In [4]:

```
# !pip install sbi

import torch
import numpy as np
from scipy.signal import welch
from sbi import utils as sbi_utils
from sbi.inference import SNPE
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from tensorflow.keras.models import load_model
```

In [5]:

```
#  Load Trained Generator
generator = load_model("saved_models/qpo_cgan_phy_generator.keras")

# --- Hyperparameters ---
latent_dim = 100
num_simulations = 5000
num_test_samples = 100

#  Prior: fc ∈ [0.01, 1.0], amp ∈ [0.1, 1.0]
prior = sbi_utils.BoxUniform(low=torch.tensor(
    [0.01, 0.1]), high=torch.tensor([1.0, 1.0]))

#  GAN-based simulator for SBI


def simulator(theta):
    fc, amp = theta.numpy()
    z = torch.randn((1, latent_dim))
    label = torch.tensor([[fc, amp, 1.0]], dtype=torch.float32)  # force
    generated = generator([z, label], training=False).numpy().squeeze()
    f, Pxx = welch(generated, fs=1, nperseg=256)
    return torch.tensor(Pxx, dtype=torch.float32)


# --- Generate Simulations
print("Generating synthetic QPOs for SBI...")
thetas = prior.sample((num_simulations,))
xs = torch.stack([simulator(theta) for theta in thetas])

# --- Train SBI Posterior
inference = SNPE(prior)
density_estimator = inference.append_simulations(thetas, xs).train()
posterior = inference.build_posterior(density_estimator)

#  Save Trained Posterior
torch.save(posterior, "saved_models/trained_sbi_posterior.pt")
print(" SBI Posterior trained and saved.")
```

```
Generating synthetic QPOs for SBI...
 Neural network successfully converged after 52 epochs. SBI Posterior trai
ned and saved.
```

In [ ]:

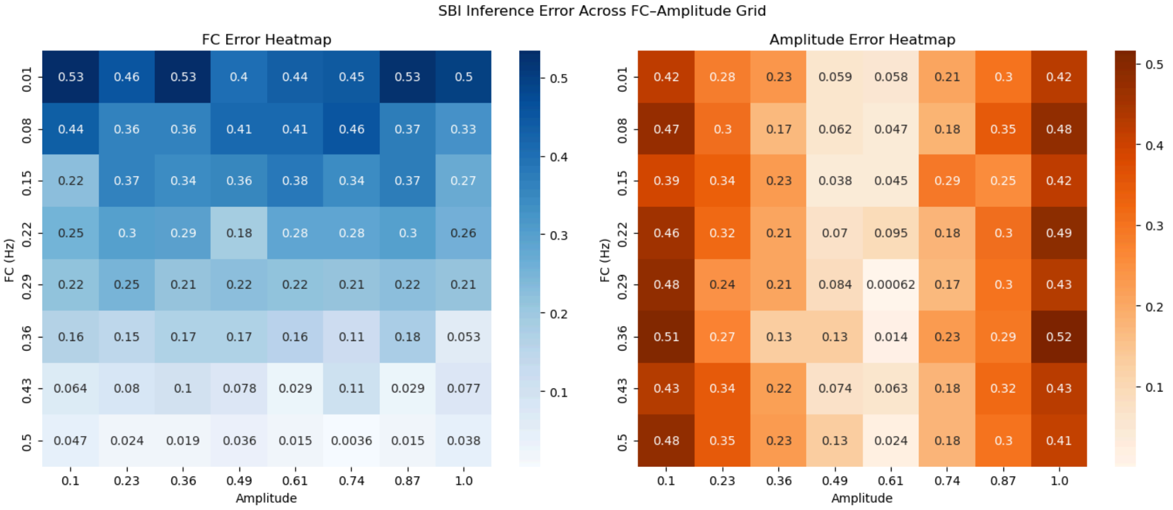
```
import numpy as np
import torch
from scipy.signal import welch
import matplotlib.pyplot as plt
```

```python
import seaborn as sns


def run_grid_sbi_evaluation(generator, posterior, latent_dim=100,
                            fc_vals=np.linspace(0.01, 0.5, 8),
                            amp_vals=np.linspace(0.1, 1.0, 8),
                            num_samples=500):
    """
    Run full-loop inference on a grid of fc/amp values and collect mean e
    Returns 2D arrays of errors.
    """

    fc_error_map = np.zeros((len(fc_vals), len(amp_vals)))
    amp_error_map = np.zeros((len(fc_vals), len(amp_vals)))

    for i, fc in enumerate(fc_vals):
        for j, amp in enumerate(amp_vals):
            # Step 1: Generate GAN sample
            label = torch.tensor([[fc, amp, 1.0]], dtype=torch.float32)
            z = torch.randn((1, latent_dim))
            signal = generator([z, label], training=False).numpy().squeez

            # Step 2: Compute PSD
            f, Pxx = welch(signal, fs=1, nperseg=256)
            x_obs = torch.tensor(Pxx, dtype=torch.float32)

            # Step 3: Run SBI inference
            samples = posterior.sample(
                (num_samples,), x=x_obs, show_progress_bars=False)
            fc_pred = samples[:, 0].mean().item()
            amp_pred = samples[:, 1].mean().item()

            # Step 4: Record absolute error
            fc_error_map[i, j] = abs(fc_pred - fc)
            amp_error_map[i, j] = abs(amp_pred - amp)

    return fc_vals, amp_vals, fc_error_map, amp_error_map


def plot_error_heatmaps(fc_vals, amp_vals, fc_errs, amp_errs):
    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    sns.heatmap(fc_errs, xticklabels=np.round(amp_vals, 2), yticklabels=n
                ax=axs[0], cmap="Blues", annot=True)
    axs[0].set_title("FC Error Heatmap")
    axs[0].set_xlabel("Amplitude")
    axs[0].set_ylabel("FC (Hz)")

    sns.heatmap(amp_errs, xticklabels=np.round(amp_vals, 2), yticklabels=
                ax=axs[1], cmap="Oranges", annot=True)
    axs[1].set_title("Amplitude Error Heatmap")
    axs[1].set_xlabel("Amplitude")
    axs[1].set_ylabel("FC (Hz)")

    plt.suptitle("SBI Inference Error Across FC-Amplitude Grid")
    plt.tight_layout()
    plt.show()
```

In [ ]:

```python
# Load posterior (already trained and saved)
posterior = torch.load("saved_models/trained_sbi_posterior.pt")
```

```
generator = load_model("saved_models/qpo_cgan_phy_generator.keras")

fc_vals, amp_vals, fc_errs, amp_errs = run_grid_sbi_evaluation(
    generator, posterior)
plot_error_heatmaps(fc_vals, amp_vals, fc_errs, amp_errs)
```



FC Error Heatmap

FC (Hz) \ Amplitude	0.1	0.23	0.36	0.49	0.61	0.74	0.87	1.0
0.01	0.53	0.46	0.53	0.4	0.44	0.45	0.53	0.5
0.08	0.44	0.36	0.36	0.41	0.41	0.46	0.37	0.33
0.15	0.22	0.37	0.34	0.36	0.38	0.34	0.37	0.27
0.22	0.25	0.3	0.29	0.18	0.28	0.28	0.3	0.26
0.29	0.22	0.25	0.21	0.22	0.22	0.21	0.22	0.21
0.36	0.16	0.15	0.17	0.17	0.16	0.11	0.18	0.053
0.43	0.064	0.08	0.1	0.078	0.029	0.11	0.029	0.077
0.5	0.047	0.024	0.019	0.036	0.015	0.0036	0.015	0.038

Amplitude Error Heatmap

FC (Hz) \ Amplitude	0.1	0.23	0.36	0.49	0.61	0.74	0.87	1.0
0.01	0.42	0.28	0.23	0.059	0.058	0.21	0.3	0.42
0.08	0.47	0.3	0.17	0.062	0.047	0.18	0.35	0.48
0.15	0.39	0.34	0.23	0.038	0.045	0.29	0.25	0.42
0.22	0.46	0.32	0.21	0.07	0.095	0.18	0.3	0.49
0.29	0.48	0.24	0.21	0.084	0.00062	0.17	0.3	0.43
0.36	0.51	0.27	0.13	0.13	0.014	0.23	0.29	0.52
0.43	0.43	0.34	0.22	0.074	0.063	0.18	0.32	0.43
0.5	0.48	0.35	0.23	0.13	0.024	0.18	0.3	0.41