap -Bxa20f10+l"Y-Distance [km]" -Bya10f5+l"Z-Depth [km]" -BwStr -R$range_yz1
28
    -JX12c/-6c
29
    gmt grdimage -R$range_yz1 -JX12c/-6c result.nc -Crbow.cpt
30
31
    # y grid lines
32
    gmt plot -Wfaint,white line_y.txt -Frs
33
34
    # z grid lines
    gmt plot -Wfaint,white line_z.txt -Frs
35
36
37
    # the core region
```

```
38 gmt plot -w0.8p <<EOF
39
    -6.40
    -6.4 6.0
40
41 6.4 6.0
    6.4 0
43
   -6.4 0
44
    EOF
45
    gmt text -F+f16p, Helvetica << EOF
46
47
    -93 43 (a)
48
    EOF
49
    function preset2()
50
51
52
        ymin=-6400
53
        ymax=6400
        zmin=0
54
55
        zmax=6000
56
        scale=1000
57
        ymin=`echo $ymin $scale | awk '{print ($1/$2)}'`
58
        ymax=`echo $ymax $scale | awk '{print ($1/$2)}'`
59
        zmin=`echo $zmin $scale | awk '{print ($1/$2)}'`
60
        zmax=`echo $zmax $scale | awk '{print ($1/$2)}'`
        echo $ymin $ymax $zmin $zmax
61
62
        range_yz2=$ymin/$ymax/$zmin/$zmax
63
    }
64
    preset2
65
    gmt grdconvert coreRangeGrids.grd out.grd
    gmt grdedit out.grd -R$range_yz2 -Gresult.nc
    gmt basemap -Bxa2f1+|"Y-Distance [km]" -Bya1f1+|"Z-Depth [km]" -BWStr -R$range_yz2 -
    JX12c/-6c -X14
68
    gmt grdimage -R$range_yz2 -JX12c/-6c result.nc -Crbow.cpt
69
70
    gmt text -F+f16p,Helvetica << EOF
71
    -5.95 5.55 (b)
72
    EOF
73
74
    gmt colorbar -R$range_yz1 -Crbow.cpt -DjTC+w8c/0.5c+o-7.0c/8.0c+m1+e -N -
    Bxaf+l"Log@-10@-[@\sim W\267@\sim m]" -By -FONT_ANNOT_PRIMARY=15p
75
76
   rm out.grd rbow.cpt *.nc
77
78
    gmt end
```

Q PyGMT Script

```
import pygmt

def preset1():
    # Convert range units to kilometers
    ymin, ymax, zmin, zmax = -100, 100, -50, 50 # Assuming these values are already in kilometers
    return ymin, ymax, zmin, zmax

def preset2():
```

```
9
       # Convert range units to kilometers
10
        ymin, ymax, zmin, zmax = -6.4, 6.4, 0, 6 # Again, assuming these values are
    already in kilometers
        return ymin, ymax, zmin, zmax
11
12
    # Set file and directory paths
13
14
    dir_path = "./"
15
    cpt_file_in = f"{dir_path}thermal.cpt"
    cpt_file_out = f"{dir_path}rbow.cpt"
16
17
    result1_grid = f"{dir_path}result1.grd"
18
    result2_grid = f"{dir_path}result2.grd"
    out_pdf_png = f"{dir_path}Figure5_py"
19
20
21
    # Set range by calling functions
22
    ymin, ymax, zmin, zmax = preset1()
    range_yz1 = f''{ymin}/{ymax}/{zmin}/{zmax}"
23
24
25
    pygmt.config(FONT_ANNOT_PRIMARY="12p", FONT_LABEL="14p", MAP_FRAME_PEN="1p,black")
26
    fig = pygmt.Figure()
27
28
    pygmt.makecpt(cmap=cpt_file_in, series="0/4.0/0.01", continuous=True,
    output=cpt_file_out)
29
    # Convert grid range using grdedit in original GMT, currently, PyGMT does not support
30
    grdedit and grdconvert modules
31
    # gmt grdedit wholeRangeGrids.grd -R-100/100/-50/50 -Gresult1.grd
32
33
    fig.grdimage(grid=result1_grid, region=range_yz1, projection="X12c/-6c",
    cmap=cpt_file_out, frame=["xa20f10+lY-Distance [km]", "ya10f10+lDepth [km]", "WStr"])
34
    fig.plot(data=f"{dir_path}line_y.txt", pen="faint,white")
    fig.plot(data=f"{dir_path}line_z.txt", pen="faint,white")
35
36
    fig.plot(data=[[-6.4, 0], [-6.4, 6.0], [6.4, 6.0], [6.4, 0], [-6.4, 0]], pen="0.8p")
    fig.text(x=-93, y=43, text="(a)", font="16p,Helvetica")
37
38
39
    ymin, ymax, zmin, zmax = preset2()
40
    range_yz2 = f''\{ymin\}/\{ymax\}/\{zmin\}/\{zmax\}''
41
    # Convert grid range using grdedit in original GMT, currently, PyGMT does not support
42
    grdedit and grdconvert modules
43
    # gmt grdedit coreRangeGrids.grd -R-6.4/6.4/0/6 -Gresult2.grd
44
    fig.shift_origin(xshift="14c")
45
46
    fig.grdimage(grid=result2_grid, region=range_yz2, projection="X12c/-6c",
    cmap=cpt_file_out, frame=["xa2f1+|Y-Distance [km]", "ya1f1+|Depth [km]", "WStr"])
47
    fig.text(x=-5.95, y=5.55, text="(b)", font="16p,Helvetica")
48
49
    pygmt.config(FONT_ANNOT_PRIMARY="14p", FONT_LABEL="14p")
50
    fig.colorbar(cmap=cpt_file_out, frame=["xaf+lLog@-10@-[@~w\267@~m]", "y"],
    position="jTC+w8c/0.5c+o-7.0c/8.0c+m1+e")
51
52
    # Save in PDF and PNG formats
53
    fig.savefig(f"{out_pdf_png}.pdf")
    fig.savefig(f"{out_pdf_png}.png")
54
55
```

• 🐈 Julia GMT Script

```
using GMT
1
2
 3
    function preset1()
4
        ymin = -100000
        ymax = 100000
 6
        zmin = -50000
        zmax = 50000
8
        scale = 1000
9
        ymin /= scale
10
        ymax /= scale
11
        zmin /= scale
        zmax /= scale
12
13
        println("$ymin $ymax $zmin $zmax")
14
        range_yz1 = string(ymin, "/", ymax, "/", zmin, "/", zmax)
15
        return range_yz1
    end
16
17
18
    function preset2()
19
        ymin = -6400
20
        ymax = 6400
21
        zmin = 0
22
        zmax = 6000
23
        scale = 1000
24
        ymin /= scale
25
        ymax /= scale
26
        zmin /= scale
        zmax /= scale
27
28
        println("$ymin $ymax $zmin $zmax")
29
        range_yz2 = string(ymin, "/", ymax, "/", zmin, "/", zmax)
30
        return range_yz2
31
    end
32
33
    range_yz1 = preset1()
34
    range_yz2 = preset2()
35
36
    gmtbegin("Figure5_jl", fmt="pdf,png")
37
        # Create a color palette
38
        C = makecpt("-Cthermal.cpt -T0/4/0.1 -Z > rbow.cpt")
39
40
41
        # Figure (a)
        data1 = grdedit("wholeRangeGrids.grd", region=range_yz1)
42
        basemap!(region=range_yz1, figsize=(12, -6), frame=(axes=:WStr,),
43
44
                  xaxis=(annot=20, ticks=10, label=:"Y-Distance [km]"),
                  yaxis=(annot=10, ticks=5, label=:"Z-Depth [km]"),
45
                  par=(FONT_ANNOT_PRIMARY=12, FONT_LABEL=14, MAP_FRAME_PEN="1p,black",)
46
47
48
        grdimage!(data1, region=range_yz1, figsize=(12, -6), cmap="rbow.cpt")
        plot!("line_y.txt", pen=(0.1, :White))
49
        plot!("line_z.txt", pen=(0.1, :White))
50
51
        boxline = [-6.4 \ 0; -6.4 \ 6.0; 6.4 \ 6.0; 6.4 \ 0; -6.4 \ 0]
52
        plot!(boxline, pen="0.8p", close=true)
        text!(["(a)"], font=16, x=-93, y=43)
53
```

```
54
55
        # Figure (b)
        data2 = grdedit("coreRangeGrids.grd", region=range_yz2)
56
57
        basemap!(region=range_yz2, figsize=(12, -6), frame=(axes=:WStr,),
                 xaxis=(annot=2, ticks=1, label=:"Y-Distance [km]"),
58
59
                 yaxis=(annot=1, ticks=1, label=:"Z-Depth [km]"),
                 par=(FONT_ANNOT_PRIMARY=12, FONT_LABEL=14,MAP_FRAME_PEN="1p,black",),
60
                 xshift=14
61
63
        grdimage!(data2, region=range_yz2, figsize=(12, -6), cmap="rbow.cpt")
64
        text!(["(b)"], font=16, x=-5.95, y=5.55)
65
        # Adding the colorbar
66
67
        gmtset(FONT_ANNOT_PRIMARY = "14p,Helvetica,black", FONT_LABEL = "14p,black")
        colorbar!(xaxis=(annot=1, ticks=0.2, label=:"Log@-10@-[@~w\267@~m]"),
                  pos=(paper=true, anchor=(-5,-2.5), size=(8,0.5), horizontal=true,
69
                  move_annot=:1, triangles=true))
70
71
    gmtend()
```

As you can see, each approach achieves the same result while catering to different programming preferences and workflows.

3. Setup and Installation

3.1 Installation Guide

3.1.1 1 GMT and PyGMT Setup

We recommend using conda to create dedicated environments:

```
# Setting up GMT
conda create -n gmt python=3.12
conda activate gmt
conda install -c conda-forge gmt
# if `gmt --version` displays some error, please try the following way
conda remove gdal sqlite
conda install -c conda-forge gmt gdal sqlite
conda update gmt

# Setting up PyGMT
conda create -n pygmt python=3.12
conda activate pygmt
conda install numpy scipy pandas xarray netcdf4 packaging pygmt
```

3.1.2 2 Julia GMT Setup

For Julia users, it's just one line:

```
1 using Pkg
2 Pkg.add("GMT") # That's all!
```

Recommendation: We suggest using the latest version of GMT in WSL2 with Ubuntu for the best experience!

4. M Usage Guide

4.1 II Using the Plotting Scripts

All figure scripts are located in the Plotting_scripts directory:

```
cd Plotting_scripts/Figure5

# For GMT users
conda activate gmt
bash Figure5.sh

# For Python users
conda activate pygmt
python Figure5.py

# For Julia users
julia Figure5.jl
```

4.2 **I** Jupyter Notebook Support

We also provide Geophysics_plotting_scripts.ipynb which includes:

- Q Detailed step-by-step explanations
- | Interactive environment

5. Getting Help

- **Usit the GMT official documentation**
- **%** Submit issues on GitHub