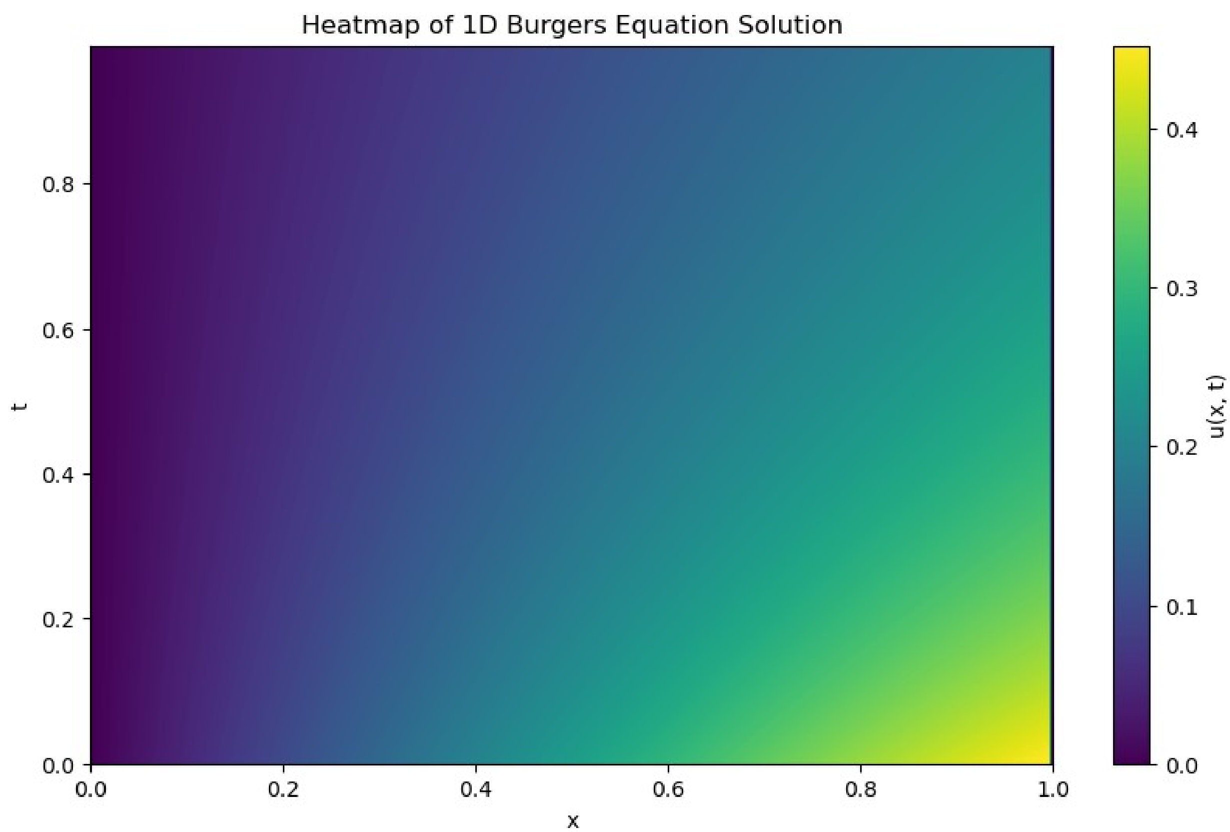
```python
import deepxde as dde
import numpy as np
from deepxde.backend import tf
import matplotlib.pyplot as plt
```

Analytical Solution

```python
# Define the Reynolds number for the analytical solution
# Re = 1,50,100,300
Re_fixed = 1

# Define the spatial and temporal grid
N_t = 100  # Number of time points
N_x = 256  # Number of spatial points
t = np.linspace(0, 0.99, N_t)  # Time grid
x = np.linspace(0, 1, N_x)    # Spatial grid

# Define the analytical solution for the 1D Burgers equation
def analytical_solution(x, t, Re_fixed):
    to = np.exp(Re_fixed / 8)  # Parameter in the equation
    u = x / (t + 1) / (1 + np.sqrt((t + 1) / to) * np.exp(Re_fixed *
x**2 / (4 * (t + 1))))
    return u

# Compute the solution
usol = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x]
for ti in t])

# Explicitly enforce boundary conditions
usol[:, 0] = 0  # u(0, t) = 0
usol[:, -1] = 0  # u(1, t) = 0

# Verify the boundary conditions
assert np.allclose(usol[:, 0], 0), "Boundary condition u(0, t) = 0 not
satisfied"
assert np.allclose(usol[:, -1], 0), "Boundary condition u(1, t) = 0
not satisfied"

# Save the data to a .npz file
np.savez("dataset/Burgers.npz", t=t, x=x, usol=usol)

# Create a new figure window with a size of 10x6 inches
plt.figure(figsize=(10, 6))

# Plot the heatmap
# `usol` is the 2D array containing the solution, `extent` specifies
the range for x and t axes,
# `origin='lower'` ensures the heatmap starts from the bottom-left
corner,
# `aspect='auto'` adjusts the aspect ratio automatically, and
```

```
`cmap='viridis'` sets the colormap
plt.imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],
           origin='lower', aspect='auto', cmap='viridis')

# Add a colorbar to indicate the value of u(x, t) for each color
plt.colorbar(label="u(x, t)")

plt.title("Heatmap of 1D Burgers Equation Solution")
plt.xlabel("x")
plt.ylabel("t")

plt.show()
```



Part (a): Forward Problem

```
# Define fixed Reynolds number
# Re = 1,50,100,300
Re_fixed = 1

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
```

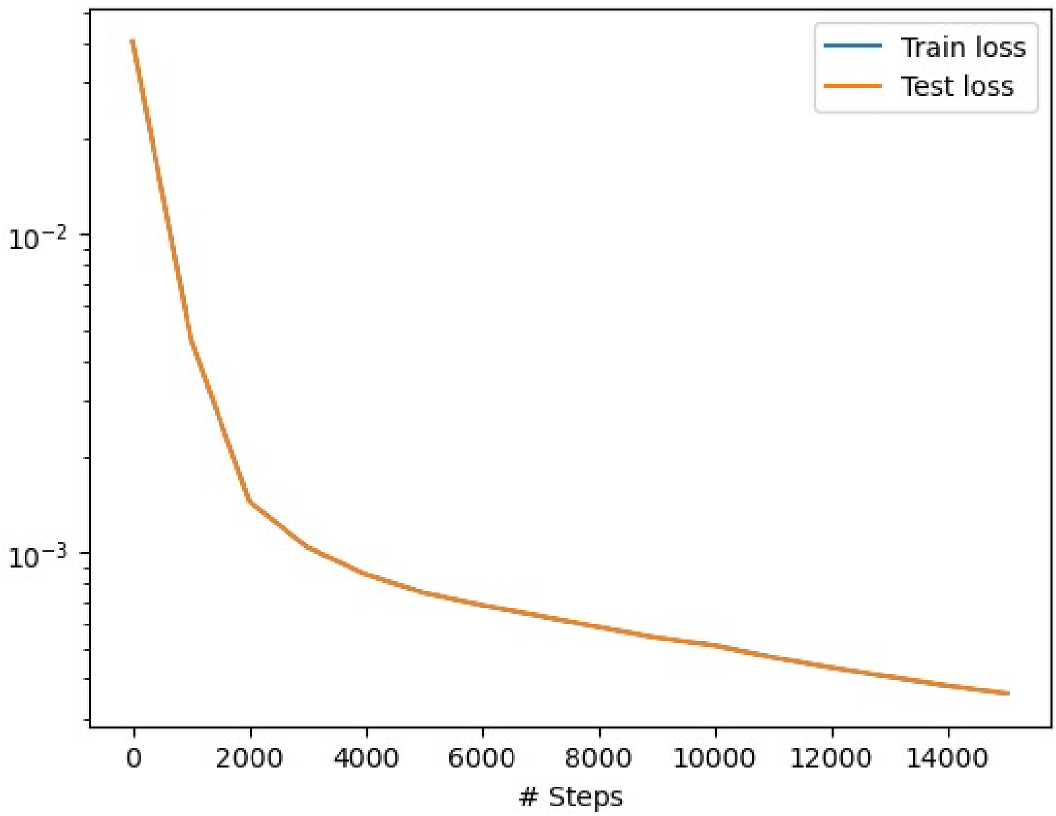
```
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde(x, y, Re):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 0.01 / Re * dy_xx # Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)
ic = dde.IC(
    geomtime, lambda x: x[:, 0:1] / (1 + np.sqrt(np.exp(Re_fixed / 8))
* np.exp(Re_fixed * x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial
)

# Solve the forward problem for fixed Re
print(f"Training for fixed Re = {Re_fixed}")

# Define dataset for fixed Re
data_fixed = dde.data.TimePDE(
    geomtime, lambda x, y: pde(x, y, Re_fixed), [bc, ic],
num_domain=2540, num_boundary=80, num_initial=160
)

# Define the neural network
net_fixed = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")
model_fixed = dde.Model(data_fixed, net_fixed)
model_fixed.compile("adam", lr=1e-3)

# Train the model
model_fixed.train(epochs=15000)
model_fixed.compile("L-BFGS")
losshistory_fixed, train_state_fixed = model_fixed.train()

# Save results
dde.saveplot(losshistory_fixed, train_state_fixed, issave=True,
isplot=True)

# Test the model
X_fixed, y_true_fixed = gen_testdata()
```

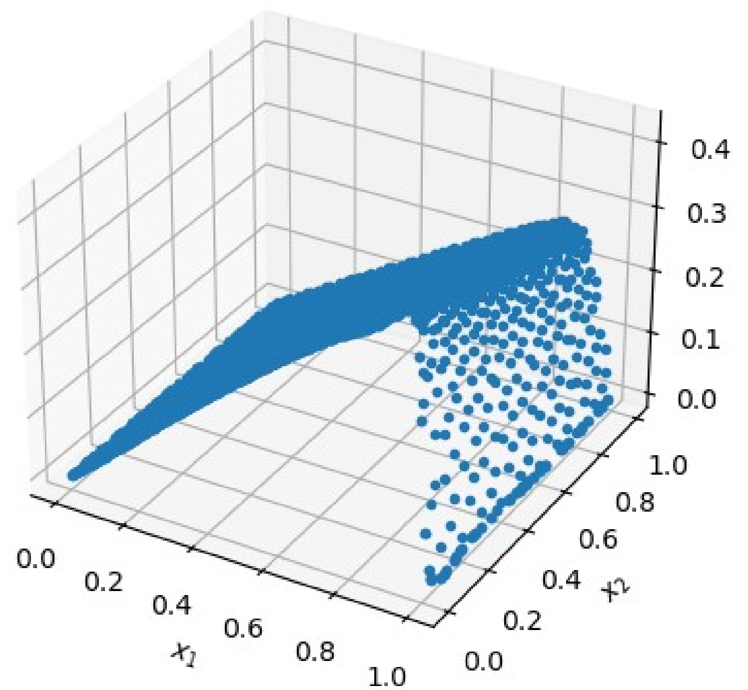
```python
y_pred_fixed = model_fixed.predict(X_fixed)
f_fixed = model_fixed.predict(X_fixed, operator=lambda x, y: pde(x, y,
Re_fixed))

# Print errors for fixed Re
print(f"Fixed Re = {Re_fixed}, Mean residual:
{np.mean(np.absolute(f_fixed))}")
print(f"Fixed Re = {Re_fixed}, L2 relative error:
{dde.metrics.l2_relative_error(y_true_fixed, y_pred_fixed)}")
np.savetxt(f"test_fixed_Re_{Re_fixed}.dat", np.hstack((X_fixed,
y_true_fixed, y_pred_fixed)))
```

```
Training for fixed Re = 1
Compiling model...
Building feed-forward neural network...
'build' took 0.077185 s

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
model.py:168: The name tf.train.Saver is deprecated. Please use
tf.compat.v1.train.Saver instead.

'compile' took 0.645177 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [8.22e-03, 1.62e-02, 1.61e-02]    [8.22e-03, 1.62e-02,
1.61e-02]    []
1000      [1.07e-03, 4.66e-04, 3.17e-03]    [1.07e-03, 4.66e-04,
3.17e-03]    []
2000      [9.17e-05, 2.28e-04, 1.13e-03]    [9.17e-05, 2.28e-04,
1.13e-03]    []
3000      [4.33e-05, 2.21e-04, 7.74e-04]    [4.33e-05, 2.21e-04,
7.74e-04]    []
4000      [3.73e-05, 1.98e-04, 6.19e-04]    [3.73e-05, 1.98e-04,
6.19e-04]    []
5000      [4.19e-05, 1.79e-04, 5.26e-04]    [4.19e-05, 1.79e-04,
5.26e-04]    []
6000      [5.14e-05, 1.61e-04, 4.70e-04]    [5.14e-05, 1.61e-04,
4.70e-04]    []
7000      [5.19e-05, 1.54e-04, 4.25e-04]    [5.19e-05, 1.54e-04,
4.25e-04]    []
8000      [5.00e-05, 1.44e-04, 3.89e-04]    [5.00e-05, 1.44e-04,
3.89e-04]    []
9000      [4.87e-05, 1.34e-04, 3.57e-04]    [4.87e-05, 1.34e-04,
3.57e-04]    []
10000     [5.46e-05, 1.20e-04, 3.35e-04]    [5.46e-05, 1.20e-04,
```

```
3.35e-04]    []
11000     [4.50e-05, 1.18e-04, 3.05e-04]    [4.50e-05, 1.18e-04,
3.05e-04]    []
12000     [3.96e-05, 1.15e-04, 2.80e-04]    [3.96e-05, 1.15e-04,
2.80e-04]    []
13000     [3.56e-05, 1.12e-04, 2.59e-04]    [3.56e-05, 1.12e-04,
2.59e-04]    []
14000     [3.25e-05, 1.05e-04, 2.43e-04]    [3.25e-05, 1.05e-04,
2.43e-04]    []
15000     [3.07e-05, 1.02e-04, 2.28e-04]    [3.07e-05, 1.02e-04,
2.28e-04]    []

Best model at step 15000:
  train loss: 3.60e-04
  test loss: 3.60e-04
  test metric: []

'train' took 84.258569 s

Compiling model...
'compile' took 0.506700 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [3.07e-05, 1.02e-04, 2.28e-04]    [3.07e-05, 1.02e-04,
2.28e-04]    []
WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
optimizers\tensorflow_compat_v1\scipy_optimizer.py:398: The name
tf.logging.info is deprecated. Please use tf.compat.v1.logging.info
instead.

INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000360
  Number of iterations: 1
  Number of functions evaluations: 30
15015     [3.07e-05, 1.02e-04, 2.28e-04]    [3.07e-05, 1.02e-04,
2.28e-04]    []

Best model at step 15000:
  train loss: 3.60e-04
  test loss: 3.60e-04
  test metric: []

'train' took 0.890586 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
```

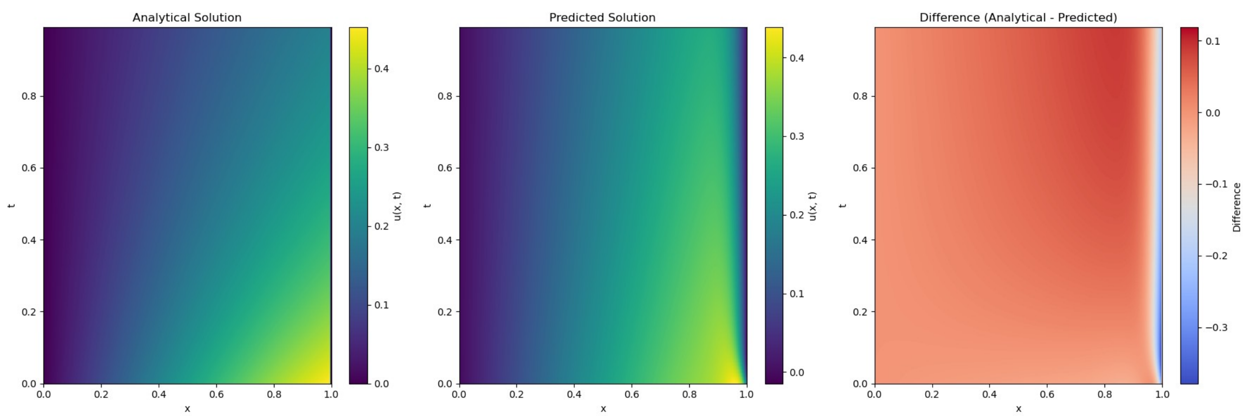
```
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\ME_964\Final_Project\test.dat ...
```






```
Fixed Re = 1, Mean residual: 0.0028968951664865017
Fixed Re = 1, L2 relative error: 0.8406294099463957

# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_fixed.reshape(len(t), len(x))
y_pred_reshaped = y_pred_fixed.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
```

```
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Part (b): Combined Inverse-Forward Problem

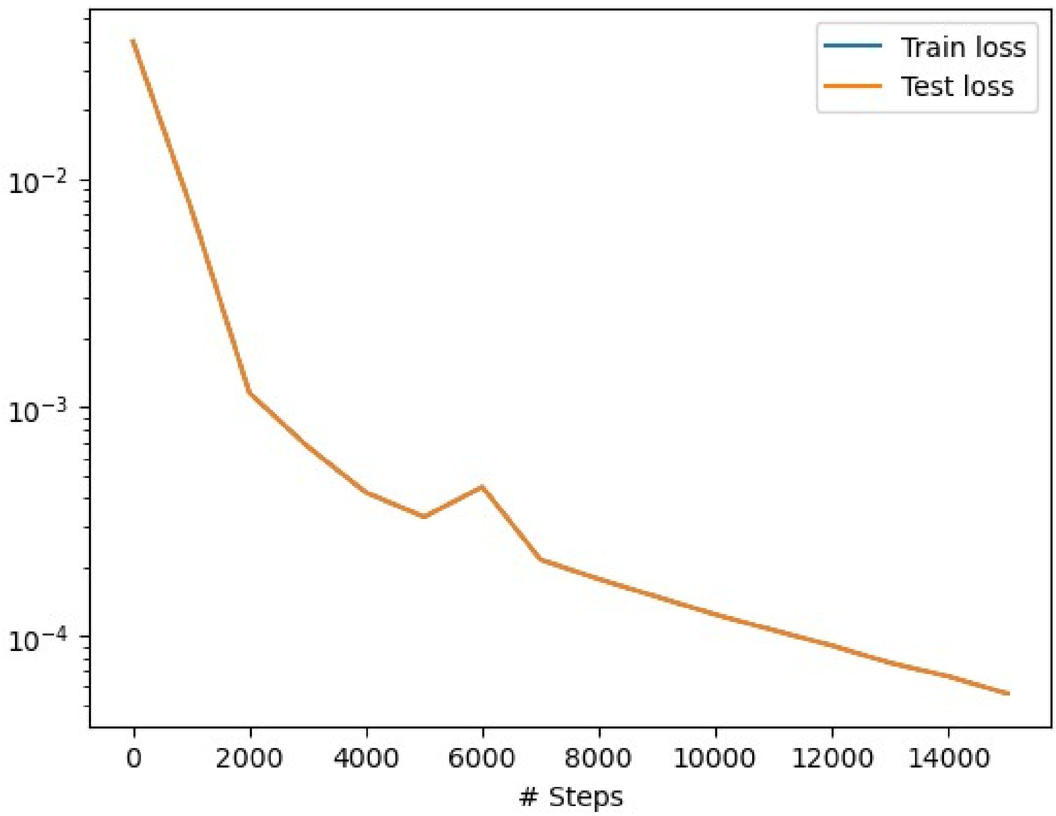
```
# Define Reynolds number as a trainable variable
Re_trainable = tf.Variable(Re_fixed, trainable=True, dtype=tf.float32)

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde_trainable(x, y):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
```

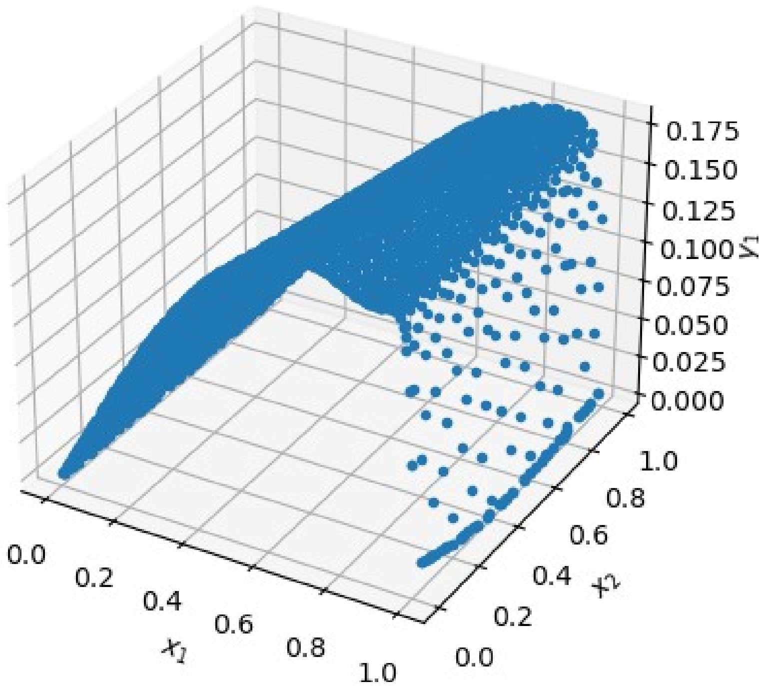
```
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 0.01 / Re_trainable * dy_xx #
Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

# Initial condition matches the analytical solution
to = tf.exp(Re_trainable / 8)
ic = dde.IC(
    geomtime,
    lambda x: x[:, 0:1] / (1 + tf.sqrt(to) * tf.exp(Re_trainable *
x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial,
)

# Dirichlet boundary conditions
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)

# Solve the combined inverse-forward problem
print("Training for combined inverse-forward problem")

# Define dataset for trainable Re
data_trainable = dde.data.TimePDE(
    geomtime, pde_trainable, [bc, ic], num_domain=2540,
num_boundary=80, num_initial=160
)

# Define the neural network
net_trainable = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")

# Compile the model
model_trainable = dde.Model(data_trainable, net_trainable)
model_trainable.compile("adam", lr=1e-3)

# Train the model
model_trainable.train(epochs=15000)
model_trainable.compile("L-BFGS")
losshistory_trainable, train_state_trainable = model_trainable.train()

# Save results
dde.saveplot(losshistory_trainable, train_state_trainable,
issave=True, isplot=True)

# Test the model
X_trainable, y_true_trainable = gen_testdata()
```

```
y_pred_trainable = model_trainable.predict(X_trainable)
f_trainable = model_trainable.predict(X_trainable,
operator=pde_trainable)

# Print errors
print("Mean residual for trainable Re:",
np.mean(np.absolute(f_trainable)))
print("L2 relative error for trainable Re:",
dde.metrics.l2_relative_error(y_true_trainable, y_pred_trainable))

# Use a TensorFlow session to evaluate Re_trainable
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    learned_Re = sess.run(Re_trainable)

print("Learned Reynolds number:", learned_Re)

# Save test results
np.savetxt(f"test_trainable_Re_{learned_Re}.dat",
np.hstack((X_trainable, y_true_trainable, y_pred_trainable)))

Training for combined inverse-forward problem
Compiling model...
Building feed-forward neural network...
'build' took 0.161557 s

'compile' took 1.337566 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [8.22e-03, 1.56e-02, 1.61e-02]    [8.22e-03, 1.56e-02,
1.61e-02]    []
1000      [1.93e-03, 3.72e-04, 5.09e-03]    [1.93e-03, 3.72e-04,
5.09e-03]    []
2000      [2.02e-04, 9.27e-05, 8.58e-04]    [2.02e-04, 9.27e-05,
8.58e-04]    []
3000      [9.87e-05, 4.59e-05, 5.28e-04]    [9.87e-05, 4.59e-05,
5.28e-04]    []
4000      [4.14e-05, 4.91e-05, 3.33e-04]    [4.14e-05, 4.91e-05,
3.33e-04]    []
5000      [3.25e-05, 4.02e-05, 2.58e-04]    [3.25e-05, 4.02e-05,
2.58e-04]    []
6000      [1.27e-04, 1.26e-04, 1.94e-04]    [1.27e-04, 1.26e-04,
1.94e-04]    []
7000      [1.91e-05, 2.79e-05, 1.68e-04]    [1.91e-05, 2.79e-05,
1.68e-04]    []
```

```
8000      [1.58e-05, 2.36e-05, 1.37e-04]    [1.58e-05, 2.36e-05,
1.37e-04]    []
9000      [1.35e-05, 2.21e-05, 1.12e-04]    [1.35e-05, 2.21e-05,
1.12e-04]    []
10000     [1.18e-05, 1.70e-05, 9.48e-05]    [1.18e-05, 1.70e-05,
9.48e-05]    []
11000     [1.05e-05, 1.64e-05, 7.88e-05]    [1.05e-05, 1.64e-05,
7.88e-05]    []
12000     [9.51e-06, 1.28e-05, 6.82e-05]    [9.51e-06, 1.28e-05,
6.82e-05]    []
13000     [7.83e-06, 1.13e-05, 5.68e-05]    [7.83e-06, 1.13e-05,
5.68e-05]    []
14000     [7.42e-06, 1.05e-05, 4.84e-05]    [7.42e-06, 1.05e-05,
4.84e-05]    []
15000     [5.72e-06, 9.50e-06, 4.06e-05]    [5.72e-06, 9.50e-06,
4.06e-05]    []

Best model at step 15000:
  train loss: 5.58e-05
  test loss: 5.58e-05
  test metric: []

'train' took 92.452979 s

Compiling model...
'compile' took 0.721677 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [5.72e-06, 9.50e-06, 4.06e-05]    [5.72e-06, 9.50e-06,
4.06e-05]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000056
  Number of iterations: 1
  Number of functions evaluations: 35
15015     [5.72e-06, 9.50e-06, 4.06e-05]    [5.72e-06, 9.50e-06,
4.06e-05]    []

Best model at step 15000:
  train loss: 5.58e-05
  test loss: 5.58e-05
  test metric: []

'train' took 1.458555 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
```

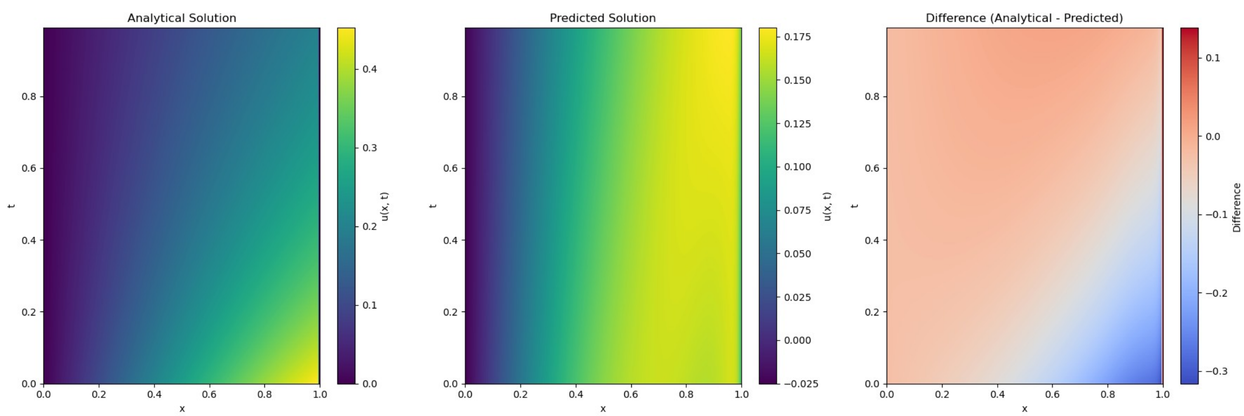
```
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```






```
Mean residual for trainable Re: 0.0011264571
L2 relative error for trainable Re: 0.6567039693839263
Learned Reynolds number: 1.0

# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_trainable.reshape(len(t), len(x))
y_pred_reshaped = y_pred_trainable.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
```

```
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

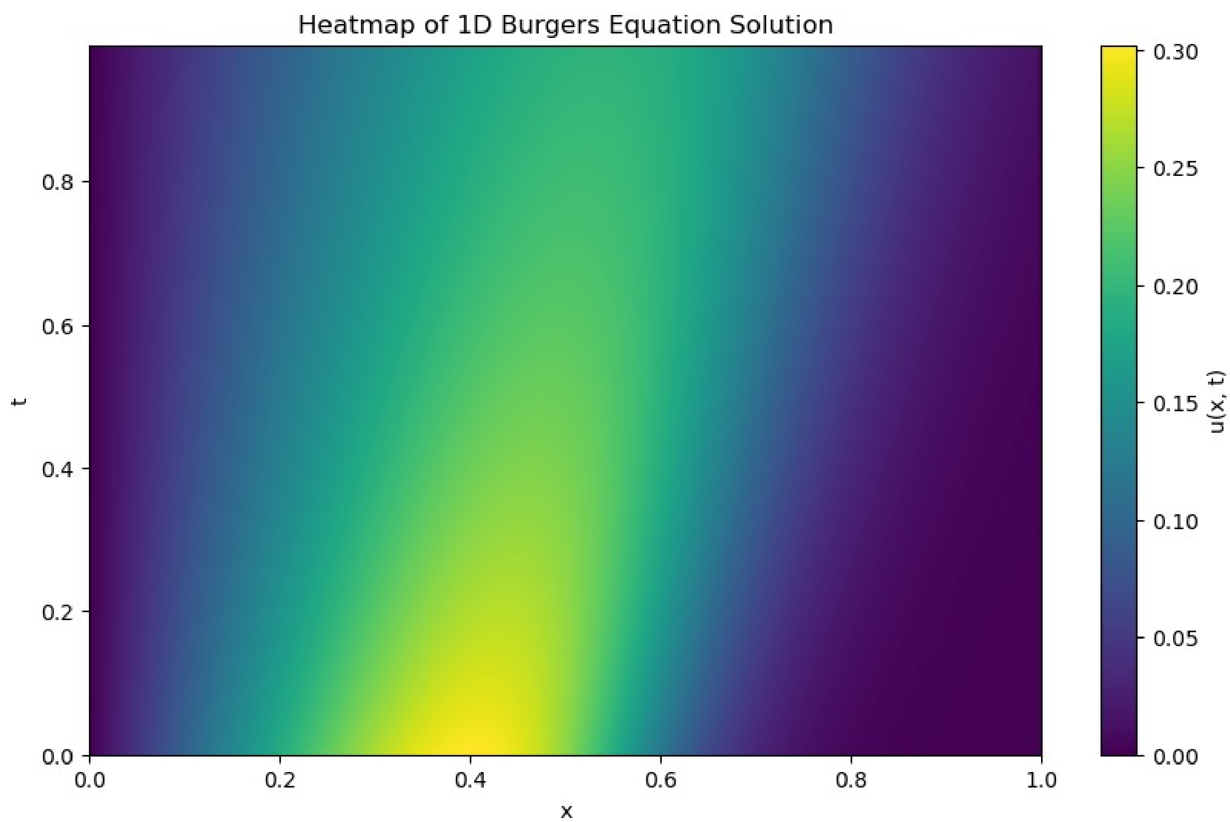


```python
import deepxde as dde
import numpy as np
from deepxde.backend import tf
import matplotlib.pyplot as plt
```

Analytical Solution

```python
# Define the Reynolds number for the analytical solution
# Re = 1,50,100,300
Re_fixed = 50

# Define the spatial and temporal grid
N_t = 100  # Number of time points
N_x = 256  # Number of spatial points
t = np.linspace(0, 0.99, N_t)  # Time grid
x = np.linspace(0, 1, N_x)    # Spatial grid

# Define the analytical solution for the 1D Burgers equation
def analytical_solution(x, t, Re_fixed):
    to = np.exp(Re_fixed / 8)  # Parameter in the equation
    u = x / (t + 1) / (1 + np.sqrt((t + 1) / to) * np.exp(Re_fixed *
x**2 / (4 * (t + 1))))
    return u

# Compute the solution
usol = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x]
for ti in t])

# Explicitly enforce boundary conditions
usol[:, 0] = 0  # u(0, t) = 0
usol[:, -1] = 0  # u(1, t) = 0

# Verify the boundary conditions
assert np.allclose(usol[:, 0], 0), "Boundary condition u(0, t) = 0 not
satisfied"
assert np.allclose(usol[:, -1], 0), "Boundary condition u(1, t) = 0
not satisfied"

# Save the data to a .npz file
np.savez("dataset/Burgers.npz", t=t, x=x, usol=usol)

# Create a new figure window with a size of 10x6 inches
plt.figure(figsize=(10, 6))

# Plot the heatmap
# `usol` is the 2D array containing the solution, `extent` specifies
the range for x and t axes,
# `origin='lower'` ensures the heatmap starts from the bottom-left
corner,
# `aspect='auto'` adjusts the aspect ratio automatically, and
```

```
`cmap='viridis'` sets the colormap
plt.imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],
           origin='lower', aspect='auto', cmap='viridis')

# Add a colorbar to indicate the value of u(x, t) for each color
plt.colorbar(label="u(x, t)")

plt.title("Heatmap of 1D Burgers Equation Solution")
plt.xlabel("x")
plt.ylabel("t")

plt.show()
```



Part (a): Forward Problem

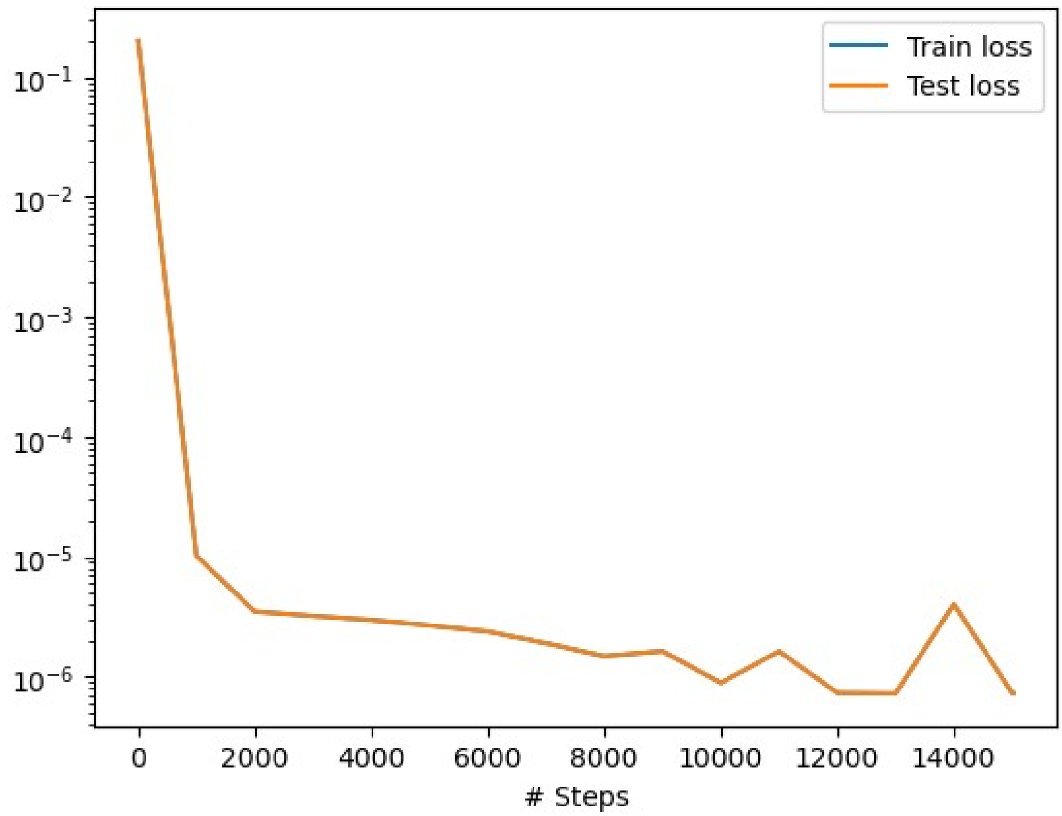
```
# Define fixed Reynolds number
# Re = 1,50,100,300
Re_fixed = 50

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
```

```
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde(x, y, Re):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 0.5 / Re * dy_xx # Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)
ic = dde.IC(
    geomtime, lambda x: x[:, 0:1] / (1 + np.sqrt(np.exp(Re_fixed / 8))
* np.exp(Re_fixed * x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial
)

# Solve the forward problem for fixed Re
print(f"Training for fixed Re = {Re_fixed}")

# Define dataset for fixed Re
data_fixed = dde.data.TimePDE(
    geomtime, lambda x, y: pde(x, y, Re_fixed), [bc, ic],
num_domain=2540, num_boundary=80, num_initial=160
)

# Define the neural network
net_fixed = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")
model_fixed = dde.Model(data_fixed, net_fixed)
model_fixed.compile("adam", lr=1e-3)

# Train the model
model_fixed.train(epochs=15000)
model_fixed.compile("L-BFGS")
losshistory_fixed, train_state_fixed = model_fixed.train()

# Save results
dde.saveplot(losshistory_fixed, train_state_fixed, issave=True,
isplot=True)

# Test the model
X_fixed, y_true_fixed = gen_testdata()
```

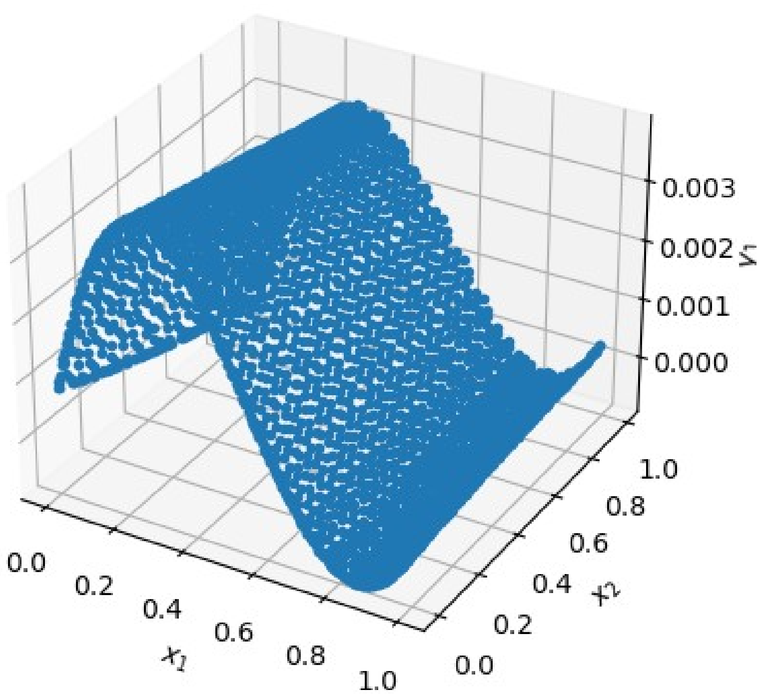
```
y_pred_fixed = model_fixed.predict(X_fixed)
f_fixed = model_fixed.predict(X_fixed, operator=lambda x, y: pde(x, y,
Re_fixed))

# Print errors for fixed Re
print(f"Fixed Re = {Re_fixed}, Mean residual:
{np.mean(np.absolute(f_fixed))}")
print(f"Fixed Re = {Re_fixed}, L2 relative error:
{dde.metrics.l2_relative_error(y_true_fixed, y_pred_fixed)}")
np.savetxt(f"test_fixed_Re_{Re_fixed}.dat", np.hstack((X_fixed,
y_true_fixed, y_pred_fixed)))
```

```
Training for fixed Re = 50
Compiling model...
Building feed-forward neural network...
'build' took 0.060913 s

'compile' took 0.462253 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [2.99e-02, 8.98e-02, 8.23e-02]    [2.99e-02, 8.98e-02,
8.23e-02]    []
1000      [4.80e-06, 1.77e-06, 3.61e-06]    [4.80e-06, 1.77e-06,
3.61e-06]    []
2000      [4.91e-07, 4.45e-07, 2.55e-06]    [4.91e-07, 4.45e-07,
2.55e-06]    []
3000      [2.96e-07, 4.06e-07, 2.50e-06]    [2.96e-07, 4.06e-07,
2.50e-06]    []
4000      [2.49e-07, 3.81e-07, 2.33e-06]    [2.49e-07, 3.81e-07,
2.33e-06]    []
5000      [2.21e-07, 3.49e-07, 2.10e-06]    [2.21e-07, 3.49e-07,
2.10e-06]    []
6000      [2.03e-07, 3.61e-07, 1.81e-06]    [2.03e-07, 3.61e-07,
1.81e-06]    []
7000      [1.69e-07, 2.51e-07, 1.48e-06]    [1.69e-07, 2.51e-07,
1.48e-06]    []
8000      [1.25e-07, 1.94e-07, 1.16e-06]    [1.25e-07, 1.94e-07,
1.16e-06]    []
9000      [4.66e-07, 1.97e-07, 9.64e-07]    [4.66e-07, 1.97e-07,
9.64e-07]    []
10000     [5.46e-08, 1.05e-07, 7.26e-07]    [5.46e-08, 1.05e-07,
7.26e-07]    []
11000     [1.21e-07, 3.92e-07, 1.11e-06]    [1.21e-07, 3.92e-07,
1.11e-06]    []
12000     [3.55e-08, 1.08e-07, 5.87e-07]    [3.55e-08, 1.08e-07,
```

```
5.87e-07]    []
13000     [3.52e-08, 1.03e-07, 5.84e-07]    [3.52e-08, 1.03e-07, 
5.84e-07]    []
14000     [4.00e-07, 2.23e-06, 1.40e-06]    [4.00e-07, 2.23e-06, 
1.40e-06]    []
15000     [4.37e-08, 9.57e-08, 5.87e-07]    [4.37e-08, 9.57e-08, 
5.87e-07]    []

Best model at step 13000:
  train loss: 7.23e-07
  test loss: 7.23e-07
  test metric: []

'train' took 41.743165 s

Compiling model...
'compile' took 0.283328 s

Training model...

Step      Train loss                        Test loss                         
Test metric
15000     [4.37e-08, 9.57e-08, 5.87e-07]    [4.37e-08, 9.57e-08, 
5.87e-07]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000001
  Number of iterations: 1
  Number of functions evaluations: 32
15019     [4.37e-08, 9.57e-08, 5.87e-07]    [4.37e-08, 9.57e-08, 
5.87e-07]    []

Best model at step 13000:
  train loss: 7.23e-07
  test loss: 7.23e-07
  test metric: []

'train' took 0.705540 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```



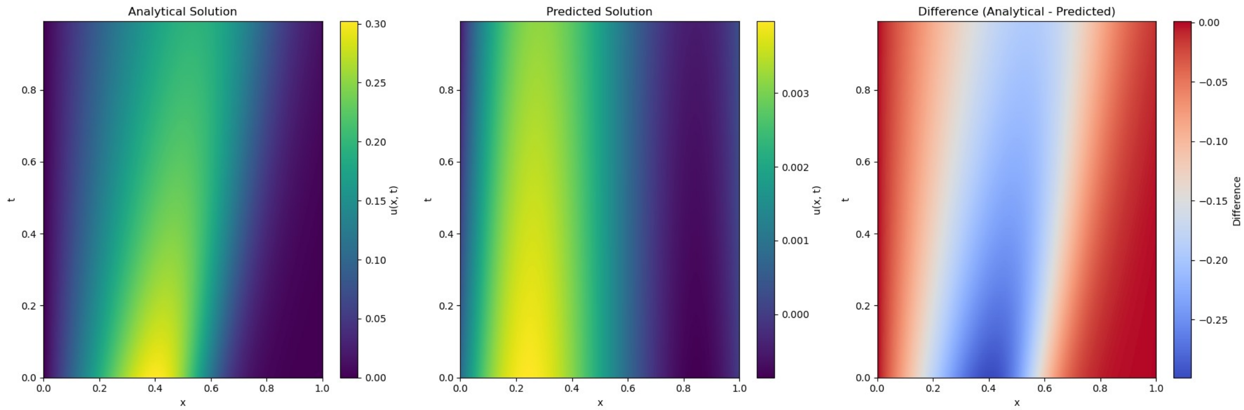
Train loss
Test loss
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
2000
4000
6000
8000
10000
12000
14000
Steps



0.003
0.002
0.001
0.000
v_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
0.0
0.2
0.4
0.6
0.8
1.0
x_2

```
Fixed Re = 50, Mean residual: 0.00013992008462082595
Fixed Re = 50, L2 relative error: 0.9921797726363656
```

```
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_fixed.reshape(len(t), len(x))
y_pred_reshaped = y_pred_fixed.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



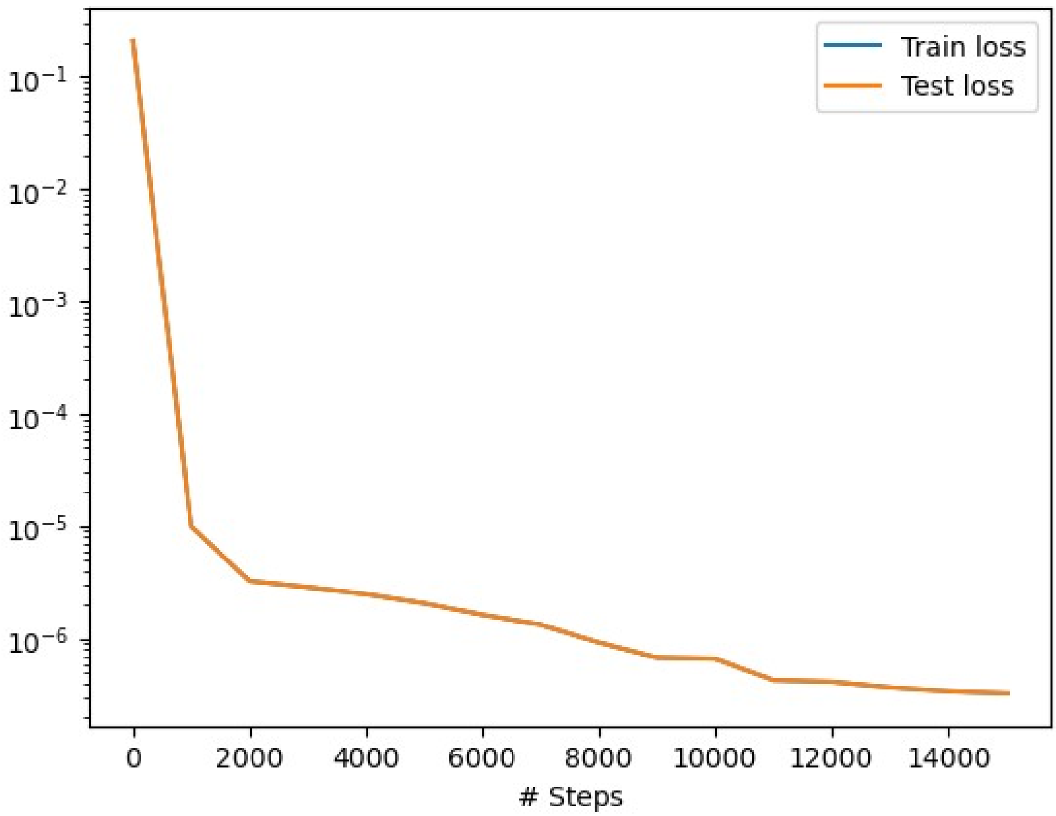
Part (b): Combined Inverse-Forward Problem

```
# Define Reynolds number as a trainable variable
Re_trainable = tf.Variable(Re_fixed, trainable=True, dtype=tf.float32)

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde_trainable(x, y):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 0.5 / Re_trainable * dy_xx #
Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

# Initial condition matches the analytical solution
to = tf.exp(Re_trainable / 8)
ic = dde.IC(
    geomtime,
    lambda x: x[:, 0:1] / (1 + tf.sqrt(to) * tf.exp(Re_trainable *
x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial,
)
```

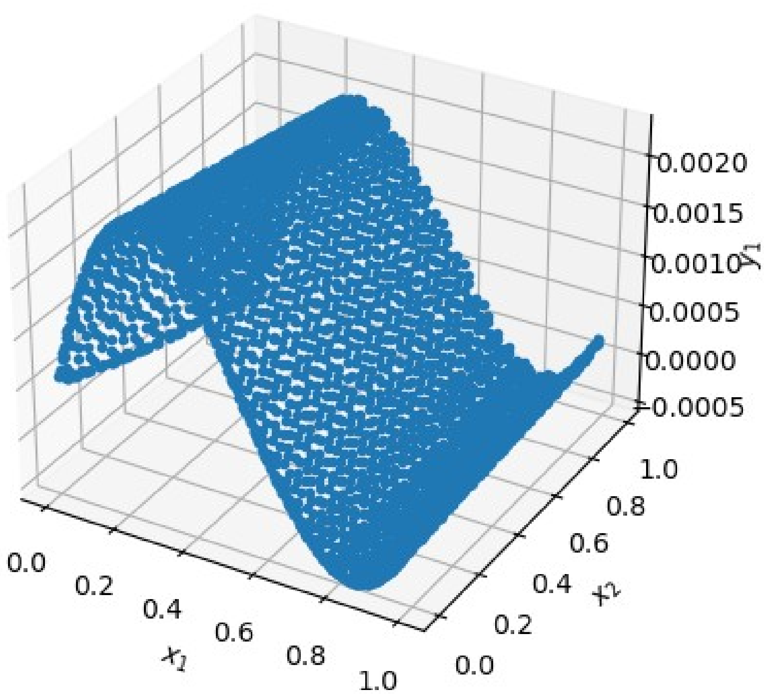
```
# Dirichlet boundary conditions
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)

# Solve the combined inverse-forward problem
print("Training for combined inverse-forward problem")

# Define dataset for trainable Re
data_trainable = dde.data.TimePDE(
    geomtime, pde_trainable, [bc, ic], num_domain=2540,
num_boundary=80, num_initial=160
)

# Define the neural network
net_trainable = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")

# Compile the model
model_trainable = dde.Model(data_trainable, net_trainable)
model_trainable.compile("adam", lr=1e-3)

# Train the model
model_trainable.train(epochs=15000)
model_trainable.compile("L-BFGS")
losshistory_trainable, train_state_trainable = model_trainable.train()

# Save results
dde.saveplot(losshistory_trainable, train_state_trainable,
issave=True, isplot=True)

# Test the model
X_trainable, y_true_trainable = gen_testdata()
y_pred_trainable = model_trainable.predict(X_trainable)
f_trainable = model_trainable.predict(X_trainable,
operator=pde_trainable)

# Print errors
print("Mean residual for trainable Re:",
np.mean(np.absolute(f_trainable)))
print("L2 relative error for trainable Re:",
dde.metrics.l2_relative_error(y_true_trainable, y_pred_trainable))

# Use a TensorFlow session to evaluate Re_trainable
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    learned_Re = sess.run(Re_trainable)

print("Learned Reynolds number:", learned_Re)

# Save test results
```

```
np.savetxt(f"test_trainable_Re_{learned_Re}.dat",
np.hstack((X_trainable, y_true_trainable, y_pred_trainable)))

Training for combined inverse-forward problem
Compiling model...
Building feed-forward neural network...
'build' took 0.059386 s

'compile' took 0.518670 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                          Test loss
Test metric
0         [2.99e-02, 9.48e-02, 8.23e-02]    [2.99e-02, 9.48e-02,
8.23e-02]    []
1000      [4.57e-06, 1.81e-06, 3.55e-06]    [4.57e-06, 1.81e-06,
3.55e-06]    []
2000      [4.40e-07, 4.34e-07, 2.38e-06]    [4.40e-07, 4.34e-07,
2.38e-06]    []
3000      [2.69e-07, 3.74e-07, 2.22e-06]    [2.69e-07, 3.74e-07,
2.22e-06]    []
4000      [2.17e-07, 3.29e-07, 1.94e-06]    [2.17e-07, 3.29e-07,
1.94e-06]    []
5000      [1.79e-07, 2.79e-07, 1.61e-06]    [1.79e-07, 2.79e-07,
1.61e-06]    []
6000      [1.51e-07, 2.19e-07, 1.26e-06]    [1.51e-07, 2.19e-07,
1.26e-06]    []
7000      [1.27e-07, 2.46e-07, 9.58e-07]    [1.27e-07, 2.46e-07,
9.58e-07]    []
8000      [8.93e-08, 1.27e-07, 7.05e-07]    [8.93e-08, 1.27e-07,
7.05e-07]    []
9000      [5.56e-08, 9.21e-08, 5.30e-07]    [5.56e-08, 9.21e-08,
5.30e-07]    []
10000     [1.57e-07, 7.80e-08, 4.28e-07]    [1.57e-07, 7.80e-08,
4.28e-07]    []
11000     [2.28e-08, 5.46e-08, 3.49e-07]    [2.28e-08, 5.46e-08,
3.49e-07]    []
12000     [1.99e-08, 4.13e-08, 3.51e-07]    [1.99e-08, 4.13e-08,
3.51e-07]    []
13000     [1.52e-08, 3.27e-08, 3.19e-07]    [1.52e-08, 3.27e-08,
3.19e-07]    []
14000     [1.65e-08, 4.14e-08, 2.82e-07]    [1.65e-08, 4.14e-08,
2.82e-07]    []
15000     [1.50e-08, 4.64e-08, 2.64e-07]    [1.50e-08, 4.64e-08,
2.64e-07]    []

Best model at step 15000:
```

```
  train loss: 3.25e-07
  test loss: 3.25e-07
  test metric: []

'train' took 42.691809 s

Compiling model...
'compile' took 0.347429 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [1.50e-08, 4.64e-08, 2.64e-07]    [1.50e-08, 4.64e-08,
2.64e-07]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000000
  Number of iterations: 1
  Number of functions evaluations: 35
15018     [1.50e-08, 4.64e-08, 2.64e-07]    [1.50e-08, 4.64e-08,
2.64e-07]    []

Best model at step 15000:
  train loss: 3.25e-07
  test loss: 3.25e-07
  test metric: []

'train' took 0.913704 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```



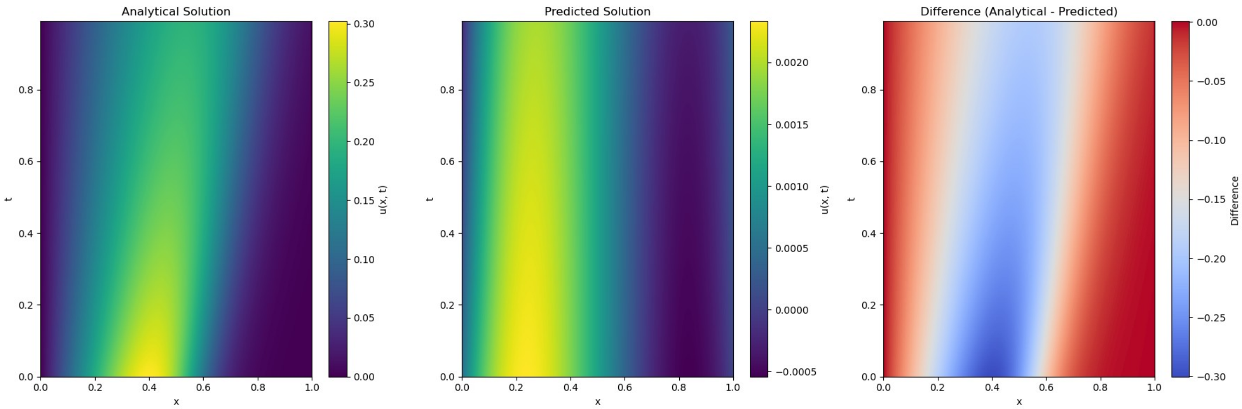
Train loss
Test loss
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
2000
4000
6000
8000
10000
12000
14000
Steps



0.0020
0.0015
0.0010
0.0005
0.0000
-0.0005
y_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
x_2

```
Mean residual for trainable Re: 9.480372e-05
L2 relative error for trainable Re: 0.9953817047510793
Learned Reynolds number: 50.0
```

```
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_trainable.reshape(len(t), len(x))
y_pred_reshaped = y_pred_trainable.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Analytical Solution
Predicted Solution
Difference (Analytical - Predicted)
x
t
u(x, t)
Difference

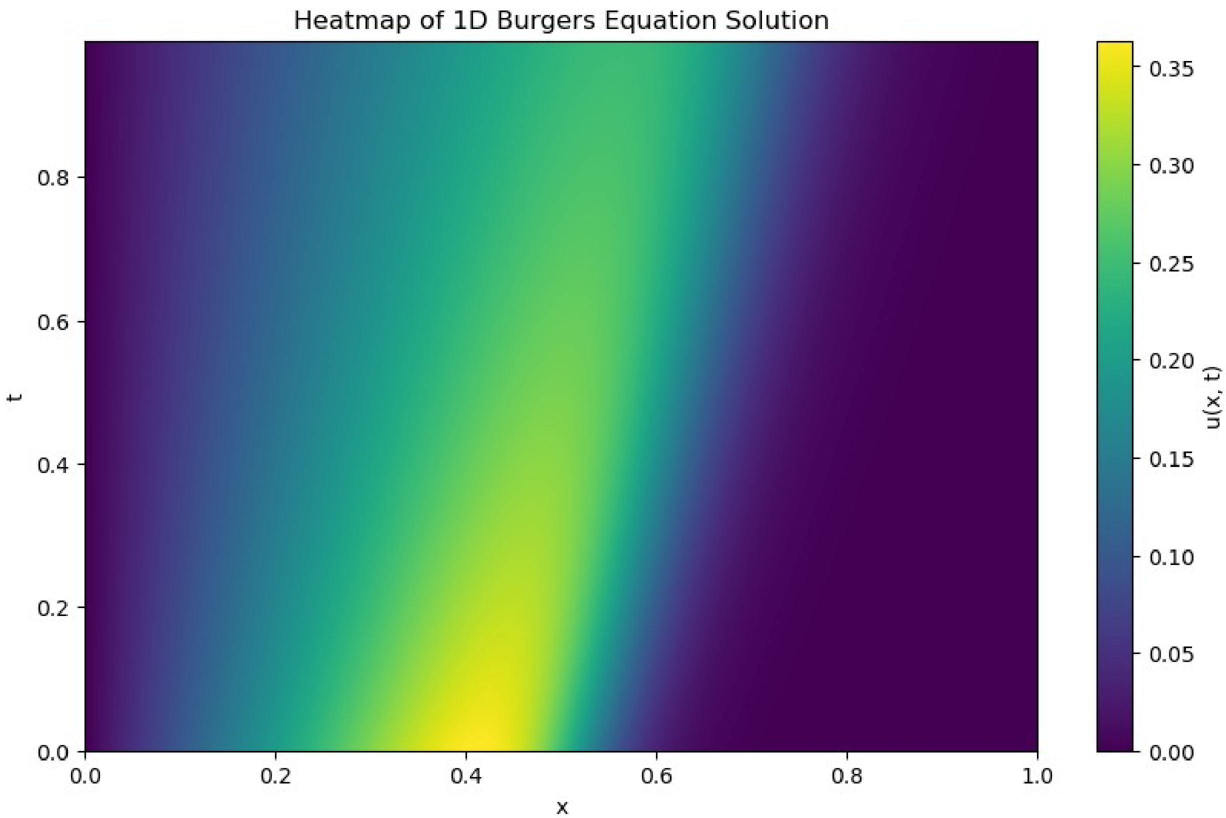
```
import deepxde as dde
import numpy as np
from deepxde.backend import tf
import matplotlib.pyplot as plt
```

```
Using backend: tensorflow.compat.v1
Other supported backends: tensorflow, pytorch, jax, paddle.
paddle supports more examples now and is recommended.

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
backend\tensorflow_compat_v1\tensor.py:25: The name
tf.disable_v2_behavior is deprecated. Please use
tf.compat.v1.disable_v2_behavior instead.

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\tensorflow\
python\compat\v2_compat.py:98: disable_resource_variables (from
tensorflow.python.ops.resource_variables_toggle) is deprecated and
will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

Analytical Solution

```
# Define the Reynolds number for the analytical solution
# Re = 1,50,100,300
Re_fixed = 100

# Define the spatial and temporal grid
N_t = 100  # Number of time points
N_x = 256  # Number of spatial points
t = np.linspace(0, 0.99, N_t)  # Time grid
x = np.linspace(0, 1, N_x)    # Spatial grid

# Define the analytical solution for the 1D Burgers equation
def analytical_solution(x, t, Re_fixed):
    to = np.exp(Re_fixed / 8)  # Parameter in the equation
    u = x / (t + 1) / (1 + np.sqrt((t + 1) / to) * np.exp(Re_fixed *
x**2 / (4 * (t + 1))))
    return u

# Compute the solution
usol = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x]
for ti in t])

# Explicitly enforce boundary conditions
usol[:, 0] = 0  # u(0, t) = 0
usol[:, -1] = 0  # u(1, t) = 0

# Verify the boundary conditions
assert np.allclose(usol[:, 0], 0), "Boundary condition u(0, t) = 0 not
```

```
satisfied"
assert np.allclose(usol[:, -1], 0), "Boundary condition u(1, t) = 0
not satisfied"

# Save the data to a .npz file
np.savez("dataset/Burgers.npz", t=t, x=x, usol=usol)

# Create a new figure window with a size of 10x6 inches
plt.figure(figsize=(10, 6))

# Plot the heatmap
# `usol` is the 2D array containing the solution, `extent` specifies
the range for x and t axes,
# `origin='lower'` ensures the heatmap starts from the bottom-left
corner,
# `aspect='auto'` adjusts the aspect ratio automatically, and
`cmap='viridis'` sets the colormap
plt.imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],
           origin='lower', aspect='auto', cmap='viridis')

# Add a colorbar to indicate the value of u(x, t) for each color
plt.colorbar(label="u(x, t)")

plt.title("Heatmap of 1D Burgers Equation Solution")
plt.xlabel("x")
plt.ylabel("t")

plt.show()
```



Part (a): Forward Problem

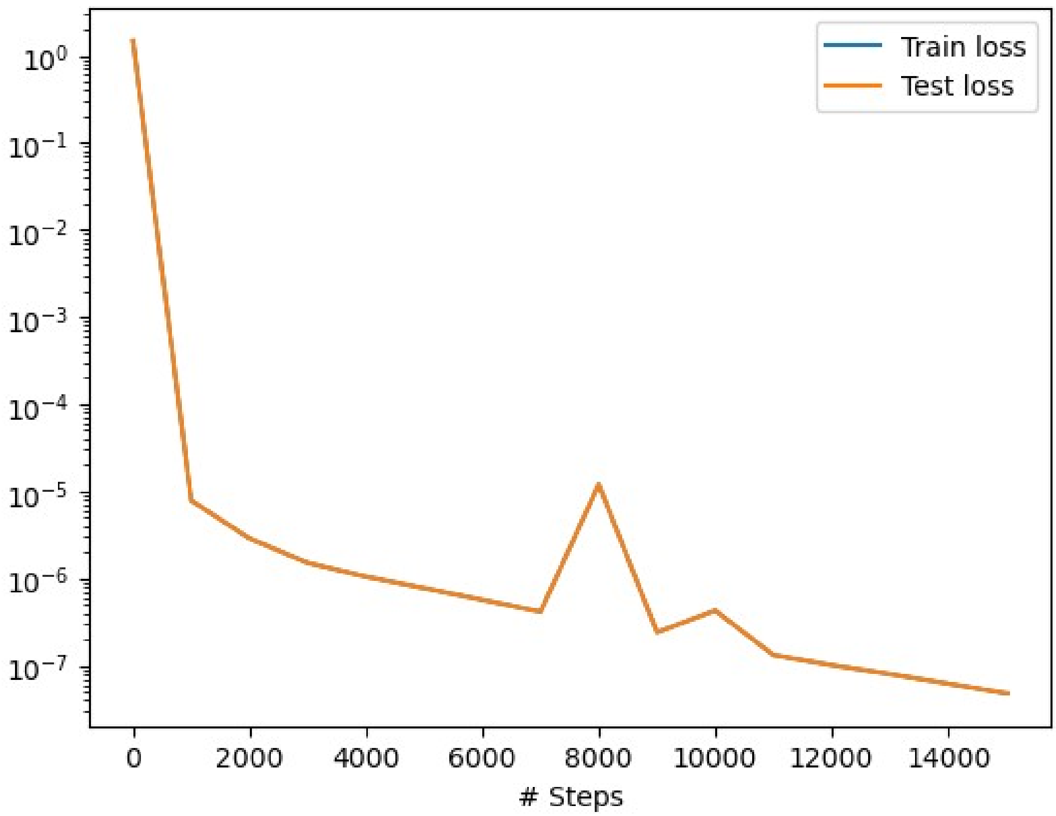
```
# Define fixed Reynolds number
# Re = 1,50,100,300
Re_fixed = 100

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde(x, y, Re):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 1 / Re * dy_xx # Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
```

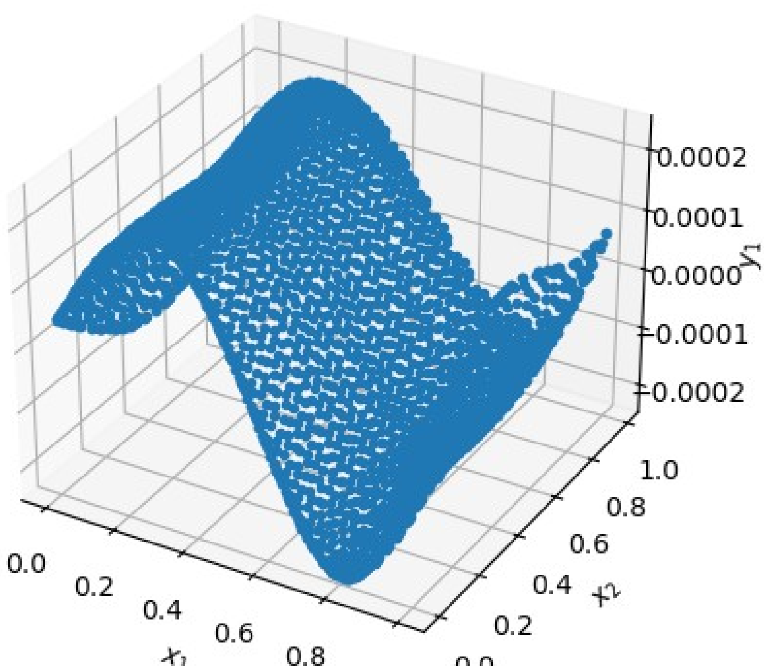
```
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)
ic = dde.IC(
    geomtime, lambda x: x[:, 0:1] / (1 + np.sqrt(np.exp(Re_fixed / 8))
* np.exp(Re_fixed * x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial
)

# Solve the forward problem for fixed Re
print(f"Training for fixed Re = {Re_fixed}")

# Define dataset for fixed Re
data_fixed = dde.data.TimePDE(
    geomtime, lambda x, y: pde(x, y, Re_fixed), [bc, ic],
num_domain=2540, num_boundary=80, num_initial=160
)

# Define the neural network
net_fixed = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")
model_fixed = dde.Model(data_fixed, net_fixed)
model_fixed.compile("adam", lr=1e-3)

# Train the model
model_fixed.train(epochs=15000)
model_fixed.compile("L-BFGS")
losshistory_fixed, train_state_fixed = model_fixed.train()

# Save results
dde.saveplot(losshistory_fixed, train_state_fixed, issave=True,
isplot=True)

# Test the model
X_fixed, y_true_fixed = gen_testdata()
y_pred_fixed = model_fixed.predict(X_fixed)
f_fixed = model_fixed.predict(X_fixed, operator=lambda x, y: pde(x, y,
Re_fixed))

# Print errors for fixed Re
print(f"Fixed Re = {Re_fixed}, Mean residual:
{np.mean(np.absolute(f_fixed))}")
print(f"Fixed Re = {Re_fixed}, L2 relative error:
{dde.metrics.l2_relative_error(y_true_fixed, y_pred_fixed)}")
np.savetxt(f"test_fixed_Re_{Re_fixed}.dat", np.hstack((X_fixed,
y_true_fixed, y_pred_fixed)))
```

```
Training for fixed Re = 100
Compiling model...
```

```
Building feed-forward neural network...
'build' took 0.068291 s

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
model.py:168: The name tf.train.Saver is deprecated. Please use
tf.compat.v1.train.Saver instead.

'compile' took 0.590234 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [5.50e-01, 4.09e-01, 5.17e-01]    [5.50e-01, 4.09e-01,
5.17e-01]    []
1000      [5.72e-06, 9.39e-07, 1.18e-06]    [5.72e-06, 9.39e-07,
1.18e-06]    []
2000      [1.85e-06, 4.39e-07, 5.97e-07]    [1.85e-06, 4.39e-07,
5.97e-07]    []
3000      [9.39e-07, 1.95e-07, 3.75e-07]    [9.39e-07, 1.95e-07,
3.75e-07]    []
4000      [6.00e-07, 1.21e-07, 3.26e-07]    [6.00e-07, 1.21e-07,
3.26e-07]    []
5000      [4.04e-07, 8.92e-08, 2.82e-07]    [4.04e-07, 8.92e-08,
2.82e-07]    []
6000      [2.72e-07, 6.70e-08, 2.29e-07]    [2.72e-07, 6.70e-08,
2.29e-07]    []
7000      [1.89e-07, 5.02e-08, 1.76e-07]    [1.89e-07, 5.02e-08,
1.76e-07]    []
8000      [3.65e-06, 6.06e-06, 2.28e-06]    [3.65e-06, 6.06e-06,
2.28e-06]    []
9000      [1.08e-07, 3.45e-08, 9.79e-08]    [1.08e-07, 3.45e-08,
9.79e-08]    []
10000     [1.92e-07, 1.21e-07, 1.13e-07]    [1.92e-07, 1.21e-07,
1.13e-07]    []
11000     [6.05e-08, 1.50e-08, 5.67e-08]    [6.05e-08, 1.50e-08,
5.67e-08]    []
12000     [4.67e-08, 1.14e-08, 4.30e-08]    [4.67e-08, 1.14e-08,
4.30e-08]    []
13000     [3.85e-08, 1.15e-08, 2.99e-08]    [3.85e-08, 1.15e-08,
2.99e-08]    []
14000     [3.05e-08, 6.41e-09, 2.50e-08]    [3.05e-08, 6.41e-09,
2.50e-08]    []
15000     [2.49e-08, 4.57e-09, 1.84e-08]    [2.49e-08, 4.57e-09,
1.84e-08]    []

Best model at step 15000:
  train loss: 4.79e-08
```

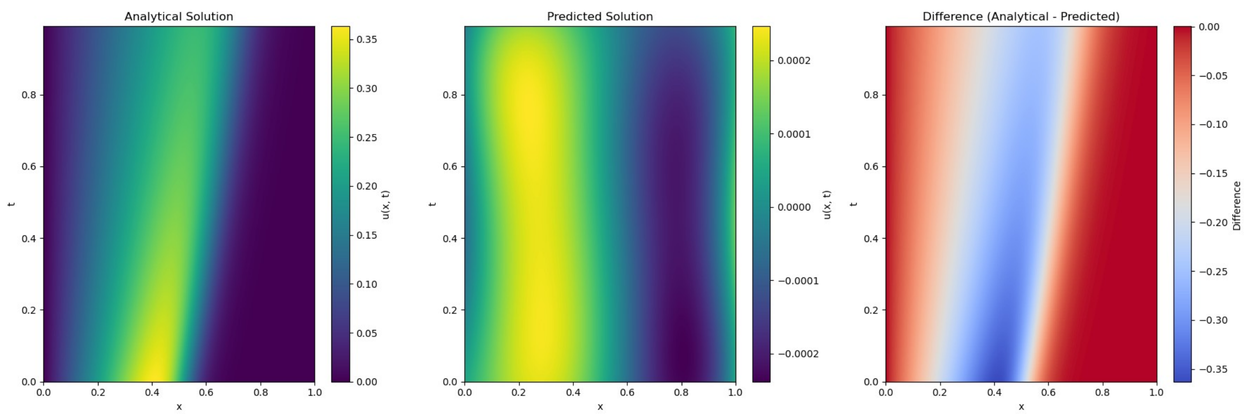
```
  test loss: 4.79e-08
  test metric: []

'train' took 41.870211 s

Compiling model...
'compile' took 0.196505 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [2.49e-08, 4.57e-09, 1.84e-08]    [2.49e-08, 4.57e-09,
1.84e-08]    []
WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
optimizers\tensorflow_compat_v1\scipy_optimizer.py:398: The name
tf.logging.info is deprecated. Please use tf.compat.v1.logging.info
instead.

INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000000
  Number of iterations: 1
  Number of functions evaluations: 30
15017     [2.49e-08, 4.57e-09, 1.84e-08]    [2.49e-08, 4.57e-09,
1.84e-08]    []

Best model at step 15000:
  train loss: 4.79e-08
  test loss: 4.79e-08
  test metric: []

'train' took 0.381176 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```


Train loss
Test loss
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
2000
4000
6000
8000
10000
12000
14000
Steps


0.0002
0.0001
0.0000
−0.0001
−0.0002
y_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
0.0
0.2
0.4
0.6
0.8
1.0
x_2

```
Fixed Re = 100, Mean residual: 0.00011355217429809272
Fixed Re = 100, L2 relative error: 0.9998883381317551
```

```python
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_fixed.reshape(len(t), len(x))
y_pred_reshaped = y_pred_fixed.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



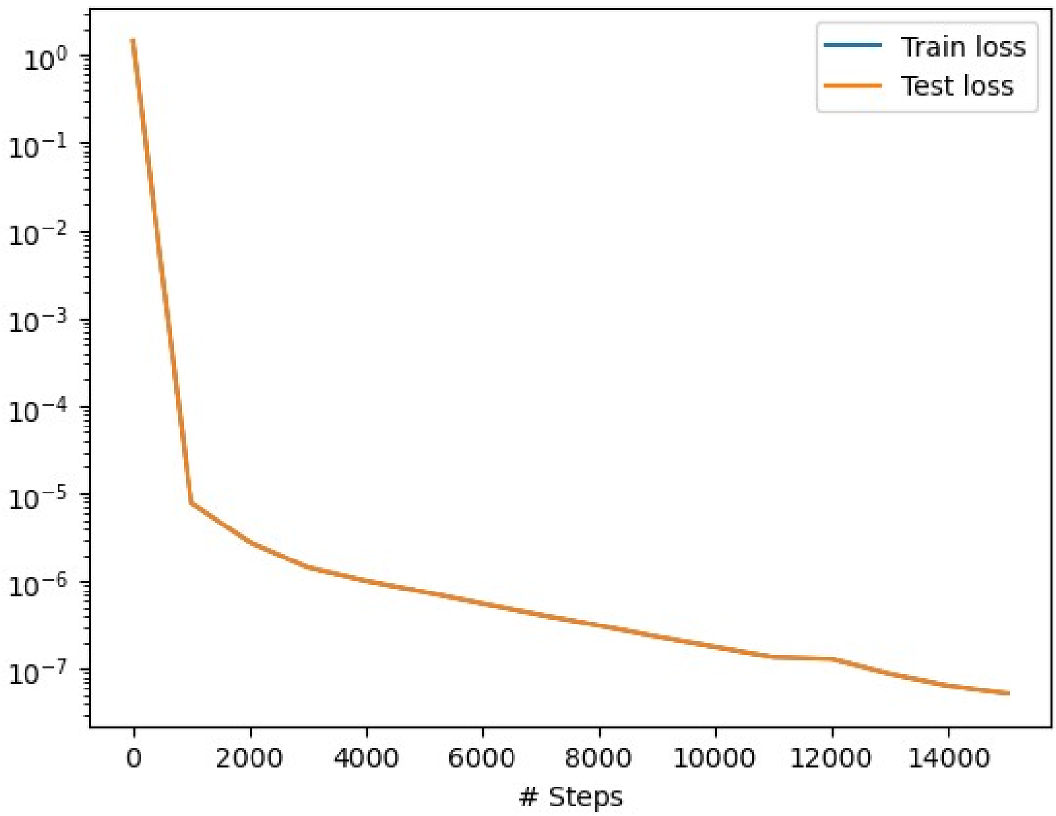
Part (b): Combined Inverse-Forward Problem

```python
# Define Reynolds number as a trainable variable
Re_trainable = tf.Variable(Re_fixed, trainable=True, dtype=tf.float32)

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde_trainable(x, y):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 1 / Re_trainable * dy_xx # Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

# Initial condition matches the analytical solution
to = tf.exp(Re_trainable / 8)
ic = dde.IC(
    geomtime,
    lambda x: x[:, 0:1] / (1 + tf.sqrt(to) * tf.exp(Re_trainable * x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial,
)
```

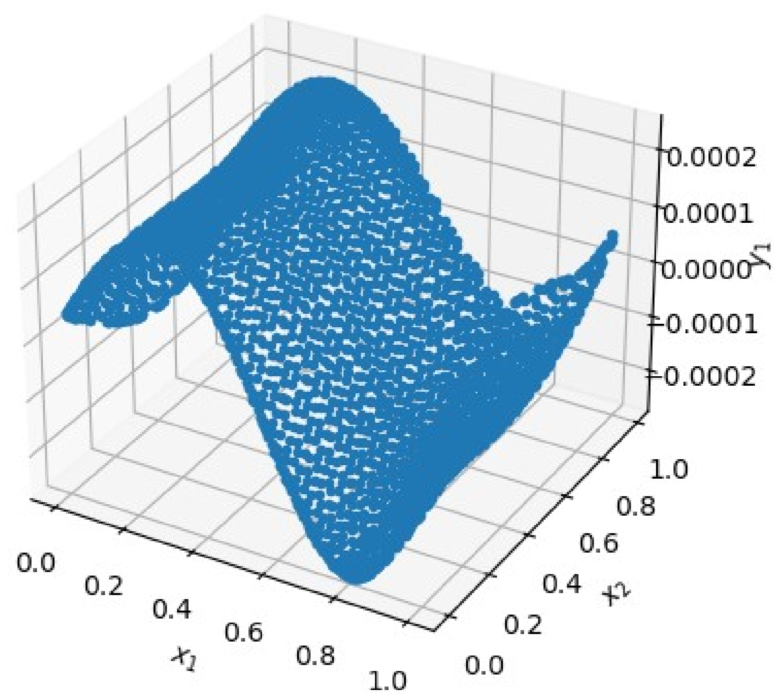
```
# Dirichlet boundary conditions
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)

# Solve the combined inverse-forward problem
print("Training for combined inverse-forward problem")

# Define dataset for trainable Re
data_trainable = dde.data.TimePDE(
    geomtime, pde_trainable, [bc, ic], num_domain=2540,
num_boundary=80, num_initial=160
)

# Define the neural network
net_trainable = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")

# Compile the model
model_trainable = dde.Model(data_trainable, net_trainable)
model_trainable.compile("adam", lr=1e-3)

# Train the model
model_trainable.train(epochs=15000)
model_trainable.compile("L-BFGS")
losshistory_trainable, train_state_trainable = model_trainable.train()

# Save results
dde.saveplot(losshistory_trainable, train_state_trainable,
issave=True, isplot=True)

# Test the model
X_trainable, y_true_trainable = gen_testdata()
y_pred_trainable = model_trainable.predict(X_trainable)
f_trainable = model_trainable.predict(X_trainable,
operator=pde_trainable)

# Print errors
print("Mean residual for trainable Re:",
np.mean(np.absolute(f_trainable)))
print("L2 relative error for trainable Re:",
dde.metrics.l2_relative_error(y_true_trainable, y_pred_trainable))

# Use a TensorFlow session to evaluate Re_trainable
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    learned_Re = sess.run(Re_trainable)

print("Learned Reynolds number:", learned_Re)

# Save test results
```

```
np.savetxt(f"test_trainable_Re_{learned_Re}.dat",
np.hstack((X_trainable, y_true_trainable, y_pred_trainable)))

Training for combined inverse-forward problem
Compiling model...
Building feed-forward neural network...
'build' took 0.059683 s

'compile' took 0.470328 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                       Test loss
Test metric
0         [5.46e-01, 3.92e-01, 5.17e-01]    [5.46e-01, 3.92e-01,
5.17e-01]    []
1000      [6.06e-06, 7.66e-07, 1.01e-06]    [6.06e-06, 7.66e-07,
1.01e-06]    []
2000      [1.85e-06, 3.60e-07, 5.98e-07]    [1.85e-06, 3.60e-07,
5.98e-07]    []
3000      [8.70e-07, 1.67e-07, 4.04e-07]    [8.70e-07, 1.67e-07,
4.04e-07]    []
4000      [5.55e-07, 1.09e-07, 3.49e-07]    [5.55e-07, 1.09e-07,
3.49e-07]    []
5000      [3.76e-07, 8.28e-08, 2.98e-07]    [3.76e-07, 8.28e-08,
2.98e-07]    []
6000      [2.56e-07, 6.27e-08, 2.39e-07]    [2.56e-07, 6.27e-08,
2.39e-07]    []
7000      [1.83e-07, 4.68e-08, 1.84e-07]    [1.83e-07, 4.68e-08,
1.84e-07]    []
8000      [1.38e-07, 3.63e-08, 1.40e-07]    [1.38e-07, 3.63e-08,
1.40e-07]    []
9000      [1.05e-07, 2.58e-08, 1.04e-07]    [1.05e-07, 2.58e-08,
1.04e-07]    []
10000     [8.02e-08, 1.97e-08, 7.90e-08]    [8.02e-08, 1.97e-08,
7.90e-08]    []
11000     [6.13e-08, 1.48e-08, 6.06e-08]    [6.13e-08, 1.48e-08,
6.06e-08]    []
12000     [5.32e-08, 3.15e-08, 4.54e-08]    [5.32e-08, 3.15e-08,
4.54e-08]    []
13000     [4.20e-08, 1.15e-08, 3.45e-08]    [4.20e-08, 1.15e-08,
3.45e-08]    []
14000     [3.13e-08, 6.16e-09, 2.70e-08]    [3.13e-08, 6.16e-09,
2.70e-08]    []
15000     [2.82e-08, 3.40e-09, 2.11e-08]    [2.82e-08, 3.40e-09,
2.11e-08]    []

Best model at step 15000:
```

```
  train loss: 5.27e-08
  test loss: 5.27e-08
  test metric: []

'train' took 41.958719 s

Compiling model...
'compile' took 0.273623 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [2.82e-08, 3.40e-09, 2.11e-08]    [2.82e-08, 3.40e-09,
2.11e-08]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000000
  Number of iterations: 1
  Number of functions evaluations: 35
15017     [2.82e-08, 3.40e-09, 2.11e-08]    [2.82e-08, 3.40e-09,
2.11e-08]    []

Best model at step 15000:
  train loss: 5.27e-08
  test loss: 5.27e-08
  test metric: []

'train' took 0.521041 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```



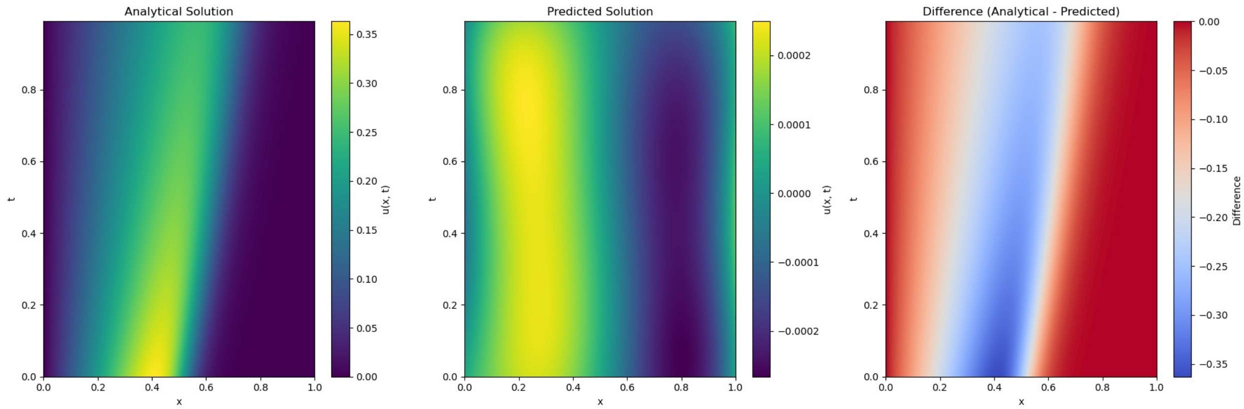
Train loss
Test loss
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
2000
4000
6000
8000
10000
12000
14000
Steps



0.0002
0.0001
0.0000
−0.0001
−0.0002
y_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
x_2

```
Mean residual for trainable Re: 0.00012403584
L2 relative error for trainable Re: 0.9999573442808729
WARNING:tensorflow:From C:\Users\jhyang\AppData\Local\Temp\
ipykernel_1136\3495667732.py:69: The name tf.Session is deprecated.
Please use tf.compat.v1.Session instead.

Learned Reynolds number: 100.0
```

```
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_trainable.reshape(len(t), len(x))
y_pred_reshaped = y_pred_trainable.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```


Analytical Solution
Predicted Solution
Difference (Analytical - Predicted)
t
x
u(x, t)
Difference
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.0002
0.0001
0.0000
−0.0001
−0.0002
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35

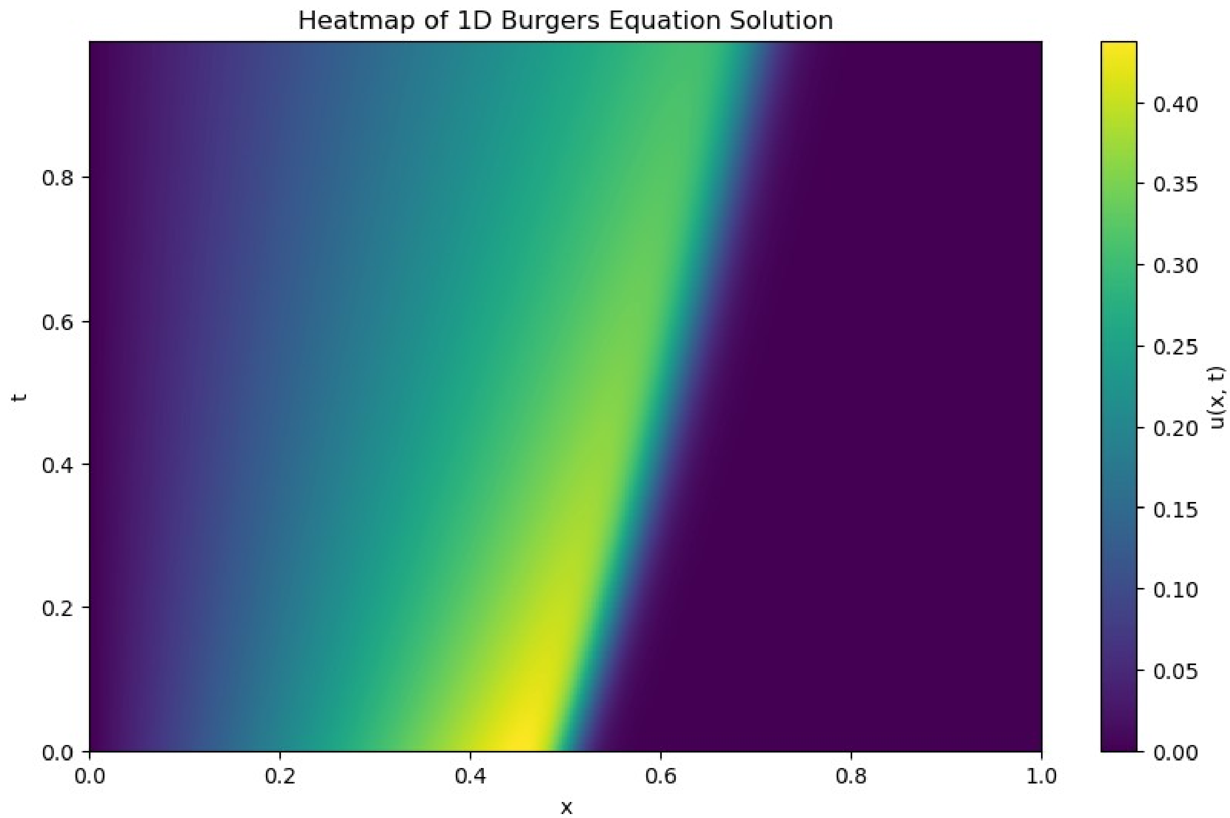
```python
import deepxde as dde
import numpy as np
from deepxde.backend import tf
import matplotlib.pyplot as plt
```

```
Using backend: tensorflow.compat.v1
Other supported backends: tensorflow, pytorch, jax, paddle.
paddle supports more examples now and is recommended.

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
backend\tensorflow_compat_v1\tensor.py:25: The name
tf.disable_v2_behavior is deprecated. Please use
tf.compat.v1.disable_v2_behavior instead.

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\tensorflow\
python\compat\v2_compat.py:98: disable_resource_variables (from
tensorflow.python.ops.resource_variables_toggle) is deprecated and
will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

Analytical Solution

```python
# Define the Reynolds number for the analytical solution
# Re = 1,50,100,300
Re_fixed = 300

# Define the spatial and temporal grid
N_t = 100  # Number of time points
N_x = 256  # Number of spatial points
t = np.linspace(0, 0.99, N_t)  # Time grid
x = np.linspace(0, 1, N_x)    # Spatial grid

# Define the analytical solution for the 1D Burgers equation
def analytical_solution(x, t, Re_fixed):
    to = np.exp(Re_fixed / 8)  # Parameter in the equation
    u = x / (t + 1) / (1 + np.sqrt((t + 1) / to) * np.exp(Re_fixed *
x**2 / (4 * (t + 1))))
    return u

# Compute the solution
usol = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x]
for ti in t])

# Explicitly enforce boundary conditions
usol[:, 0] = 0  # u(0, t) = 0
usol[:, -1] = 0  # u(1, t) = 0

# Verify the boundary conditions
assert np.allclose(usol[:, 0], 0), "Boundary condition u(0, t) = 0 not
```

```
satisfied"
assert np.allclose(usol[:, -1], 0), "Boundary condition u(1, t) = 0
not satisfied"

# Save the data to a .npz file
np.savez("dataset/Burgers.npz", t=t, x=x, usol=usol)

# Create a new figure window with a size of 10x6 inches
plt.figure(figsize=(10, 6))

# Plot the heatmap
# `usol` is the 2D array containing the solution, `extent` specifies
the range for x and t axes,
# `origin='lower'` ensures the heatmap starts from the bottom-left
corner,
# `aspect='auto'` adjusts the aspect ratio automatically, and
`cmap='viridis'` sets the colormap
plt.imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],
           origin='lower', aspect='auto', cmap='viridis')

# Add a colorbar to indicate the value of u(x, t) for each color
plt.colorbar(label="u(x, t)")

plt.title("Heatmap of 1D Burgers Equation Solution")
plt.xlabel("x")
plt.ylabel("t")

plt.show()
```



Part (a): Forward Problem

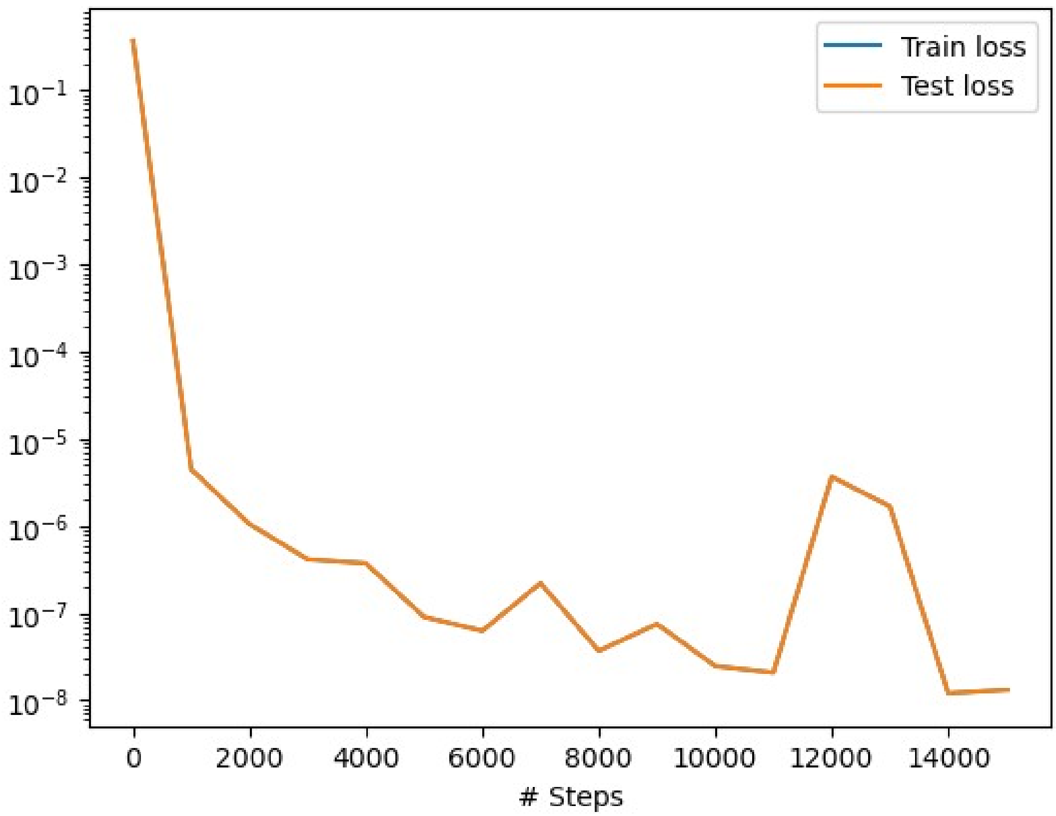
```python
# Define fixed Reynolds number
# Re = 1,50,100,300
Re_fixed = 300

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde(x, y, Re):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 3 / Re * dy_xx # Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
```

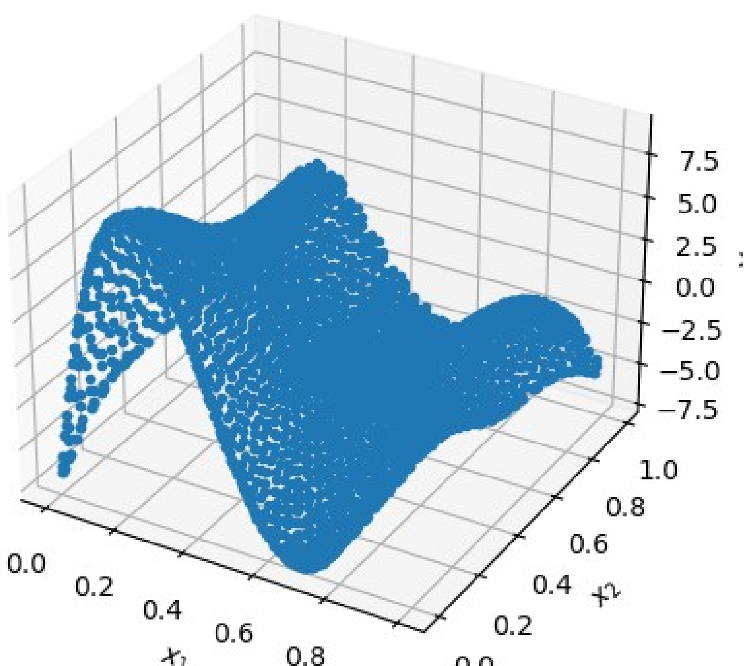
```
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)
ic = dde.IC(
    geomtime, lambda x: x[:, 0:1] / (1 + np.sqrt(np.exp(Re_fixed / 8))
* np.exp(Re_fixed * x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial
)

# Solve the forward problem for fixed Re
print(f"Training for fixed Re = {Re_fixed}")

# Define dataset for fixed Re
data_fixed = dde.data.TimePDE(
    geomtime, lambda x, y: pde(x, y, Re_fixed), [bc, ic],
num_domain=2540, num_boundary=80, num_initial=160
)

# Define the neural network
net_fixed = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")
model_fixed = dde.Model(data_fixed, net_fixed)
model_fixed.compile("adam", lr=1e-3)

# Train the model
model_fixed.train(epochs=15000)
model_fixed.compile("L-BFGS")
losshistory_fixed, train_state_fixed = model_fixed.train()

# Save results
dde.saveplot(losshistory_fixed, train_state_fixed, issave=True,
isplot=True)

# Test the model
X_fixed, y_true_fixed = gen_testdata()
y_pred_fixed = model_fixed.predict(X_fixed)
f_fixed = model_fixed.predict(X_fixed, operator=lambda x, y: pde(x, y,
Re_fixed))

# Print errors for fixed Re
print(f"Fixed Re = {Re_fixed}, Mean residual:
{np.mean(np.absolute(f_fixed))}")
print(f"Fixed Re = {Re_fixed}, L2 relative error:
{dde.metrics.l2_relative_error(y_true_fixed, y_pred_fixed)}")
np.savetxt(f"test_fixed_Re_{Re_fixed}.dat", np.hstack((X_fixed,
y_true_fixed, y_pred_fixed)))
```

```
Training for fixed Re = 300
Compiling model...
```

```
Building feed-forward neural network...
'build' took 0.063288 s

WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
model.py:168: The name tf.train.Saver is deprecated. Please use
tf.compat.v1.train.Saver instead.


C:\Users\jhyang\AppData\Local\Temp\ipykernel_18744\2581919726.py:27:
RuntimeWarning: overflow encountered in multiply
  geomtime, lambda x: x[:, 0:1] / (1 + np.sqrt(np.exp(Re_fixed / 8)) *
np.exp(Re_fixed * x[:, 0:1]**2 / 4)),

'compile' took 0.529124 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [2.52e-01, 6.14e-02, 5.72e-02]    [2.52e-01, 6.14e-02,
5.72e-02]    []
1000      [4.15e-06, 2.04e-07, 1.27e-07]    [4.15e-06, 2.04e-07,
1.27e-07]    []
2000      [9.16e-07, 9.16e-08, 4.54e-08]    [9.16e-07, 9.16e-08,
4.54e-08]    []
3000      [3.73e-07, 2.54e-08, 1.66e-08]    [3.73e-07, 2.54e-08,
1.66e-08]    []
4000      [1.55e-07, 1.61e-07, 5.58e-08]    [1.55e-07, 1.61e-07,
5.58e-08]    []
5000      [8.21e-08, 3.78e-09, 4.51e-09]    [8.21e-08, 3.78e-09,
4.51e-09]    []
6000      [5.61e-08, 2.60e-09, 3.95e-09]    [5.61e-08, 2.60e-09,
3.95e-09]    []
7000      [5.19e-08, 1.33e-07, 3.66e-08]    [5.19e-08, 1.33e-07,
3.66e-08]    []
8000      [3.10e-08, 1.65e-09, 4.33e-09]    [3.10e-08, 1.65e-09,
4.33e-09]    []
9000      [3.38e-08, 1.74e-08, 2.38e-08]    [3.38e-08, 1.74e-08,
2.38e-08]    []
10000     [1.71e-08, 1.37e-09, 6.24e-09]    [1.71e-08, 1.37e-09,
6.24e-09]    []
11000     [1.36e-08, 1.21e-09, 5.90e-09]    [1.36e-08, 1.21e-09,
5.90e-09]    []
12000     [4.84e-07, 1.98e-06, 1.21e-06]    [4.84e-07, 1.98e-06,
1.21e-06]    []
13000     [1.19e-06, 2.51e-07, 2.50e-07]    [1.19e-06, 2.51e-07,
2.50e-07]    []
14000     [7.49e-09, 1.13e-09, 3.36e-09]    [7.49e-09, 1.13e-09,
```

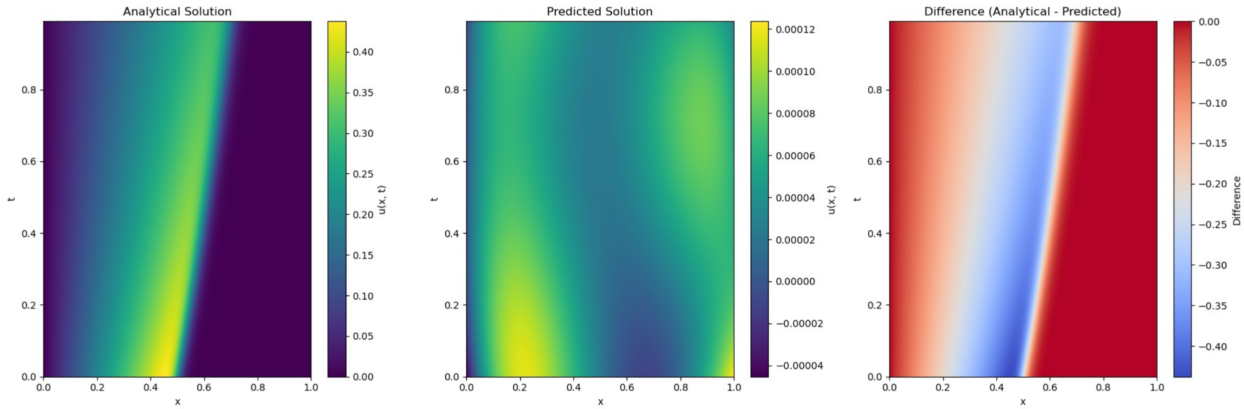
```
3.36e-09]    []
15000     [6.39e-09, 2.41e-09, 4.35e-09]    [6.39e-09, 2.41e-09,
4.35e-09]    []

Best model at step 14000:
  train loss: 1.20e-08
  test loss: 1.20e-08
  test metric: []

'train' took 40.350875 s

Compiling model...
'compile' took 0.203661 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [6.39e-09, 2.41e-09, 4.35e-09]    [6.39e-09, 2.41e-09,
4.35e-09]    []
WARNING:tensorflow:From d:\anaconda3\Lib\site-packages\deepxde\
optimizers\tensorflow_compat_v1\scipy_optimizer.py:398: The name
tf.logging.info is deprecated. Please use tf.compat.v1.logging.info
instead.

INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000000
  Number of iterations: 1
  Number of functions evaluations: 36
15017     [6.39e-09, 2.41e-09, 4.35e-09]    [6.39e-09, 2.41e-09,
4.35e-09]    []

Best model at step 14000:
  train loss: 1.20e-08
  test loss: 1.20e-08
  test metric: []

'train' took 0.380802 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```


Train loss
Test loss
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
0
2000
4000
6000
8000
10000
12000
14000
Steps


7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0.0
0.2
0.4
0.6
0.8
1.0
x_1
x_2

```
Fixed Re = 300, Mean residual: 5.804481770610437e-05
Fixed Re = 300, L2 relative error: 0.9998068515099797
```

```python
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_fixed.reshape(len(t), len(x))
y_pred_reshaped = y_pred_fixed.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                     origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Part (b): Combined Inverse-Forward Problem

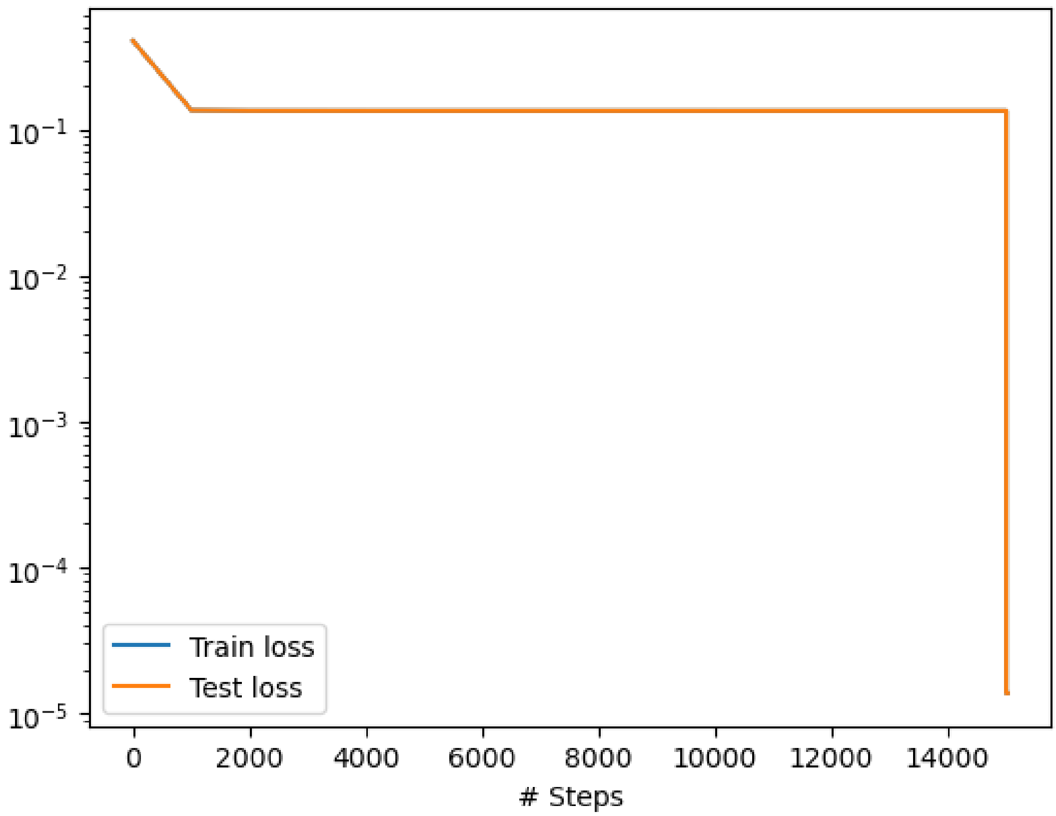
```
# Define Reynolds number as a trainable variable
Re_trainable = tf.Variable(300.0, trainable=True, dtype=tf.float32)

# Define the PDE
def pde_trainable(x, y):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 3 / Re_trainable * dy_xx

# Define the initial condition
to = tf.exp(Re_trainable / 8)
ic = dde.IC(
    geomtime,
    lambda x: x[:, 0:1] / (1 + tf.sqrt(tf.minimum(10.0, to)) *
tf.exp(tf.minimum(10.0, Re_trainable * x[:, 0:1]**2 / 4))),
    lambda _, on_initial: on_initial,
)

# Dirichlet boundary conditions
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)

# Define the dataset
data_trainable = dde.data.TimePDE(
    geomtime, pde_trainable, [bc, ic], num_domain=3000,
num_boundary=100, num_initial=200
)

# Define the neural network
net_trainable = dde.maps.FNN([2] + [40] * 5 + [1], "tanh", "Glorot
normal")

# Define custom loss function with L2 regularization
def custom_loss(y_true, y_pred):
    # Compute Mean Squared Error (MSE)
```

```
    mse = tf.reduce_mean(tf.square(y_true - y_pred))
    # Collect trainable variables
    trainable_vars = tf.compat.v1.trainable_variables()
    # Apply L2 regularization
    l2_reg = 1e-6 * tf.reduce_sum([tf.nn.l2_loss(v) for v in
trainable_vars])
    return mse + l2_reg

# Compile and train the model
model_trainable = dde.Model(data_trainable, net_trainable)
model_trainable.compile("adam", lr=1e-5, loss=custom_loss)
model_trainable.train(epochs=15000)
model_trainable.compile("L-BFGS")
losshistory_trainable, train_state_trainable = model_trainable.train()

# Save results
dde.saveplot(losshistory_trainable, train_state_trainable,
issave=True, isplot=True)

# Test and evaluate the model
X_trainable, y_true_trainable = gen_testdata()
y_pred_trainable = model_trainable.predict(X_trainable)
f_trainable = model_trainable.predict(X_trainable,
operator=pde_trainable)

# Print errors and learned Reynolds number
print("Mean residual for trainable Re:",
np.mean(np.absolute(f_trainable)))
print("L2 relative error for trainable Re:",
dde.metrics.l2_relative_error(y_true_trainable, y_pred_trainable))

with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    learned_Re = sess.run(Re_trainable)
print("Learned Reynolds number:", learned_Re)

# Save test results
np.savetxt(f"test_trainable_Re_{learned_Re:.1f}.dat",
np.hstack((X_trainable, y_true_trainable, y_pred_trainable)))
```

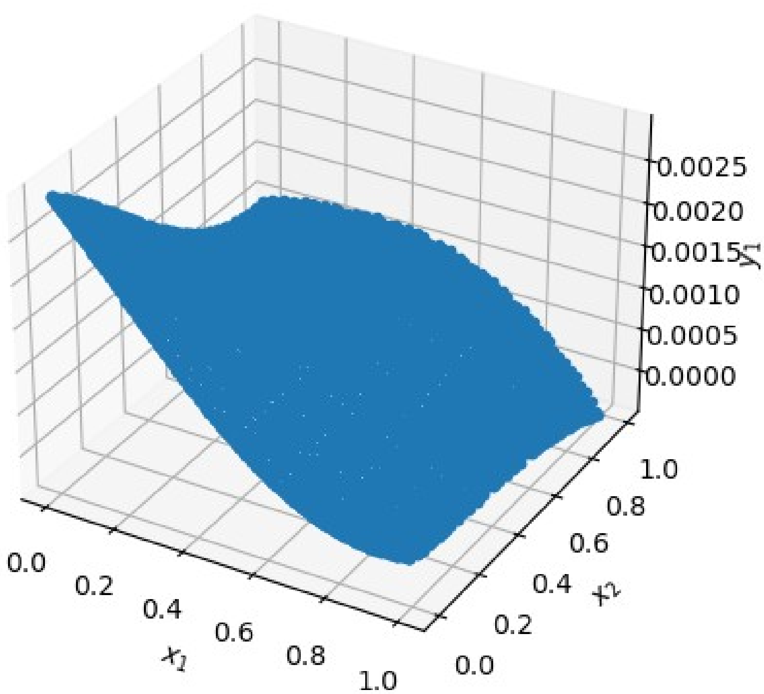
```
Compiling model...
Building feed-forward neural network...
'build' took 0.088369 s

WARNING:tensorflow:From C:\Users\jhyang\AppData\Local\Temp\
ipykernel_18744\2858174054.py:35: The name tf.trainable_variables is
deprecated. Please use tf.compat.v1.trainable_variables instead.

'compile' took 1.155185 s
```

```
Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [5.48e-02, 2.08e-01, 1.45e-01]    [5.48e-02, 2.08e-01,
1.45e-01]    []
1000      [4.53e-02, 4.59e-02, 4.52e-02]    [4.53e-02, 4.59e-02,
4.52e-02]    []
2000      [4.52e-02, 4.51e-02, 4.51e-02]    [4.52e-02, 4.51e-02,
4.51e-02]    []
3000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
4000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
5000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
6000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
7000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
8000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
9000      [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
10000     [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
11000     [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
12000     [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
13000     [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
14000     [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []
15000     [4.51e-02, 4.51e-02, 4.51e-02]    [4.51e-02, 4.51e-02,
4.51e-02]    []

Best model at step 15000:
  train loss: 1.35e-01
  test loss: 1.35e-01
  test metric: []

'train' took 117.012841 s

Compiling model...
'compile' took 0.347285 s

Training model...
```

```
Step      Train loss                      Test loss
Test metric
15000     [8.40e-07, 1.83e-06, 1.10e-05]    [8.40e-07, 1.83e-06,
1.10e-05]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000014
  Number of iterations: 1
  Number of functions evaluations: 28
15019     [8.40e-07, 1.83e-06, 1.10e-05]    [8.40e-07, 1.83e-06,
1.10e-05]    []

Best model at step 15000:
  train loss: 1.37e-05
  test loss: 1.37e-05
  test metric: []

'train' took 0.813819 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```



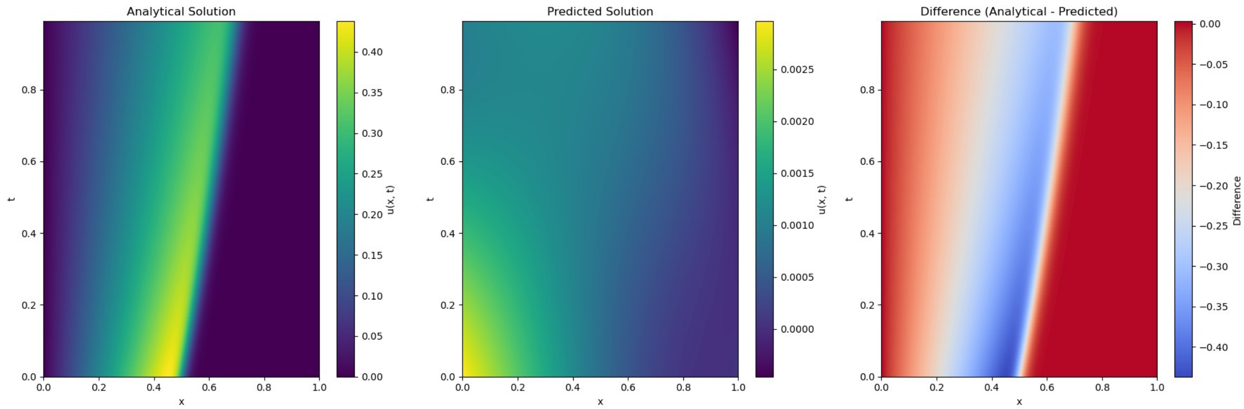
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
Train loss
Test loss
0
2000
4000
6000
8000
10000
12000
14000
Steps



0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
y_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
0.0
0.2
0.4
0.6
0.8
1.0
x_2

```
Mean residual for trainable Re: 0.0006394048
L2 relative error for trainable Re: 0.9965252271207028
Learned Reynolds number: 300.0
```

```
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_trainable.reshape(len(t), len(x))
y_pred_reshaped = y_pred_trainable.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

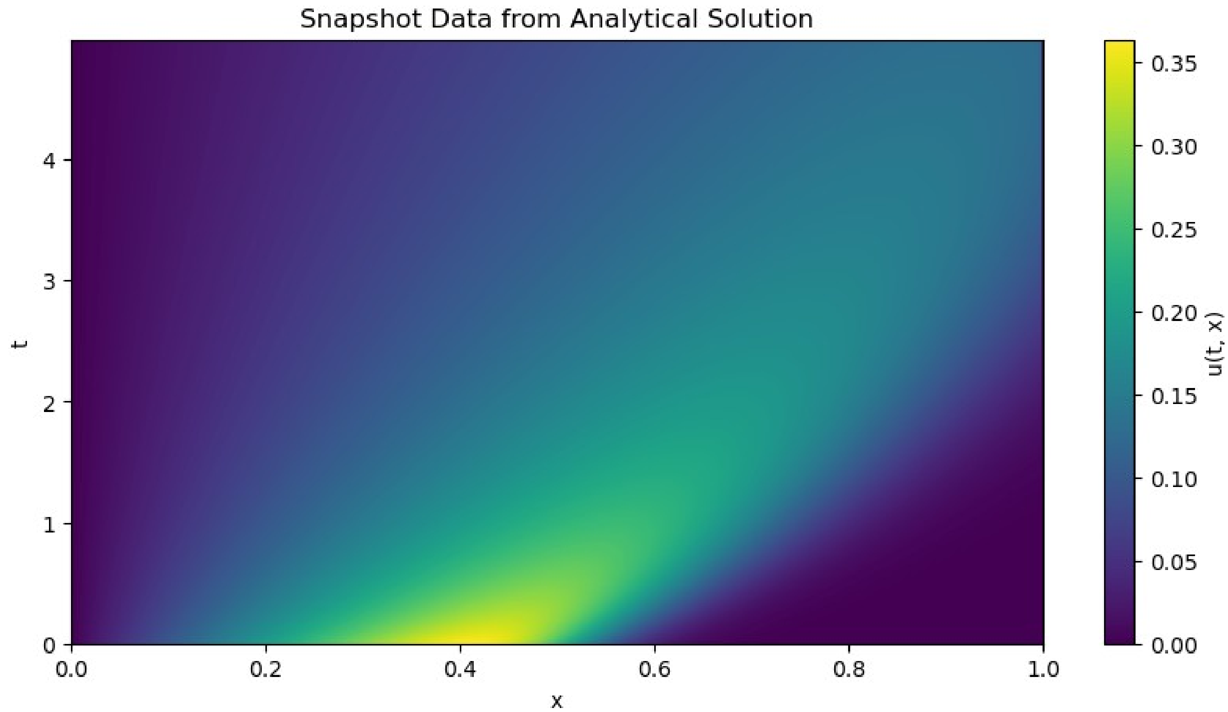

Analytical Solution
Predicted Solution
Difference (Analytical - Predicted)
u(x, t)
Difference
x
t

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import svd
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
```

Question 2.1: Generate m Snapshot Data

```python
# Parameters for snapshot generation
dt = 1e-2  # Time step
dx = 1e-3  # Spatial grid length
m = 500    # Extend snapshots to cover T = 2.0
t = np.arange(0, m * dt, dt)  # Time grid from t1 to tm
x = np.arange(0, 1 + dx, dx)  # Spatial grid

# Define the analytical solution for the 1D Burgers equation
def analytical_solution(x, t, Re_fixed):
    to = np.exp(Re_fixed / 8)  # Parameter in the equation
    u = x / (t + 1) / (1 + np.sqrt((t + 1) / to) * np.exp(Re_fixed *
x**2 / (4 * (t + 1))))
    return u

# Generate snapshot data
Re_fixed = 100  # Reynolds number for this question
usol = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x]
for ti in t])

# Explicitly enforce boundary conditions
usol[:, 0] = 0  # u(0, t) = 0
usol[:, -1] = 0  # u(1, t) = 0

# Verify the boundary conditions
assert np.allclose(usol[:, 0], 0), "Boundary condition u(0, t) = 0 not
satisfied"
assert np.allclose(usol[:, -1], 0), "Boundary condition u(1, t) = 0
not satisfied"

# Save snapshot data
np.savez("dataset/Burgers_snapshots.npz", t=t, x=x, usol=usol)
print(f"Snapshot data generated with m={m}, dt={dt}, dx={dx}.")

# Visualize snapshot data
plt.figure(figsize=(10, 5))
plt.imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],
aspect='auto', origin='lower')
plt.colorbar(label="u(t, x)")
plt.title("Snapshot Data from Analytical Solution")
plt.xlabel("x")
plt.ylabel("t")
plt.show()
```

```
Snapshot data generated with m=500, dt=0.01, dx=0.001.
```

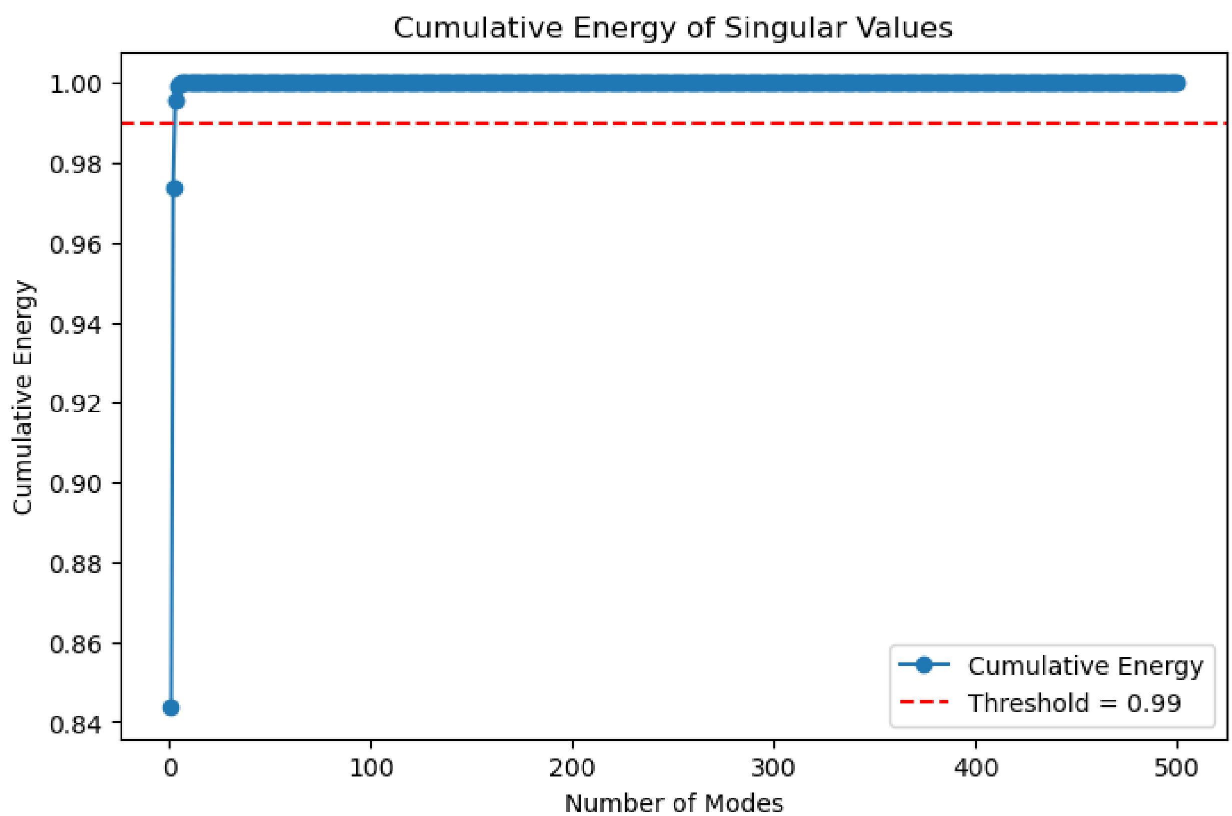


Question 2.2: Construct ROM and Determine t∗

```python
# Step 1: Perform SVD

# Load snapshot data
data = np.load("dataset/Burgers_snapshots.npz")
t = data["t"]  # Time grid
x = data["x"]  # Spatial grid
usol = data["usol"]  # Snapshot matrix

# Perform SVD on snapshot matrix A
A = usol
U, S, VT = svd(A, full_matrices=False)

# Select k modes based on cumulative energy threshold
alpha_SVD = 0.01  # Adjusted energy threshold
k = np.sum(np.cumsum(S**2) / np.sum(S**2) < 1 - alpha_SVD)
if k == 0:
    raise ValueError("No modes selected. Check the snapshot data or adjust alpha_SVD.")
print(f"Number of selected modes (k): {k}")

# Extract the first k modes
U_k = U[:, :k]  # Temporal coefficients
S_k = S[:k]     # Singular values
```

```
V_k = VT[:k, :]  # Spatial modes

# Step 2: Fit uq(ti) Using GPR

# Fit GPR models for each mode q=1,...,k
gp_models = []
for q in range(k):
    kernel = C(1.0, (1e-4, 1e4)) * RBF(1.0, (1e-4, 1e4))
    gp = GaussianProcessRegressor(kernel=kernel,
n_restarts_optimizer=10, alpha=1e-6)
    gp.fit(t.reshape(-1, 1), U_k[:, q])  # Fit GPR for temporal mode
u_q(t)
    gp_models.append(gp)

# Step 3: Predict ROM Solution for New Time Points

# Predict ROM solution u_ROM(t, x)
def predict_u_rom(t_new, x, gp_models, S_k, V_k):
    U_pred = np.zeros((len(t_new), k))
    for q, gp in enumerate(gp_models):
        U_pred[:, q] = gp.predict(t_new.reshape(-1, 1))
    A_pred = np.dot(U_pred * S_k, V_k)  # A ≈ u_ROM(t, x)
    return A_pred

# Example: Predict u_ROM(t, x) for a new time range
t_new = np.arange(0, 2.0, dt)
u_rom_pred = predict_u_rom(t_new, x, gp_models, S_k, V_k)

# Step 4: Determine Maximum Permissible Forecast Time t∗

# Compute average standard deviation σ¯(t')
def compute_avg_std(t_new, gp_models):
    std_total = np.zeros(len(t_new))
    for gp in gp_models:
        _, std = gp.predict(t_new.reshape(-1, 1), return_std=True)
        std_total += std
    return std_total / len(gp_models)

# Determine t* based on σ¯(t') ≤ σ_tolerance
sigma_tolerance = 0.01  # Allowed prediction error threshold
std_avg = compute_avg_std(t_new, gp_models)

if std_avg.size == 0:
    raise ValueError("No permissible forecast time found. Check GPR
models or sigma_tolerance.")

t_star_index = np.where(std_avg <= sigma_tolerance)[0][-1]
t_star = t_new[t_star_index]
print(f"Maximum permissible forecast time t^*: {t_star}")
```

```
Number of selected modes (k): 2
Maximum permissible forecast time t^*: 1.99

# Visualize cumulative energy of singular values
energy_cumulative = np.cumsum(S**2) / np.sum(S**2)
plt.figure(figsize=(8, 5))
plt.plot(np.arange(1, len(S) + 1), energy_cumulative, marker='o',
label="Cumulative Energy")
plt.axhline(1 - alpha_SVD, color='r', linestyle='--',
label=f"Threshold = {1 - alpha_SVD}")
plt.xlabel("Number of Modes")
plt.ylabel("Cumulative Energy")
plt.legend()
plt.title("Cumulative Energy of Singular Values")
plt.show()
```



Question 2.3: Predict Full-Order Solutions Up to t∗

```
# Use the ROM to predict up to t*
u_rom_at_t_star = predict_u_rom(np.array([t_star]), x, gp_models, S_k,
V_k)

# Save and print the solution at t^*
#np.savez(f"results/full_order_solution_at_t_star_{t_star:.2f}.npz",
```

```
t_star=t_star, u_full=u_rom_at_t_star)
print(f"Full-order solution at t^* = {t_star} is ready.")
```

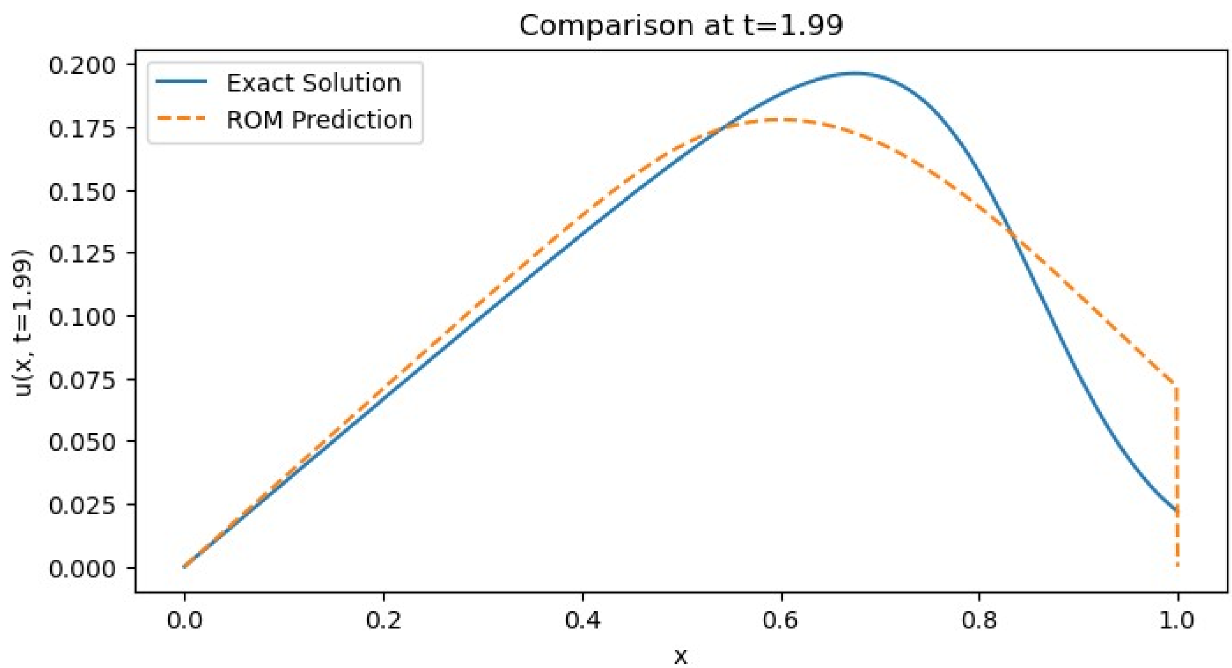
```
Full-order solution at t^* = 1.99 is ready.
```

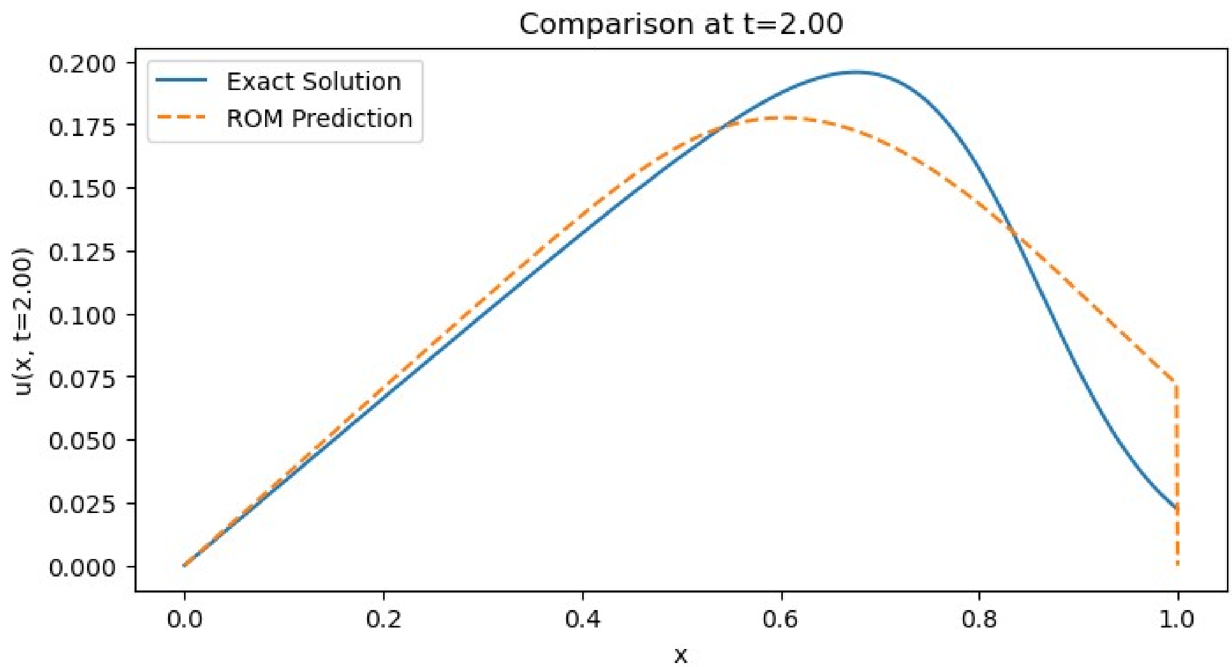
Question 2.4: Repeat Steps 1-3 Until T=2.0

```
T_target = 2.0  # Target time
t_current = t  # Current time grid
u_rom_current = usol  # Initial solution

iteration = 0
while np.max(t_current) < T_target:
    iteration += 1
    print(f"Iteration {iteration}: Current maximum time =
{np.max(t_current)}")

    # Recompute SVD and GPR models
    A = u_rom_current
    U, S, VT = svd(A, full_matrices=False)
    k = np.sum(np.cumsum(S**2) / np.sum(S**2) < 1 - alpha_SVD)
    U_k = U[:, :k]
    S_k = S[:k]
    V_k = VT[:k, :]

    gp_models = []
    for q in range(k):
        kernel = C(1.0, (1e-4, 1e4)) * RBF(1.0, (1e-4, 1e4))
        gp = GaussianProcessRegressor(kernel=kernel,
n_restarts_optimizer=10, alpha=1e-6)
        gp.fit(t_current.reshape(-1, 1), U_k[:, q])
        gp_models.append(gp)

    # Predict ROM solution for the new time range
    t_new = np.linspace(np.max(t_current), np.max(t_current) + 0.5,
50)
    u_rom_pred = predict_u_rom(t_new, x, gp_models, S_k, V_k)

    # Update current time and solution
    t_current = np.concatenate([t_current, t_new])
    u_rom_current = np.vstack([u_rom_current, u_rom_pred])

    # Check if t^* has reached the target time
    if np.max(t_current) >= T_target:
        break

print(f"Long-time prediction completed up to T = {T_target}.")
```

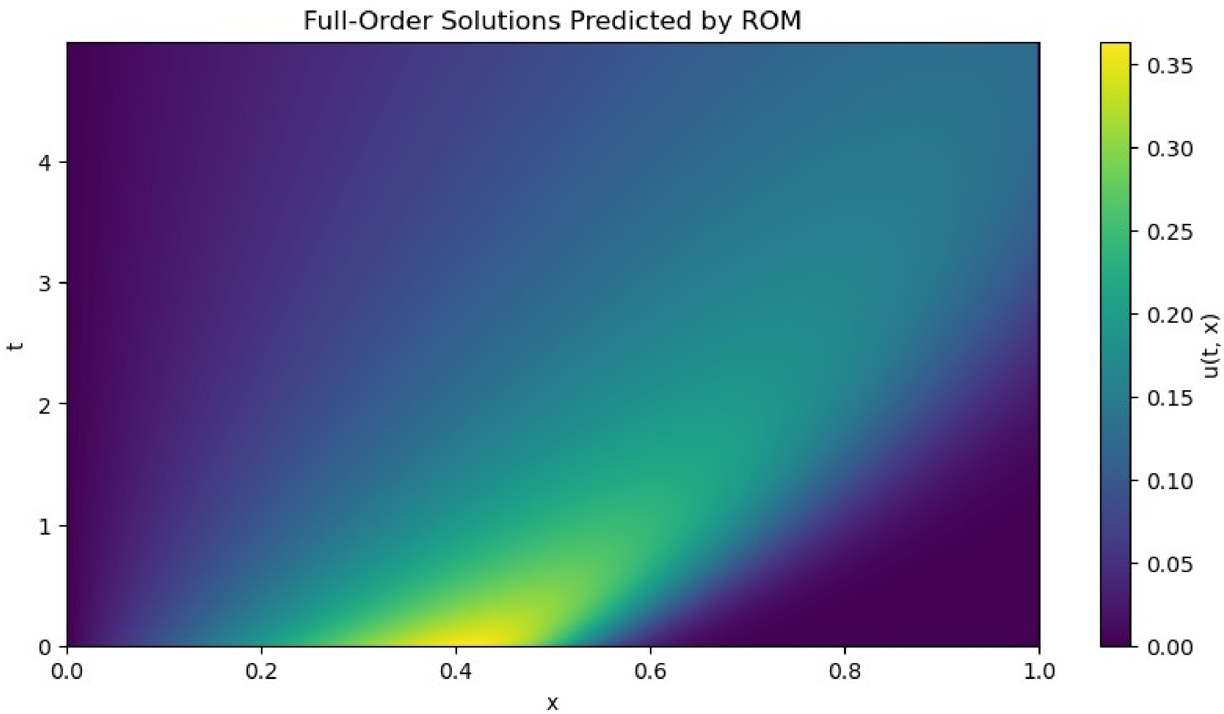
```
Long-time prediction completed up to T = 2.0.
```

```python
# Compare ROM prediction and exact solution at specific time points
for t_idx, t_val in enumerate([t_star, 2.0]):
    u_rom = predict_u_rom(np.array([t_val]), x, gp_models, S_k, V_k)
    u_exact = np.array([analytical_solution(xi, t_val, Re_fixed) for
xi in x])

    plt.figure(figsize=(8, 4))
    plt.plot(x, u_exact, label="Exact Solution")
    plt.plot(x, u_rom.flatten(), label="ROM Prediction",
linestyle="--")
    plt.xlabel("x")
    plt.ylabel(f"u(x, t={t_val:.2f})")
    plt.title(f"Comparison at t={t_val:.2f}")
    plt.legend()
    plt.show()
```






```
# Visualize full ROM prediction over time
plt.figure(figsize=(10, 5))
plt.imshow(u_rom_current, extent=[x.min(), x.max(), t_current.min(),
t_current.max()], aspect='auto', origin='lower')
plt.colorbar(label="u(t, x)")
plt.title("Full-Order Solutions Predicted by ROM")
plt.xlabel("x")
plt.ylabel("t")
plt.show()
```




```
# Generate the reference solution (true solution)
# The true solution should match the shape of the ROM solution
u_true = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x]
for ti in t_current])

# Calculate the difference between the ROM solution and the true
solution
u_diff = u_true - u_rom_current

# Create a figure with three subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 5))

# Subplot 1: Heatmap of the true solution
im1 = axs[0].imshow(u_true, extent=[x.min(), x.max(), t_current.min(),
t_current.max()],
                    aspect='auto', origin='lower', cmap='viridis')
axs[0].set_title("True Solution (u_true)")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(t, x)")

# Subplot 2: Heatmap of the ROM prediction
im2 = axs[1].imshow(u_rom_current, extent=[x.min(), x.max(),
t_current.min(), t_current.max()],
                    aspect='auto', origin='lower', cmap='viridis')
axs[1].set_title("ROM Prediction (u_rom)")
axs[1].set_xlabel("x")
```

```
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(t, x)")

# Subplot 3: Heatmap of the difference (True Solution - ROM
Prediction)
im3 = axs[2].imshow(u_diff, extent=[x.min(), x.max(), t_current.min(),
t_current.max()],
                    aspect='auto', origin='lower', cmap='coolwarm')
axs[2].set_title("Difference (u_true - u_rom)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust the layout and display the plots
plt.tight_layout()
plt.show()
```