In [ ]:
```
pip install scikit-optimize
```

```
Requirement already satisfied: scikit-optimize in /usr/local/lib/python3.11/dis
t-packages (0.10.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.11/dist-p
ackages (from scikit-optimize) (1.5.1)
Requirement already satisfied: pyaml>=16.9 in /usr/local/lib/python3.11/dist-pa
ckages (from scikit-optimize) (25.7.0)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.11/dist-
packages (from scikit-optimize) (2.0.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.11/dist-p
ackages (from scikit-optimize) (1.16.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.1
1/dist-packages (from scikit-optimize) (1.6.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dis
t-packages (from scikit-optimize) (25.0)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-package
s (from pyaml>=16.9->scikit-optimize) (6.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.1
1/dist-packages (from scikit-learn>=1.0.0->scikit-optimize) (3.6.0)
```

In [ ]:
```
pip install cma
```

```
Requirement already satisfied: cma in /usr/local/lib/python3.11/dist-packages
(4.2.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from cma) (2.0.2)
```

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
```

```
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
```

```
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0, 0, 0, 0.005, 0.005], sigma0=0.2, options={'seed'
    return res[0][:3], res[0][3:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(0, 200), (200, 400), (400, 600), (600, 800), (800, 1000
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)
```

```
    if not test_positions:
        raise ValueError("No positions generated.")

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

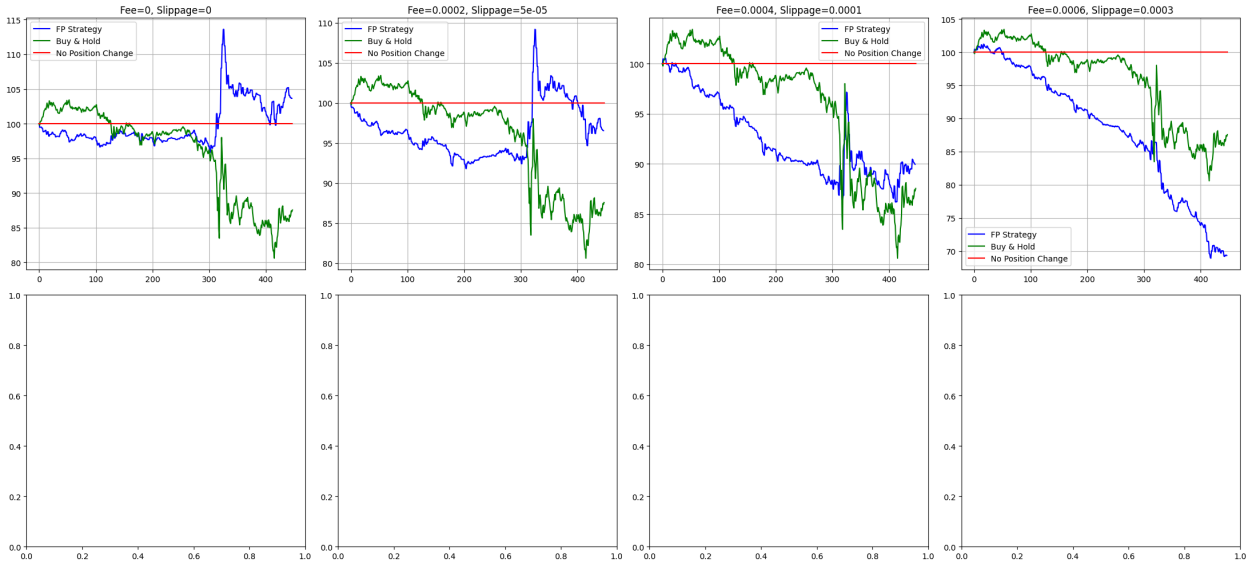
```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:23
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.464804897229344e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -8.704299852480557e-03 1.1e+00 1.85e-01  2e-01  2e-01 0:00.0
    3     24 -2.789829817449388e-02 1.3e+00 2.01e-01  2e-01  2e-01 0:00.0
   87    696 -1.241989274351664e-01 4.3e+01 4.50e-02  3e-02  5e-02 0:01.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:24 2025)
final/bestever f-value = -1.241989e-01 -1.355520e-01 after 697/517 evaluations
incumbent solution: [-1.99222787, -4.63462553, 4.714142, 3.7120206, -4.6186290
9]
std deviation: [0.03434214, 0.02912079, 0.02793754, 0.05420163, 0.04135245]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:24
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.711373459210945e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -3.332086884862889e-03 1.1e+00 1.84e-01  2e-01  2e-01 0:00.0
    3     24 -6.926994333928785e-02 1.3e+00 1.91e-01  2e-01  2e-01 0:00.1
   52    416 -1.041706507496969e-01 1.0e+01 2.60e-02  5e-03  3e-02 0:01.8
termination on tolflatfitness=1 (Tue Jul 22 12:55:26 2025)
final/bestever f-value = -1.041707e-01 -1.041707e-01 after 417/265 evaluations
incumbent solution: [ 0.44224471, -1.24016011, 0.08821176, -1.41269362, -0.6216
3969]
std deviation: [0.01005673, 0.00518235, 0.01969578, 0.02843648, 0.01026621]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:26
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.198898865044072e-02 1.0e+00 2.19e-01  2e-01  2e-01 0:00.0
    2     16 -2.576642403816651e-02 1.4e+00 2.51e-01  2e-01  3e-01 0:00.1
    3     24 -2.436582113030966e-02 1.6e+00 2.90e-01  2e-01  4e-01 0:00.1
   75    600 -4.440847651930194e-02 5.3e+01 8.29e-02  2e-02  1e-01 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 12:55:28 2025)
final/bestever f-value = -4.440848e-02 -5.548780e-02 after 601/25 evaluations
incumbent solution: [ 0.01787293, 0.45318398, 1.36691279, -0.12755442, -3.08013
679]
std deviation: [0.01647983, 0.0452785, 0.05545737, 0.06747553, 0.09879791]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:28
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.975026232439056e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -1.900450381720978e-02 1.1e+00 1.77e-01  2e-01  2e-01 0:00.0
    3     24 -1.900450381720978e-02 1.3e+00 1.79e-01  2e-01  2e-01 0:00.1
    4     32 -1.900450381720978e-02 1.4e+00 1.80e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:29 2025)
final/bestever f-value = -1.900450e-02 -1.975026e-02 after 33/1 evaluations
incumbent solution: [ 0.40247747, -0.23333494, 0.1737308, 0.03046585, -0.199680
24]
std deviation: [0.20269907, 0.16395009, 0.16098088, 0.17154833, 0.16286163]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:29
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.295013888276462e-02 1.0e+00 2.01e-01  2e-01  2e-01 0:00.0
    2     16 -3.318490399624091e-02 1.3e+00 2.15e-01  2e-01  2e-01 0:00.0
    3     24 -6.151953751820563e-02 1.4e+00 2.06e-01  2e-01  2e-01 0:00.1
```

```
   55    440 -8.178787056367892e-02 6.4e+00 3.26e-02  7e-03  2e-02 0:01.4
termination on tolflatfitness=1 (Tue Jul 22 12:55:30 2025)
final/bestever f-value = -8.178787e-02 -8.178787e-02 after 441/286 evaluations
incumbent solution: [-0.29283265, -0.5391225, 0.3767359, -0.30042985, -1.014480
95]
std deviation: [0.01343194, 0.00670051, 0.02158289, 0.0160392, 0.02018564]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:31
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.454299852480557e-03 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -8.454299852480557e-03 1.2e+00 1.74e-01  2e-01  2e-01 0:00.1
    3     24 -8.454299852480557e-03 1.3e+00 1.67e-01  1e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:32 2025)
final/bestever f-value = -8.454300e-03 -8.454300e-03 after 25/1 evaluations
incumbent solution: [ 0.27984882, -0.03323126, 0.07415682, -0.21738858, 0.01136
993]
std deviation: [0.16932493, 0.15577624, 0.14463038, 0.17319558, 0.15038317]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:32
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.611373459210944e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.1
    2     16 -3.082086884862889e-03 1.2e+00 1.73e-01  2e-01  2e-01 0:00.1
    3     24 -1.234546152553057e-02 1.4e+00 1.72e-01  2e-01  2e-01 0:00.2
   59    472 -8.177747281384445e-02 1.3e+01 2.78e-02  4e-03  2e-02 0:03.2
   82    656 -8.177747281384445e-02 3.2e+01 8.41e-03  6e-04  7e-03 0:03.8
termination on tolflatfitness=1 (Tue Jul 22 12:55:36 2025)
final/bestever f-value = -8.177747e-02 -8.177747e-02 after 657/447 evaluations
incumbent solution: [ 0.46150839, -0.96810384, 0.20844716, -0.83852865, -0.4902
4257]
std deviation: [0.00376821, 0.00062539, 0.00422667, 0.00657756, 0.00284819]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:36
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.498898865044071e-02 1.0e+00 2.16e-01  2e-01  2e-01 0:00.0
    2     16 -1.164104202412147e-02 1.4e+00 1.94e-01  2e-01  2e-01 0:00.1
    3     24 -1.995629340848334e-02 1.4e+00 1.77e-01  1e-01  2e-01 0:00.1
   99    792 -4.554704066599657e-02 1.2e+02 3.16e-02  2e-03  8e-02 0:03.1
  100    800 -4.554704066599657e-02 1.3e+02 2.82e-02  2e-03  7e-02 0:03.1
  113    904 -4.554704066599657e-02 2.0e+02 1.63e-02  1e-03  3e-02 0:03.7
termination on tolflatfitness=1 (Tue Jul 22 12:55:40 2025)
final/bestever f-value = -4.554704e-02 -4.744579e-02 after 905/473 evaluations
incumbent solution: [-0.18830619, 0.26309094, 1.49673685, -0.27805253, -0.91405
347]
std deviation: [0.00380955, 0.00101137, 0.03419716, 0.00763735, 0.00205546]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:40
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.900026232439056e-02 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -2.413079617814715e-02 1.2e+00 1.66e-01  2e-01  2e-01 0:00.1
    3     24 -1.875450381720978e-02 1.3e+00 1.56e-01  1e-01  2e-01 0:00.2
   79    632 -4.251653110946021e-02 1.5e+01 8.03e-03  1e-03  4e-03 0:02.5
termination on tolflatfitness=1 (Tue Jul 22 12:55:43 2025)
final/bestever f-value = -4.251653e-02 -4.251653e-02 after 633/478 evaluations
incumbent solution: [ 0.15655977, -0.5981109, 0.3897078, 0.2496568, -0.2869851
```

```
2]
std deviation: [0.00145686, 0.00352834, 0.00271187, 0.00229228, 0.00408981]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:43
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.970013888276461e-02 1.0e+00 2.01e-01  2e-01  2e-01 0:00.1
    2     16 -2.468490399624090e-02 1.3e+00 2.10e-01  2e-01  2e-01 0:00.1
    3     24 -3.963039609464730e-02 1.3e+00 2.12e-01  2e-01  2e-01 0:00.1
   75    600 -6.924974847026139e-02 5.8e+01 1.01e-01  2e-02  1e-01 0:01.8
termination on tolflatfitness=1 (Tue Jul 22 12:55:45 2025)
final/bestever f-value = -6.924975e-02 -6.924975e-02 after 601/341 evaluations
incumbent solution: [-1.85533725, 0.25955608, -4.5666131, 0.51747564, -2.428344
43]
std deviation: [0.04376201, 0.02036839, 0.11378387, 0.0851547, 0.04067176]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:45
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.204299852480557e-03 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -8.204299852480557e-03 1.2e+00 1.77e-01  2e-01  2e-01 0:00.0
    3     24 -8.204299852480557e-03 1.3e+00 1.83e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:46 2025)
final/bestever f-value = -8.204300e-03 -8.204300e-03 after 25/1 evaluations
incumbent solution: [ 0.45241067, -0.04153368, 0.08403166, 0.02250092, -0.21095
329]
std deviation: [0.20019771, 0.16794812, 0.17629349, 0.16040983, 0.17332252]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:46
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.511373459210944e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -2.832086884862889e-03 1.2e+00 1.73e-01  2e-01  2e-01 0:00.0
    3     24 -9.095461525530563e-03 1.4e+00 1.72e-01  2e-01  2e-01 0:00.1
   87    696 -5.582793891341812e-02 2.2e+01 8.58e-03  2e-03  5e-03 0:01.8
termination on tolflatfitness=1 (Tue Jul 22 12:55:48 2025)
final/bestever f-value = -5.582794e-02 -5.840618e-02 after 697/188 evaluations
incumbent solution: [ 0.17837965, -0.41638958, -0.3904228, -0.31978753, -0.1287
2528]
std deviation: [0.00262164, 0.00189651, 0.00526527, 0.00392836, 0.00264642]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:48
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.109475008657861e-03 1.0e+00 2.08e-01  2e-01  2e-01 0:00.0
    2     16 -5.316849337294843e-03 1.3e+00 1.91e-01  2e-01  2e-01 0:00.0
    3     24 -4.515463753653944e-03 1.4e+00 1.98e-01  2e-01  2e-01 0:00.0
   75    600 -3.272121800420205e-02 3.0e+01 1.94e-02  1e-03  2e-02 0:00.9
termination on tolflatfitness=1 (Tue Jul 22 12:55:49 2025)
final/bestever f-value = -3.272122e-02 -3.272122e-02 after 601/409 evaluations
incumbent solution: [-0.25626872, -0.14144047, 1.29481517, -1.45193279, 0.50407
142]
std deviation: [0.00501756, 0.00133607, 0.01610816, 0.01338963, 0.01464411]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.850450381720978e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     16 -1.850450381720978e-02 1.3e+00 1.87e-01  2e-01  2e-01 0:00.0
```

```
    3     24 -2.109863132623694e-02 1.4e+00 2.00e-01  2e-01  2e-01 0:00.0
   32    256 -3.111386417315695e-02 5.9e+00 1.39e-01  6e-02  2e-01 0:00.4
termination on tolflatfitness=1 (Tue Jul 22 12:55:49 2025)
final/bestever f-value = -3.111386e-02 -3.111386e-02 after 257/167 evaluations
incumbent solution: [-0.32199169, 0.64702897, 0.40598319, -0.05191104, -0.72917
21, ]
std deviation: [0.17672244, 0.09311075, 0.17607605, 0.10280448, 0.06286937]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.883729412123132e-02 1.0e+00 2.15e-01  2e-01  2e-01 0:00.0
    2     16 -4.230462758256523e-02 1.3e+00 2.26e-01  2e-01  3e-01 0:00.0
    3     24 -2.313915190566413e-02 1.5e+00 2.28e-01  2e-01  3e-01 0:00.0
   88    704 -6.758328197294794e-02 2.8e+01 1.39e-02  2e-03  1e-02 0:01.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:50 2025)
final/bestever f-value = -6.758328e-02 -6.758328e-02 after 705/529 evaluations
incumbent solution: [-0.03758348, -0.07625372, 0.21101301, -1.43013622, -0.2540
8684]
std deviation: [0.00161404, 0.00150378, 0.00826399, 0.01463316, 0.00710733]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:51
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.804299852480559e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     16 -7.804299852480559e-03 1.3e+00 1.69e-01  2e-01  2e-01 0:00.0
    3     24 -2.511255737367957e-02 1.4e+00 1.60e-01  1e-01  2e-01 0:00.0
  100    800 -7.357513342989859e-02 1.0e+02 1.66e-02  2e-03  3e-02 0:01.5
  115    920 -7.357513342989859e-02 2.1e+02 1.44e-02  1e-03  2e-02 0:01.8
termination on tolflatfitness=1 (Tue Jul 22 12:55:53 2025)
final/bestever f-value = -7.357513e-02 -8.461228e-02 after 921/372 evaluations
incumbent solution: [ 0.04901826, 0.08215023, 0.79026269, 0.41415929, -0.594695
29]
std deviation: [0.00145389, 0.00443223, 0.02236151, 0.01046281, 0.01220298]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:53
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.432086884862890e-03 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -2.432086884862890e-03 1.2e+00 1.80e-01  2e-01  2e-01 0:00.0
    3     24 -2.432086884862890e-03 1.3e+00 1.87e-01  2e-01  2e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:55:53 2025)
final/bestever f-value = -2.432087e-03 -2.432087e-03 after 25/1 evaluations
incumbent solution: [ 0.40938793, -0.27181192, 0.14710618, -0.00500089, 0.02329
407]
std deviation: [0.20966991, 0.18240957, 0.17483103, 0.16497972, 0.1679008, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:53
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.094750086578715e-04 1.0e+00 1.92e-01  2e-01  2e-01 0:00.0
    2     16 -1.543671968409462e-02 1.2e+00 1.62e-01  1e-01  2e-01 0:00.0
    3     24 -1.595017937592808e-02 1.3e+00 1.43e-01  1e-01  1e-01 0:00.0
   33    264 -1.792765396737256e-02 5.6e+00 3.93e-02  1e-02  5e-02 0:00.4
termination on tolflatfitness=1 (Tue Jul 22 12:55:53 2025)
final/bestever f-value = -1.792765e-02 -3.339133e-02 after 265/41 evaluations
incumbent solution: [-0.07021212, -0.02600852, -0.07553732, -0.50795026, 0.4022
5234]
```

```
std deviation: [0.01124232, 0.01175073, 0.03181492, 0.0323114, 0.04674855]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:53
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.810450381720978e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     16 -1.810450381720978e-02 1.3e+00 1.87e-01  2e-01  2e-01 0:00.0
    3     24 -1.810450381720978e-02 1.4e+00 1.95e-01  2e-01  2e-01 0:00.0
    5     40 -1.810450381720978e-02 1.8e+00 2.17e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:53 2025)
final/bestever f-value = -1.810450e-02 -1.810450e-02 after 41/5 evaluations
incumbent solution: [ 0.23231308, 0.35510278, 0.42269589, 0.01909996, -0.065762
4, ]
std deviation: [0.20208626, 0.193526, 0.22811301, 0.20275525, 0.21493218]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:53
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.763729412123134e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.0
    2     16 -2.060988888834994e-02 1.3e+00 2.35e-01  2e-01  3e-01 0:00.0
    3     24 -1.833065634044181e-02 1.5e+00 2.55e-01  2e-01  3e-01 0:00.0
  100    800 -5.952245126144085e-02 9.3e+00 1.89e-02  2e-03  1e-02 0:01.2
  108    864 -5.952245126144085e-02 1.1e+01 1.06e-02  8e-04  6e-03 0:01.3
termination on tolflatfitness=1 (Tue Jul 22 12:55:55 2025)
final/bestever f-value = -5.952245e-02 -5.952245e-02 after 865/681 evaluations
incumbent solution: [ 0.00682962, 0.09040245, 0.36555167, -0.33692608, -0.74418
145]
std deviation: [0.00084261, 0.00161976, 0.00498226, 0.0059681, 0.00218217]
```



Final Portfolio Values and Returns for Different Fee/Slippage Configurations:

Fee	Slippage	FP Strategy ($)	FP Return (%)	Buy & Hold ($)	Buy & Hold Return (%)	NPC ($)	NPC Return (%)
0.0000	0.00000	103.65	3.65	87.56	-12.44	100.0	0.0
0.0002	0.00005	96.53	-3.47	87.56	-12.44	100.0	0.0
0.0004	0.00010	89.97	-10.03	87.56	-12.44	100.0	0.0
0.0006	0.00030	69.34	-30.66	87.56	-12.44	100.0	0.0

In [ ]:

```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_1min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
```

```
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0, 0, 0, 0.005, 0.005], sigma0=0.2, options={'seed'
    return res[0][:3], res[0][3:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(0, 500), (500, 1000), (1000, 1500), (1500, 2000), (2000
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []
```

```
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:en
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        raise ValueError("No positions generated.")

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
```

```
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:27
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.616068731822169e-02 1.0e+00 2.09e-01  2e-01  2e-01 0:00.0
    2     16 -4.398112214579994e-02 1.3e+00 2.58e-01  2e-01  3e-01 0:00.1
    3     24 -6.771950453091571e-02 1.4e+00 2.77e-01  2e-01  3e-01 0:00.1
   36    288 -7.779440765474735e-02 8.8e+00 6.93e-02  3e-02  6e-02 0:02.3
termination on tolflatfitness=1 (Tue Jul 22 12:56:29 2025)
final/bestever f-value = -7.779441e-02 -7.779441e-02 after 289/157 evaluations
incumbent solution: [ 0.28629425, 0.54852097, 2.29569573, -0.71813066, -2.44899
267]
std deviation: [0.0268, 0.03986707, 0.06137103, 0.06094294, 0.06214788]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:29
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.791396067507966e-02 1.0e+00 1.97e-01  2e-01  2e-01 0:00.0
    2     16 -5.335942222108192e-02 1.2e+00 2.56e-01  2e-01  3e-01 0:00.0
    3     24 -6.094092688053845e-02 1.4e+00 2.92e-01  3e-01  3e-01 0:00.1
   51    408 -6.777791157520571e-02 7.7e+00 4.94e-02  1e-02  4e-02 0:03.1
  100    800 -6.954175858044984e-02 4.6e+01 4.40e-02  3e-03  4e-02 0:06.0
  102    816 -6.954175858044984e-02 5.6e+01 3.26e-02  2e-03  3e-02 0:06.2
termination on tolflatfitness=1 (Tue Jul 22 12:56:36 2025)
final/bestever f-value = -6.954176e-02 -6.954176e-02 after 817/705 evaluations
incumbent solution: [ 0.39884495, 0.0964672, -0.64428261, -1.33143205, -1.22984
975]
std deviation: [0.00714983, 0.00221396, 0.02890299, 0.03319181, 0.00361231]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:36
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.206821202756411e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.1
    2     16 -1.206821202756411e-02 1.3e+00 1.76e-01  2e-01  2e-01 0:00.1
    3     24 -1.758292140394432e-02 1.4e+00 1.64e-01  1e-01  2e-01 0:00.2
   84    672 -2.889107499229740e-02 1.1e+01 7.89e-03  1e-03  4e-03 0:03.2
   91    728 -2.889107499229740e-02 1.2e+01 5.22e-03  6e-04  3e-03 0:03.5
termination on tolflatfitness=1 (Tue Jul 22 12:56:40 2025)
final/bestever f-value = -2.889107e-02 -4.066789e-02 after 729/335 evaluations
incumbent solution: [ 0.20426027, 0.01344993, 0.2290302, 0.60912319, -0.522484
1, ]
std deviation: [0.00166922, 0.00057404, 0.0015884, 0.0025016, 0.00130399]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:40
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.277392105789149e-03 1.0e+00 1.97e-01  2e-01  2e-01 0:00.1
    2     16 -1.285533777154985e-03 1.4e+00 1.78e-01  2e-01  2e-01 0:00.2
    3     24 -1.285533777154985e-03 1.5e+00 1.69e-01  2e-01  2e-01 0:00.3
```

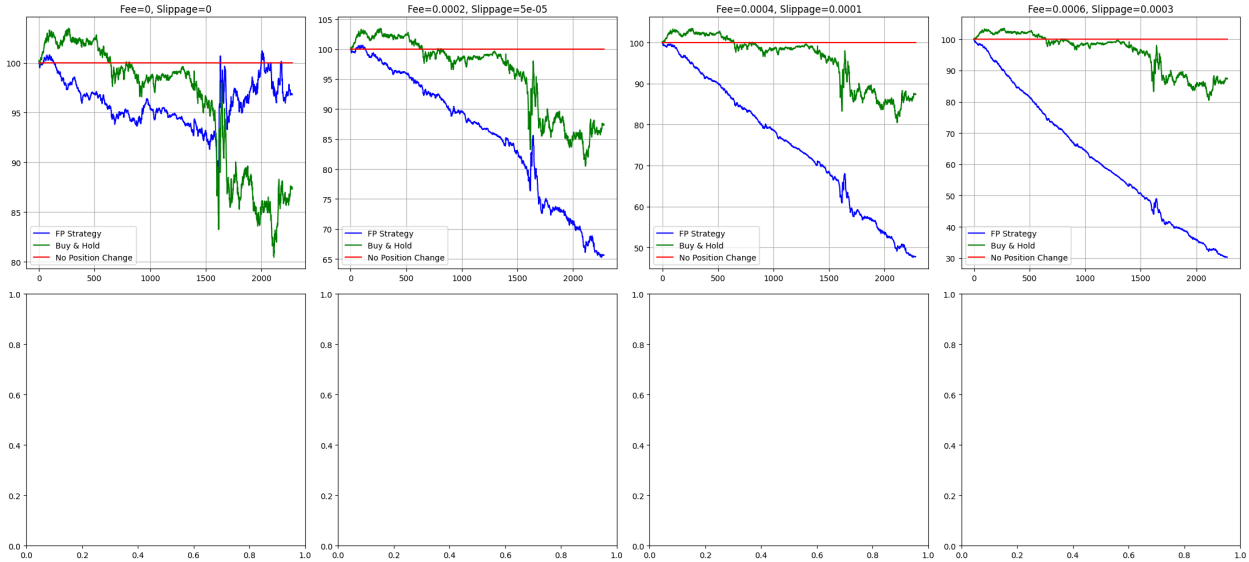
```
/tmp/ipython-input-4-3667644117.py:53: RuntimeWarning: overflow encountered in
scalar multiply
  mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
/tmp/ipython-input-4-3667644117.py:55: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-4-3667644117.py:55: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-4-3667644117.py:54: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
/tmp/ipython-input-4-3667644117.py:55: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
```

```
   44    352 -4.201423190975959e-02 1.6e+01 3.58e-01  3e-01  7e-01 0:03.3
   64    512 -4.201423190975959e-02 3.7e+01 2.08e-01  9e-02  3e-01 0:04.2
termination on tolflatfitness=1 (Tue Jul 22 12:56:44 2025)
final/bestever f-value = -4.201423e-02 -4.826053e-02 after 513/211 evaluations
incumbent solution: [-2.86663336, -5.75727899, -6.39106321, -7.52127737, -2.434
99155]
std deviation: [0.15274799, 0.179288, 0.31962353, 0.31862332, 0.09337194]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:44
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.655006917827342e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -5.958556132430348e-03 1.2e+00 1.73e-01  2e-01  2e-01 0:00.0
    3     24 -7.345683210574716e-03 1.4e+00 1.72e-01  2e-01  2e-01 0:00.1
   67    536 -4.427315150760541e-02 2.1e+01 4.79e-03  5e-04  4e-03 0:03.1
   73    584 -4.427315150760541e-02 2.2e+01 3.59e-03  5e-04  3e-03 0:03.4
termination on tolflatfitness=1 (Tue Jul 22 12:56:48 2025)
final/bestever f-value = -4.427315e-02 -4.518149e-02 after 585/273 evaluations
incumbent solution: [ 0.07655201, -0.29567046, 0.71186843, -0.63834832, -0.2638
7664]
std deviation: [0.00045588, 0.00085045, 0.0018156, 0.00279836, 0.00141694]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:50
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.886804717701203e-03 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -9.886804717701203e-03 1.2e+00 1.73e-01  1e-01  2e-01 0:00.1
    3     24 -9.886804717701203e-03 1.4e+00 1.65e-01  1e-01  2e-01 0:00.1
   66    528 -7.554849453794107e-02 9.2e+00 9.15e-03  9e-04  6e-03 0:02.7
termination on tolflatfitness=1 (Tue Jul 22 12:56:53 2025)
final/bestever f-value = -7.554849e-02 -7.554849e-02 after 529/394 evaluations
incumbent solution: [-0.04744755, -0.40370469, -0.82143229, 0.73028004, -0.6361
2148]
std deviation: [0.00093722, 0.00202938, 0.00446868, 0.00588414, 0.00567269]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:53
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.366396067507964e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.1
    2     16 -1.042886485368647e-02 1.2e+00 1.81e-01  2e-01  2e-01 0:00.2
    3     24 -1.243952350143210e-02 1.3e+00 1.82e-01  2e-01  2e-01 0:00.3
   48    384 -5.208785417677603e-02 1.8e+01 6.00e-02  1e-02  9e-02 0:03.3
  100    800 -5.765129359292848e-02 1.2e+02 1.06e-02  5e-04  2e-02 0:04.7
  107    856 -5.765129359292848e-02 1.6e+02 9.69e-03  5e-04  2e-02 0:04.9
termination on tolflatfitness=1 (Tue Jul 22 12:56:58 2025)
final/bestever f-value = -5.765129e-02 -5.772972e-02 after 857/677 evaluations
incumbent solution: [ 0.30909636, -0.22692035, 0.00787635, -1.41361619, 0.59338
554]
std deviation: [0.00280211, 0.00053062, 0.00290607, 0.015271, 0.00467715]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:58
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.181821202756411e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.181821202756411e-02 1.3e+00 1.73e-01  2e-01  2e-01 0:00.0
    3     24 -1.298773088572763e-02 1.4e+00 1.86e-01  2e-01  2e-01 0:00.1
   42    336 -2.404048810036286e-02 5.9e+00 2.44e-02  5e-03  2e-02 0:01.1
termination on tolflatfitness=1 (Tue Jul 22 12:56:59 2025)
```

```
final/bestever f-value = -2.404049e-02 -3.259559e-02 after 337/74 evaluations
incumbent solution: [ 0.53627763, -0.10348436, 0.31445868, 0.46617822, -0.50127
736]
std deviation: [0.01725224, 0.00703202, 0.02099081, 0.02158347, 0.00494387]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:59
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.035533777154984e-03 1.0e+00 1.65e-01  1e-01  2e-01 0:00.0
    2     16 -1.035533777154984e-03 1.3e+00 1.60e-01  1e-01  2e-01 0:00.1
    3     24 -1.035533777154984e-03 1.4e+00 1.58e-01  1e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:56:59 2025)
final/bestever f-value = -1.035534e-03 -1.035534e-03 after 25/1 evaluations
incumbent solution: [ 0.19813366, -0.01559226, 0.30508992, 0.08459609, -0.02798
847]
std deviation: [0.14493461, 0.12978324, 0.17151763, 0.15783279, 0.14297522]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:56:59
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.708556132430348e-03 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -1.317325925833016e-02 1.3e+00 2.17e-01  2e-01  2e-01 0:00.1
    3     24 -1.317325925833016e-02 1.4e+00 2.25e-01  2e-01  2e-01 0:00.1
   46    368 -1.919075110227303e-02 1.1e+01 3.87e-02  1e-02  5e-02 0:01.3
termination on tolflatfitness=1 (Tue Jul 22 12:57:01 2025)
final/bestever f-value = -1.919075e-02 -1.968237e-02 after 369/35 evaluations
incumbent solution: [ 0.29680205, 0.58990666, -0.14257105, 0.24918048, -0.50605
781]
std deviation: [0.01448755, 0.0444356, 0.02837388, 0.04677898, 0.01530736]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:03
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.636804717701203e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -9.815507672116447e-03 1.3e+00 1.48e-01  1e-01  1e-01 0:00.1
    3     24 -1.672310792612683e-02 1.3e+00 1.42e-01  1e-01  1e-01 0:00.1
  100    800 -3.742956702892111e-02 1.9e+01 1.08e-02  3e-03  5e-03 0:02.7
  110    880 -3.742956702892111e-02 2.7e+01 5.63e-03  1e-03  3e-03 0:03.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:06 2025)
final/bestever f-value = -3.742957e-02 -3.914957e-02 after 881/570 evaluations
incumbent solution: [ 0.06328999, -0.23513164, 0.09546125, -0.16129389, 0.68185
138]
std deviation: [0.00105623, 0.00275921, 0.00145439, 0.00188115, 0.00205897]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:06
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -6.458383374257387e-03 1.0e+00 1.68e-01  2e-01  2e-01 0:00.0
    2     16 -6.458383374257387e-03 1.3e+00 1.64e-01  1e-01  2e-01 0:00.1
    3     24 -1.486134467861316e-02 1.4e+00 1.63e-01  1e-01  2e-01 0:00.1
   34    272 -4.204883714009625e-02 7.3e+00 1.79e-01  8e-02  3e-01 0:03.2
   81    648 -6.162854365461243e-02 1.6e+01 1.40e-02  2e-03  1e-02 0:05.8
termination on tolflatfitness=1 (Tue Jul 22 12:57:12 2025)
final/bestever f-value = -6.162854e-02 -6.400895e-02 after 649/504 evaluations
incumbent solution: [ 0.30787585, 0.12365506, 0.02685074, -0.06023349, -0.67735
272]
std deviation: [0.00593493, 0.00161494, 0.0103436, 0.0068279, 0.00180049]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:12
```

```
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.156821202756411e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.156821202756411e-02 1.3e+00 1.73e-01  2e-01  2e-01 0:00.0
    3     24 -1.223773088572763e-02 1.4e+00 1.86e-01  2e-01  2e-01 0:00.1
   47    376 -2.279048810036285e-02 6.1e+00 4.04e-02  1e-02  4e-02 0:01.2
termination on tolflatfitness=1 (Tue Jul 22 12:57:14 2025)
final/bestever f-value = -2.279049e-02 -3.134559e-02 after 377/74 evaluations
incumbent solution: [ 0.63264472, -0.06727223, 0.27746046, 0.59180877, -0.47568
998]
std deviation: [0.02216537, 0.01049913, 0.03923635, 0.03821325, 0.01163256]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:14
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.855337771549842e-04 1.0e+00 1.65e-01  1e-01  2e-01 0:00.0
    2     16 -7.855337771549842e-04 1.3e+00 1.60e-01  1e-01  2e-01 0:00.1
    3     24 -7.855337771549842e-04 1.4e+00 1.46e-01  1e-01  1e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:14 2025)
final/bestever f-value = -7.855338e-04 -7.855338e-04 after 25/1 evaluations
incumbent solution: [ 0.09476296, 0.10787822, 0.10665893, 0.20514072, -0.044182
17]
std deviation: [0.1337496, 0.12517366, 0.13613093, 0.14766964, 0.13583423]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:14
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.458556132430348e-03 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -1.242325925833015e-02 1.3e+00 2.17e-01  2e-01  2e-01 0:00.1
    3     24 -1.242325925833015e-02 1.4e+00 2.25e-01  2e-01  2e-01 0:00.1
   11     88 -5.458556132430348e-03 2.1e+00 1.59e-01  1e-01  2e-01 0:00.3
termination on tolflatfitness=1 (Tue Jul 22 12:57:14 2025)
final/bestever f-value = -5.458556e-03 -1.918237e-02 after 89/35 evaluations
incumbent solution: [ 0.45981657, 0.01454099, 0.32326371, 0.25907443, -0.115593
17]
std deviation: [0.11800272, 0.15573268, 0.13915841, 0.1449552, 0.13955298]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:17
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.236804717701205e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.371012284855390e-02 1.3e+00 1.97e-01  2e-01  2e-01 0:00.1
    3     24 -9.236804717701205e-03 1.6e+00 1.90e-01  2e-01  2e-01 0:00.1
    9     72 -9.236804717701205e-03 2.1e+00 1.78e-01  2e-01  2e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:57:17 2025)
final/bestever f-value = -9.236805e-03 -1.371012e-02 after 73/15 evaluations
incumbent solution: [ 0.36066038, 0.13147896, 0.29868935, 0.41197098, -0.039726
75]
std deviation: [0.16007213, 0.15286801, 0.19816154, 0.20123897, 0.15126592]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:17
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -6.058383374257388e-03 1.0e+00 1.68e-01  2e-01  2e-01 0:00.0
    2     16 -6.058383374257388e-03 1.3e+00 1.64e-01  1e-01  2e-01 0:00.1
    3     24 -1.166134467861318e-02 1.4e+00 1.57e-01  1e-01  2e-01 0:00.1
   71    568 -3.489766514158930e-02 5.4e+01 1.12e-02  2e-03  1e-02 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:19 2025)
```

```
final/bestever f-value = -3.489767e-02 -3.489767e-02 after 569/411 evaluations
incumbent solution: [ 0.1349776, -0.16972313, 0.89268391, 0.73714431, -0.556426
67]
std deviation: [0.00248552, 0.00233622, 0.00711815, 0.01409195, 0.01081702]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:19
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.116821202756411e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.116821202756411e-02 1.3e+00 1.73e-01  2e-01  2e-01 0:00.1
    3     24 -1.116821202756411e-02 1.4e+00 1.76e-01  1e-01  2e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:57:19 2025)
final/bestever f-value = -1.116821e-02 -1.116821e-02 after 25/1 evaluations
incumbent solution: [ 0.30705491, -0.22270342, 0.22391874, 0.08426072, 0.013667
98]
std deviation: [0.18713329, 0.16852188, 0.17640361, 0.14884138, 0.15612921]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:19
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.855337771549866e-04 1.0e+00 1.65e-01  1e-01  2e-01 0:00.0
    2     16 -3.855337771549866e-04 1.3e+00 1.60e-01  1e-01  2e-01 0:00.1
    3     24 -3.855337771549866e-04 1.4e+00 1.46e-01  1e-01  1e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:19 2025)
final/bestever f-value = -3.855338e-04 -3.855338e-04 after 25/1 evaluations
incumbent solution: [ 0.09476296, 0.10787822, 0.10665893, 0.20514072, -0.044182
17]
std deviation: [0.1337496, 0.12517366, 0.13613093, 0.14766964, 0.13583423]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:19
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.058556132430350e-03 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -5.058556132430350e-03 1.3e+00 1.47e-01  1e-01  1e-01 0:00.1
    3     24 -5.058556132430350e-03 1.4e+00 1.35e-01  1e-01  1e-01 0:00.1
    4     32 -5.058556132430350e-03 1.4e+00 1.37e-01  1e-01  1e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:19 2025)
final/bestever f-value = -5.058556e-03 -5.058556e-03 after 33/1 evaluations
incumbent solution: [ 0.11500281, 0.22199494, 0.13215443, -0.05686984, -0.13797
148]
std deviation: [0.12778623, 0.11228746, 0.14503489, 0.12102428, 0.12399101]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            96.86          -3.14           87.38
-12.62    100.0             0.0
0.0002   0.00005            65.65         -34.35           87.38
-12.62    100.0             0.0
0.0004   0.00010            47.78         -52.22           87.38
-12.62    100.0             0.0
0.0006   0.00030            30.28         -69.72           87.38
-12.62    100.0             0.0
```

In [ ]:

```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_1sec.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)
```

```
df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
```

```
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0, 0, 0, 0.005, 0.005], sigma0=0.2, options={'seed'
    return res[0][:3], res[0][3:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(0, 5000), (5000, 10000), (10000, 15000), (15000, 20000)
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
```

```
        test_trades.append(trades)

    if not test_positions:
        raise ValueError("No positions generated.")

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:31
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.276504232233914e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.2
/tmp/ipython-input-5-3535381746.py:55: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-5-3535381746.py:55: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
```

```
    2     16 -3.016706954280951e-02 1.2e+00 1.77e-01  2e-01  2e-01 0:00.4
    3     24 -2.649338249359656e-02 1.4e+00 1.90e-01  2e-01  2e-01 0:00.7
   14    112 -6.035617317248676e-02 2.9e+00 1.92e-01  1e-01  2e-01 0:03.7
   29    232 -4.584555448178634e-02 5.7e+00 9.13e-02  4e-02  1e-01 0:07.9
   53    424 -6.419335567248741e-02 7.7e+00 2.56e-02  5e-03  2e-02 0:13.1
   75    600 -6.826542699354501e-02 1.9e+01 6.28e-03  9e-04  4e-03 0:19.2
  100    800 -7.021934446392347e-02 2.2e+01 4.49e-03  5e-04  2e-03 0:24.7
  132   1056 -7.076612971040702e-02 4.7e+01 5.00e-04  2e-05  2e-04 0:32.9
  138   1104 -7.076612971040702e-02 6.0e+01 3.83e-04  2e-05  1e-04 0:34.1
termination on tolflatfitness=1 (Tue Jul 22 12:58:06 2025)
final/bestever f-value = -7.076613e-02 -7.076613e-02 after 1105/896 evaluations
incumbent solution: [ 0.18819422, -0.08639572, 0.60476422, -0.30311123, -0.1199
4332]
std deviation: [4.28290525e-05, 1.76211424e-05, 1.36546141e-04, 1.12471506e-04,
9.87046764e-05]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:06
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.418305992611842e-03 1.0e+00 1.81e-01  2e-01  2e-01 0:00.2
    2     16 -1.209000541254016e-02 1.3e+00 1.68e-01  1e-01  2e-01 0:00.4
    3     24 -9.418305992611842e-03 1.3e+00 1.48e-01  1e-01  1e-01 0:00.6
   18    144 -1.570643291055912e-02 2.4e+00 8.25e-02  5e-02  7e-02 0:03.8
   37    296 -3.838895021104938e-02 3.9e+00 2.70e-02  8e-03  2e-02 0:08.1
   54    432 -4.954813980071393e-02 7.1e+00 7.58e-03  1e-03  6e-03 0:13.2
   80    640 -5.063260981435838e-02 1.5e+01 1.75e-03  1e-04  1e-03 0:19.3
  100    800 -5.158876413673499e-02 5.0e+01 2.93e-04  2e-05  2e-04 0:25.1
  114    912 -5.158876413673499e-02 9.2e+01 2.09e-04  1e-05  1e-04 0:28.1
termination on tolflatfitness=1 (Tue Jul 22 12:58:34 2025)
final/bestever f-value = -5.158876e-02 -5.158876e-02 after 913/736 evaluations
incumbent solution: [ 0.03961951, 0.04607656, -0.287509, -0.25723643, -0.371073
06]
std deviation: [3.35953615e-05, 9.97693670e-06, 1.23464441e-04, 1.46858792e-04,
2.40625297e-05]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:34
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.694278582117947e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.2
    2     16 -1.636190910817792e-02 1.4e+00 1.88e-01  2e-01  2e-01 0:00.5
    3     24 -1.636190910817792e-02 1.4e+00 1.98e-01  2e-01  2e-01 0:00.7
   17    136 -1.644446237996822e-02 3.3e+00 1.97e-01  1e-01  3e-01 0:03.8
   33    264 -3.451708216941757e-02 5.9e+00 3.52e-01  2e-01  7e-01 0:08.0
   54    432 -3.744247615769142e-02 1.1e+01 7.42e-02  2e-02  1e-01 0:13.2
   83    664 -4.011488846174288e-02 2.5e+01 5.75e-03  1e-03  8e-03 0:19.2
  100    800 -4.019825163936386e-02 6.2e+01 4.81e-03  7e-04  8e-03 0:24.3
  108    864 -4.019825163936386e-02 7.7e+01 3.04e-03  4e-04  5e-03 0:26.0
termination on tolflatfitness=1 (Tue Jul 22 12:59:00 2025)
final/bestever f-value = -4.019825e-02 -4.023985e-02 after 865/699 evaluations
incumbent solution: [-0.5456578, -0.43020764, 2.41496429, -0.30144353, 0.517355
35]
std deviation: [0.00110122, 0.00047692, 0.00479706, 0.0008087, 0.00036932]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:00
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -6.977530670121912e-03 1.0e+00 1.95e-01  2e-01  2e-01 0:00.2
```

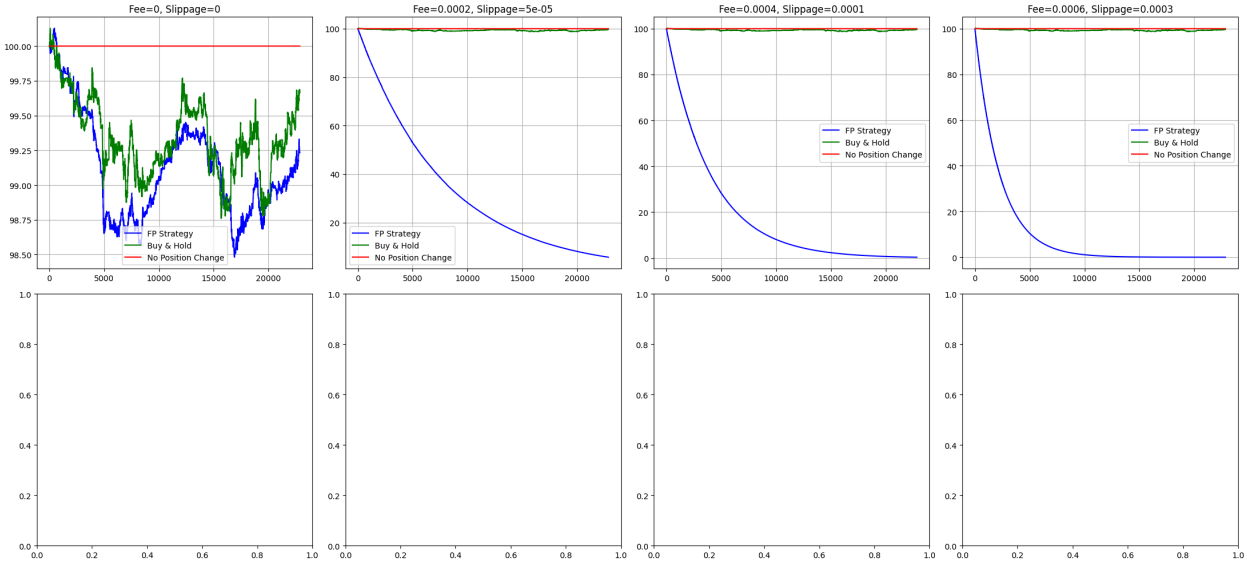
```
    2     16 -5.534838071495407e-03 1.3e+00 2.05e-01  2e-01  2e-01 0:00.5
    3     24 -9.382155795237601e-03 1.4e+00 1.91e-01  2e-01  2e-01 0:00.7
   17    136 -1.206982134026488e-02 3.1e+00 1.71e-01  1e-01  2e-01 0:03.8
   34    272 -1.401845945467359e-02 5.3e+00 5.58e-02  2e-02  6e-02 0:08.0
   54    432 -1.586524516608767e-02 1.1e+01 1.86e-02  4e-03  1e-02 0:13.1
   79    632 -1.611571897890085e-02 3.3e+01 4.09e-03  6e-04  3e-03 0:18.3
termination on tolflatfitness=1 (Tue Jul 22 12:59:19 2025)
final/bestever f-value = -1.611572e-02 -1.611572e-02 after 633/468 evaluations
incumbent solution: [ 0.4792545, -0.30192912, 0.14514162, -0.07545933, -0.45140
901]
std deviation: [0.00185275, 0.00115466, 0.00298411, 0.00095148, 0.00064754]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:19
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.697404081786016e-02 1.0e+00 1.98e-01  2e-01  2e-01 0:00.2
    2     16 -2.171308312260109e-02 1.3e+00 2.48e-01  2e-01  3e-01 0:00.5
    3     24 -1.726496517358644e-02 1.4e+00 2.76e-01  3e-01  3e-01 0:00.7
   11     88 -2.820622107249079e-02 2.8e+00 2.90e-01  2e-01  4e-01 0:03.7
   18    144 -2.939304742374263e-02 3.2e+00 2.74e-01  2e-01  3e-01 0:07.9
   42    336 -4.380425847733055e-02 8.5e+00 8.89e-02  2e-02  1e-01 0:12.9
   63    504 -4.987699148223584e-02 2.0e+01 2.61e-02  3e-03  3e-02 0:18.9
   96    768 -5.101780290696131e-02 4.6e+01 8.24e-03  5e-04  9e-03 0:26.1
  100    800 -5.106015156345659e-02 4.8e+01 6.35e-03  3e-04  7e-03 0:26.9
  118    944 -5.106015156345659e-02 5.4e+01 2.35e-03  8e-05  3e-03 0:32.0
termination on tolflatfitness=1 (Tue Jul 22 12:59:51 2025)
final/bestever f-value = -5.106015e-02 -5.110116e-02 after 945/729 evaluations
incumbent solution: [-0.91525361, -1.36086633, 1.32441803, -1.76996603, 0.56468
196]
std deviation: [1.32078532e-03, 7.99207144e-05, 8.31495876e-04, 2.52900700e-03,
1.48101001e-04]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:17
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.013815228127616e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.3
    2     16 -2.013815228127616e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.5
    3     24 -2.013815228127616e-02 1.5e+00 1.96e-01  2e-01  2e-01 0:00.8
    8     64 -2.013815228127616e-02 1.9e+00 1.91e-01  1e-01  2e-01 0:01.9
termination on tolfun=1e-11 (Tue Jul 22 13:00:19 2025)
final/bestever f-value = -2.013815e-02 -2.013815e-02 after 65/5 evaluations
incumbent solution: [ 0.60383758, -0.15808528, 0.15510503, -0.36510072, 0.18983
288]
std deviation: [0.22590895, 0.14181433, 0.17192789, 0.18082173, 0.17489202]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:19
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.168305992611842e-03 1.0e+00 1.84e-01  2e-01  2e-01 0:00.2
    2     16 -9.168305992611842e-03 1.3e+00 2.08e-01  2e-01  2e-01 0:00.4
    3     24 -9.168305992611842e-03 1.7e+00 2.21e-01  2e-01  3e-01 0:00.6
   10     80 -9.168305992611842e-03 2.7e+00 2.05e-01  2e-01  2e-01 0:02.2
termination on tolfunhist=1e-12 (Tue Jul 22 13:00:21 2025)
final/bestever f-value = -9.168306e-03 -9.168306e-03 after 81/5 evaluations
incumbent solution: [ 0.64850304, -0.50076959, 0.07985729, 0.03476062, 0.094782
5, ]
std deviation: [0.21867616, 0.18028622, 0.20603718, 0.19513497, 0.15083332]
```

```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:21
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.619278582117947e-02 1.0e+00 2.10e-01  2e-01  2e-01 0:00.2
    2     16 -1.669185351381364e-02 1.5e+00 1.83e-01  2e-01  2e-01 0:00.4
    3     24 -1.611190910817792e-02 1.5e+00 1.78e-01  2e-01  2e-01 0:00.7
   17    136 -1.611190910817792e-02 2.7e+00 1.24e-01  6e-02  1e-01 0:03.7
   19    152 -1.611190910817792e-02 3.2e+00 1.09e-01  4e-02  1e-01 0:04.1
termination on tolflatfitness=1 (Tue Jul 22 13:00:25 2025)
final/bestever f-value = -1.611191e-02 -1.694077e-02 after 153/28 evaluations
incumbent solution: [-0.58509708, -0.13499903, 0.2645132, 0.48143123, -0.145829
27]
std deviation: [0.10010958, 0.06420824, 0.09038103, 0.12677925, 0.04302453]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:25
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.850530100709045e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.2
    2     16 -1.486913352745411e-02 1.3e+00 1.74e-01  2e-01  2e-01 0:00.4
    3     24 -1.149729143179848e-02 1.4e+00 1.76e-01  2e-01  2e-01 0:00.6
   14    112 -1.124707248891912e-02 2.5e+00 2.18e-01  1e-01  3e-01 0:04.0
   31    248 -1.224757282250255e-02 5.2e+00 1.05e-01  3e-02  1e-01 0:08.1
   55    440 -1.291401736566317e-02 1.4e+01 3.21e-02  6e-03  3e-02 0:13.3
   76    608 -1.283073608308330e-02 6.4e+01 1.35e-02  1e-03  2e-02 0:19.3
  100    800 -1.283073608308330e-02 4.7e+02 3.47e-02  3e-03  7e-02 0:24.5
  121    968 -1.299729171304859e-02 1.5e+03 1.68e-02  5e-04  3e-02 0:29.6
termination on tolflatfitness=1 (Tue Jul 22 13:00:55 2025)
final/bestever f-value = -1.299729e-02 -2.007193e-02 after 969/56 evaluations
incumbent solution: [ 0.31328292, 0.23749631, -0.20570458, 0.67910794, -0.30061
912]
std deviation: [0.01056475, 0.00052656, 0.01897466, 0.02916809, 0.00209034]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:55
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.262834211098904e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.4
    2     16 -1.262834211098904e-02 1.5e+00 2.10e-01  2e-01  2e-01 0:00.8
    3     24 -1.262834211098904e-02 1.4e+00 1.98e-01  2e-01  2e-01 0:01.1
   13    104 -1.262834211098904e-02 2.2e+00 1.62e-01  1e-01  2e-01 0:03.7
termination on tolflatfitness=1 (Tue Jul 22 13:00:59 2025)
final/bestever f-value = -1.262834e-02 -1.293908e-02 after 105/32 evaluations
incumbent solution: [-0.50106702, -0.10852934, -0.63804638, -0.07254969, -0.074
54924]
std deviation: [0.13134836, 0.14490984, 0.16408765, 0.1337006, 0.11352478]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:25
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.988815228127616e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.2
    2     16 -1.988815228127616e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.4
    3     24 -1.988815228127616e-02 1.5e+00 1.96e-01  2e-01  2e-01 0:00.6
   10     80 -1.988815228127616e-02 2.1e+00 1.82e-01  1e-01  2e-01 0:02.3
termination on tolfunhist=1e-12 (Tue Jul 22 13:01:27 2025)
final/bestever f-value = -1.988815e-02 -1.988815e-02 after 81/5 evaluations
incumbent solution: [ 0.68061589, -0.05470514, 0.36758562, -0.03449197, -0.2247
2304]
std deviation: [0.18711484, 0.1376055, 0.17714934, 0.1511006, 0.16069868]
```

```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:27
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.918305992611841e-03 1.0e+00 1.90e-01  2e-01  2e-01 0:00.2
    2     16 -8.918305992611841e-03 1.4e+00 1.85e-01  2e-01  2e-01 0:00.5
    3     24 -8.918305992611841e-03 1.5e+00 1.73e-01  1e-01  2e-01 0:00.7
    9     72 -8.918305992611841e-03 2.3e+00 1.21e-01  8e-02  1e-01 0:02.0
termination on tolfun=1e-11 (Tue Jul 22 13:01:29 2025)
final/bestever f-value = -8.918306e-03 -8.918306e-03 after 73/5 evaluations
incumbent solution: [ 0.33021464, 0.01520671, 0.15988409, -0.30578787, 0.177549
58]
std deviation: [0.11493965, 0.10179417, 0.12094759, 0.13676555, 0.08033268]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:30
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.544278582117947e-02 1.0e+00 2.10e-01  2e-01  2e-01 0:00.2
    2     16 -1.619185351381364e-02 1.5e+00 1.83e-01  2e-01  2e-01 0:00.4
    3     24 -1.586190910817792e-02 1.5e+00 1.78e-01  2e-01  2e-01 0:00.6
   18    144 -1.644076969161196e-02 4.4e+00 9.07e-02  3e-02  1e-01 0:03.8
   30    240 -1.644076969161196e-02 5.6e+00 5.88e-02  2e-02  7e-02 0:07.9
   35    280 -1.644076969161196e-02 6.2e+00 5.11e-02  1e-02  5e-02 0:08.9
termination on tolflatfitness=1 (Tue Jul 22 13:01:39 2025)
final/bestever f-value = -1.644077e-02 -1.903924e-02 after 281/45 evaluations
incumbent solution: [0.0726835, 0.06422769, 0.75284444, 0.70292068, 0.13823752]
std deviation: [0.01453685, 0.01526799, 0.05292858, 0.04550151, 0.03840524]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:39
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.600530100709045e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.2
    2     16 -1.436913352745411e-02 1.3e+00 1.74e-01  2e-01  2e-01 0:00.4
    3     24 -1.099729143179848e-02 1.4e+00 1.76e-01  2e-01  2e-01 0:00.6
   13    104 -5.600530100709045e-03 3.3e+00 2.03e-01  2e-01  2e-01 0:02.7
termination on tolflatfitness=1 (Tue Jul 22 13:01:41 2025)
final/bestever f-value = -5.600530e-03 -1.436913e-02 after 105/10 evaluations
incumbent solution: [ 0.71278011, -0.42167171, 0.03667235, 0.11890289, -0.06474
42, ]
std deviation: [0.23070862, 0.2077817, 0.17696191, 0.17133213, 0.19034418]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:41
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.237834211098904e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.2
    2     16 -1.237834211098904e-02 1.5e+00 2.23e-01  2e-01  2e-01 0:00.4
    3     24 -1.237834211098904e-02 1.5e+00 2.06e-01  2e-01  2e-01 0:00.6
   15    120 -1.237834211098904e-02 2.9e+00 2.02e-01  1e-01  2e-01 0:03.1
termination on tolflatfitness=1 (Tue Jul 22 13:01:45 2025)
final/bestever f-value = -1.237834e-02 -1.867289e-02 after 121/42 evaluations
incumbent solution: [-0.6751727, -0.12569183, -0.78951025, -0.32606908, 0.03202
035]
std deviation: [0.16530212, 0.19444262, 0.23028925, 0.19072832, 0.1179752, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:10
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.948815228127616e-02 1.0e+00 1.85e-01  2e-01  2e-01 0:00.2
    2     16 -1.948815228127616e-02 1.3e+00 1.69e-01  2e-01  2e-01 0:00.4
```

```
    3     24 -1.948815228127616e-02 1.4e+00 1.55e-01  1e-01  2e-01 0:00.6
    5     40 -1.948815228127616e-02 1.8e+00 1.47e-01  1e-01  2e-01 0:01.1
termination on tolfun=1e-11 (Tue Jul 22 13:02:11 2025)
final/bestever f-value = -1.948815e-02 -1.948815e-02 after 41/5 evaluations
incumbent solution: [ 0.29244837, -0.0255148, -0.09246226, 0.10433268, -0.07144
378]
std deviation: [0.12867797, 0.12199257, 0.1507923, 0.14306658, 0.14235476]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:11
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.518305992611842e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -8.518305992611842e-03 1.4e+00 1.87e-01  2e-01  2e-01 0:00.4
    3     24 -8.518305992611842e-03 1.5e+00 1.63e-01  1e-01  2e-01 0:00.6
   16    128 -8.518305992611842e-03 3.5e+00 7.52e-02  5e-02  8e-02 0:03.9
   19    152 -8.518305992611842e-03 4.1e+00 6.75e-02  4e-02  7e-02 0:05.0
termination on tolflatfitness=1 (Tue Jul 22 13:02:16 2025)
final/bestever f-value = -8.518306e-03 -9.400284e-03 after 153/27 evaluations
incumbent solution: [ 0.10868063, 0.00863946, 0.14844971, -0.35791806, 0.181013
06]
std deviation: [0.05547147, 0.04404037, 0.05227011, 0.0706884, 0.06366208]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:16
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.424278582117947e-02 1.0e+00 2.10e-01  2e-01  2e-01 0:00.4
    2     16 -1.539185351381364e-02 1.5e+00 1.83e-01  2e-01  2e-01 0:00.6
    3     24 -1.546190910817792e-02 1.5e+00 1.78e-01  2e-01  2e-01 0:00.9
   18    144 -1.730806094277426e-02 2.4e+00 1.04e-01  6e-02  1e-01 0:03.9
   22    176 -1.546190910817792e-02 3.2e+00 6.12e-02  3e-02  6e-02 0:04.8
termination on tolflatfitness=1 (Tue Jul 22 13:02:21 2025)
final/bestever f-value = -1.546191e-02 -1.781587e-02 after 177/156 evaluations
incumbent solution: [-0.19903723, 0.09839819, -0.31276708, 0.08643728, 0.099856
5, ]
std deviation: [0.04277221, 0.05800017, 0.05611035, 0.03686753, 0.02945283]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:21
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.200530100709045e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.2
    2     16 -1.356913352745412e-02 1.3e+00 1.74e-01  2e-01  2e-01 0:00.4
    3     24 -1.019729143179848e-02 1.4e+00 1.76e-01  2e-01  2e-01 0:00.6
   18    144 -9.446446696235396e-03 3.4e+00 2.33e-01  1e-01  3e-01 0:03.7
   33    264 -1.136415270771873e-02 5.4e+00 1.30e-01  3e-02  2e-01 0:07.9
   55    440 -1.182219023316338e-02 1.5e+01 6.98e-02  7e-03  7e-02 0:13.0
   84    672 -1.236323696366498e-02 1.1e+02 1.56e-01  5e-03  2e-01 0:19.1
  100    800 -1.406770015874631e-02 2.3e+02 4.31e-02  8e-04  5e-02 0:23.9
  119    952 -1.406770015874631e-02 8.3e+02 1.78e-02  3e-04  2e-02 0:27.9
termination on tolflatfitness=1 (Tue Jul 22 13:02:49 2025)
final/bestever f-value = -1.406770e-02 -1.435842e-02 after 953/504 evaluations
incumbent solution: [ 0.0954617, 0.18227138, -0.32955342, -0.94781397, 0.085068
65]
std deviation: [0.0125162, 0.00029036, 0.02497729, 0.01245562, 0.00442149]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.197834211098904e-02 1.0e+00 2.10e-01  2e-01  2e-01 0:00.2
```

```
    2     16 -1.197834211098904e-02 1.5e+00 2.19e-01  2e-01  2e-01 0:00.4
    3     24 -1.197834211098904e-02 1.5e+00 2.08e-01  2e-01  2e-01 0:00.7
   10     80 -1.197834211098904e-02 2.5e+00 3.34e-01  2e-01  4e-01 0:02.4
termination on tolfunhist=1e-12 (Tue Jul 22 13:02:52 2025)
final/bestever f-value = -1.197834e-02 -1.197834e-02 after 81/8 evaluations
incumbent solution: [-0.86008801, -0.67772141, -0.17746793, -0.39486652, 0.1814
7408]
std deviation: [0.3519892, 0.28837277, 0.37381927, 0.37211275, 0.244496, ]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            99.23          -0.77           99.67
-0.33    100.0             0.0
0.0002   0.00005             5.67         -94.33           99.67
-0.33    100.0             0.0
0.0004   0.00010             0.32         -99.68           99.67
-0.33    100.0             0.0
0.0006   0.00030             0.00        -100.00           99.67
-0.33    100.0             0.0
```

In [ ]:

```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt
from numba import njit
from sklearn.preprocessing import RobustScaler
from scipy.stats import norm
from statsmodels.tsa.statespace.tools import constrain_stationary_univariate

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_5min.csv", parse_dates=['system_time'], index_col='syste
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}'] * (1 + 0.
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}'] * (1 + 0.
```

```
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = np.log1p(df[bid_cols + ask_cols].sum(axis=1))
df['queue_slope'] = (df['bids_notional_0'] - df['bids_notional_5']) / (df['bic

scaler = RobustScaler()
features = ['obi', 'dobi', 'depth', 'queue_slope']
df[features] = scaler.fit_transform(df[features])

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

@njit
def trading_strategy(signal, threshold, volatility):
    positions = np.zeros(len(signal))
    for i in range(1, len(signal)):
        z_score = signal[i] / (volatility[i] + 1e-8)
        if z_score > threshold:
            positions[i] = min(positions[i-1] + 0.1, 1)
        elif z_score < -threshold:
            positions[i] = max(positions[i-1] - 0.1, -1)
        else:
            positions[i] = positions[i-1] * 0.95
    # Manual diff with prepend=0
    trades = np.zeros(len(positions))
    trades[0] = positions[0]
    for i in range(1, len(positions)):
        trades[i] = positions[i] - positions[i-1]
    return positions, trades

@njit
def apply_trading_costs(positions, trades, returns, fee, slip, volatility):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip * volatility[1:len(positions)][trade_mask]
    net_pnl = raw_pnl - costs
    return net_pnl

@njit
def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
```

```
    a0, a1, a2, a3 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1] + a3 * np.tanh(x[t-1])
        sigma = np.exp(b0 + b1 * np.log1p(np.abs(x[t-1])) + b2)
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * np.random.randn()
    return x

def optimize_threshold(signal, returns, fee, slip, volatility):
    thresholds = np.geomspace(0.001, 0.1, 20)
    best_pnl = -np.inf
    best_thresh = 0.01
    for t in thresholds:
        pos, trades = trading_strategy(signal, t, volatility)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip, vola
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0

    def objective(params):
        mu_params = params[:4]
        sigma_params = params[4:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, 0.01, np.ones_like(signal)*vola
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip, np

    res = fmin(objective, [0, -0.5, 0.5, 0.1, -2, 0.1, 0.01], sigma0=0.2, opti
    return res[0][:4], res[0][4:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i*200, (i+1)*200) for i in range(5)]
    segment_models = []
    segment_thresholds = []

    for start, end in train_segments:
        if end > len(df_train):
            continue
```

```
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 5
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []

    for start in range(0, len(cv_returns) - window_size + 1, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            volatility = np.sqrt(np.mean(np.diff(signal)**2))
            pos, trades = trading_strategy(signal, segment_thresholds[i], np.
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    test_volatility = []

    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, threshold, np.ones_like(signal)
        test_positions.append(pos)
        test_trades.append(trades)
        test_volatility.extend([volatility]*len(pos))

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_volatility = np.array(test_volatility[:len(fp_positions)])
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns), len(fp_volatility))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_volatility = fp_volatility[:min_length]
```

```
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    npc_returns = fp_positions * bh_returns - (fee + slip * fp_volatility) * (
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee:.4f}, Slippage={slip:.5f}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:13 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.740109577149550e-02 1.0e+00 2.05e-01  2e-01  2e-01 0:07.9
    2     40 -2.011194181198942e-02 1.3e+00 2.27e-01  2e-01  3e-01 0:07.9
    3     60 -1.857641548358935e-02 1.7e+00 2.30e-01  2e-01  3e-01 0:07.9
  100   2000 -1.976500490534143e-02 2.7e+01 9.28e-02  3e-02  9e-02 0:09.0
  200   4000 -2.006136097942746e-02 5.4e+01 2.29e-01  3e-02  1e-01 0:10.0
  300   6000 -2.006136097942746e-02 1.7e+02 1.80e-01  8e-03  6e-02 0:10.7
  400   8000 -1.968347195242363e-02 6.8e+02 1.84e-01  6e-03  6e-02 0:11.3
  500  10000 -2.048176784798172e-02 2.6e+03 3.40e-02  7e-04  7e-03 0:11.9
  545  10900 -2.006136097942746e-02 7.5e+03 4.38e-02  9e-04  9e-03 0:12.3
termination on tolstagnation=192 (Tue Jul 22 17:23:27 2025)
final/bestever f-value = -1.839878e-02 -2.312447e-02 after 10901/92 evaluations
incumbent solution: [ 0.85870482, -0.26130122, 2.13524882, -0.19359889, -1.9622
8206, -1.64389571, 1.05998944]
std deviation: [0.00308739, 0.0009064, 0.00902589, 0.00261468, 0.00133067, 0.00
940953, 0.00705431]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:27 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -2.037247944299925e-02 1.0e+00 2.16e-01  2e-01  2e-01 0:00.0
    2     40 -5.365480329141019e-03 1.4e+00 2.30e-01  2e-01  3e-01 0:00.0
    3     60 -6.187515965503623e-03 1.5e+00 2.10e-01  2e-01  2e-01 0:00.0
/tmp/ipython-input-3-2769142098.py:103: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1452:
RuntimeWarning: invalid value encountered in subtract
  a = op(a[slice1], a[slice2])
```

```
   96   1920 -1.429134149237447e-02 3.4e+02 2.27e-01  1e-03  3e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:28 2025)
final/bestever f-value = -1.429134e-02 -3.662547e-02 after 1921/673 evaluations
incumbent solution: [-0.17410085, 0.14269729, -0.1606787, 0.3990421, -2.534935
6, -1.0384218, -1.16113488]
std deviation: [0.07073059, 0.00095826, 0.09653089, 0.17173039, 0.29609555, 0.2
3655673, 0.24583396]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:28 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -8.113595876621056e-03 1.0e+00 2.08e-01  2e-01  2e-01 0:00.0
    2     40 -8.602308963684397e-03 1.4e+00 1.90e-01  2e-01  2e-01 0:00.0
    3     60 -1.076600642761528e-02 1.5e+00 2.01e-01  2e-01  2e-01 0:00.0
  100   2000 -1.557923756725986e-02 3.4e+01 1.02e-01  3e-02  1e-01 0:00.8
  200   4000 -1.348920853008616e-02 3.1e+02 1.48e-02  3e-03  2e-02 0:01.3
  300   6000 -1.434492248648877e-02 1.2e+03 3.84e-03  3e-04  2e-03 0:01.7
  400   8000 -1.476155897637644e-02 9.9e+03 5.55e-03  2e-04  5e-03 0:02.1
  500  10000 -1.489930492635416e-02 7.0e+04 2.70e-03  1e-04  2e-03 0:02.4
  505  10100 -1.688477138865112e-02 8.7e+04 2.87e-03  1e-04  2e-03 0:02.4
termination on tolstagnation=192 (Tue Jul 22 17:23:32 2025)
final/bestever f-value = -1.372101e-02 -2.290941e-02 after 10101/81 evaluations
incumbent solution: [-0.54251705, -0.50009193, 1.03666201, -0.13962489, -2.0446
4429, 0.33239462, 0.44428445]
std deviation: [0.00137955, 0.00026678, 0.00225032, 0.00010056, 0.00150222, 0.0
0106065, 0.0014245, ]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:32 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -2.673982254923472e-02 1.0e+00 2.04e-01  2e-01  2e-01 0:00.0
    2     40 -2.610223207360131e-02 1.3e+00 2.23e-01  2e-01  2e-01 0:00.0
    3     60 -2.384617830494792e-02 1.6e+00 2.12e-01  2e-01  2e-01 0:00.0
  100   2000 -2.962443715516166e-02 1.9e+01 1.25e-01  2e-02  1e-01 0:00.3
  200   4000 -2.917646112652971e-02 1.7e+02 1.67e-01  2e-02  2e-01 0:00.8
  300   6000 -2.761064358215672e-02 9.6e+02 8.36e-03  1e-03  7e-03 0:01.3
  400   8000 -2.714012932838109e-02 2.5e+03 1.30e-02  2e-03  1e-02 0:01.7
  500  10000 -2.976483567694747e-02 1.7e+04 1.29e-01  1e-02  1e-01 0:02.2
  550  11000 -2.762949063358078e-02 2.0e+04 1.65e-01  1e-02  8e-02 0:02.5
termination on tolstagnation=192 (Tue Jul 22 17:23:35 2025)
final/bestever f-value = -1.782863e-02 -4.340551e-02 after 11001/8605 evaluatio
ns
incumbent solution: [ 0.09931256, 0.07353394, 0.27497396, -0.88753666, -0.77304
997, -1.28690556, 0.63578896]
std deviation: [0.0141038, 0.01105372, 0.05929408, 0.08398773, 0.04987581, 0.07
05869, 0.02921597]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:35 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.552355982050381e-02 1.0e+00 1.88e-01  2e-01  2e-01 0:00.0
    2     40 -1.542367188823013e-02 1.3e+00 1.81e-01  2e-01  2e-01 0:00.0
    3     60 -1.785406292920010e-02 1.4e+00 1.62e-01  1e-01  2e-01 0:00.0
  100   2000 -2.245022289900691e-02 2.2e+01 2.19e-02  3e-03  2e-02 0:00.3
  200   4000 -1.852861621558923e-02 1.8e+02 7.27e-03  9e-04  1e-02 0:00.7
  300   6000 -1.898581687739589e-02 2.3e+03 4.21e-03  6e-04  7e-03 0:01.0
  400   8000 -2.564546838102715e-02 9.3e+03 5.50e-04  5e-05  8e-04 0:01.3
```

```
  500  10000 -2.254172196805899e-02 7.2e+04 2.99e-04  3e-05  3e-04 0:01.6
  505  10100 -1.963238046454739e-02 7.8e+04 3.42e-04  3e-05  4e-04 0:01.6
termination on tolstagnation=192 (Tue Jul 22 17:23:37 2025)
final/bestever f-value = -9.756287e-03 -2.564547e-02 after 10101/7989 evaluatio
ns
incumbent solution: [-0.12083504, -0.40416239, 1.17458757, 0.09134106, -2.01753
225, 0.22405773, 0.16100976]
std deviation: [7.57789552e-05, 5.88192339e-05, 4.06736835e-04, 4.66229909e-05,
3.38916707e-05, 1.99771998e-04, 2.49605718e-04]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:37 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -2.267472949370719e-03 1.0e+00 2.17e-01  2e-01  2e-01 0:00.6
    2     40 -9.763491960806626e-03 1.5e+00 2.46e-01  2e-01  3e-01 0:00.6
    3     60 -9.866639673498556e-03 1.6e+00 2.78e-01  2e-01  3e-01 0:00.6
  100   2000 -9.987551062693376e-03 9.4e+03 2.22e-01  7e-05  5e-01 0:00.9
NOTE (module=cma, iteration=104):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.7e+08 to 1.8e+02
NOTE (module=cma, iteration=193):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 3.5e+08 to 2.7e+04
  200   4000 -9.987566144900067e-03 2.2e+02 1.41e-01  9e-09  2e+00 0:01.2
  227   4540 -9.987566145727881e-03 2.9e+02 3.99e-02  3e-10  5e-01 0:01.3
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:39 2025)
final/bestever f-value = -9.987566e-03 -1.079068e-02 after 4541/407 evaluations
incumbent solution: [ 3.65619345e-09, -1.17528197e+01, 8.86913488e-11, 1.082704
56e+01, -2.15559174e+01, -1.19515525e+01, -1.22610800e+00]
std deviation: [2.22208613e-09, 3.51238140e-01, 2.93228690e-10, 3.54441444e-01,
4.53494126e-01, 3.17347918e-01, 1.17294830e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:39 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.127066246337008e-03 1.0e+00 2.41e-01  2e-01  3e-01 0:00.0
    2     40 -1.975239090163025e-03 1.5e+00 2.76e-01  2e-01  3e-01 0:00.0
    3     60 -2.042841567505459e-03 1.6e+00 2.95e-01  3e-01  3e-01 0:00.0
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.4e+08 to 1.6e+02
  100   2000 -2.121820064177162e-03 1.5e+01 6.48e-02  2e-05  3e-01 0:00.3
NOTE (module=cma, iteration=194):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 3.9e+08 to 5.6e+03
  200   4000 -2.121825401918019e-03 1.1e+02 5.09e-02  4e-09  8e-01 0:00.6
  217   4340 -2.121825402692739e-03 1.0e+02 2.15e-02  3e-10  3e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:40 2025)
final/bestever f-value = -2.121825e-03 -2.121825e-03 after 4341/4277 evaluation
s
incumbent solution: [ 1.52634562e-09, -4.43292579e+00, 3.93043590e-12, 4.041694
27e+00, -1.05753027e+01, -1.35354634e+01, -1.27946958e+01]
std deviation: [1.52209596e-09, 1.12496334e-01, 3.29907767e-10, 1.23022752e-01,
1.48730780e-01, 3.45638315e-01, 2.64079818e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:40 2025)
```

```
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -4.465037098362948e-06 1.0e+00 2.24e-01  2e-01  2e-01 0:00.0
    2     40 -4.781659641496876e-03 1.4e+00 2.92e-01  3e-01  3e-01 0:00.0
    3     60 -1.873255554212840e-03 1.5e+00 2.94e-01  3e-01  3e-01 0:00.0
  100   2000 -1.041241339367659e-03 3.9e+03 1.72e-01  8e-05  3e-01 0:00.3
NOTE (module=cma, iteration=114):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 1.5e+01
  200   4000 -2.297235839937251e-03 6.4e+03 8.29e-02  2e-08  6e-01 0:00.6
NOTE (module=cma, iteration=249):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 3.5e+09 to 1.1e+04
  300   6000 -4.267141292656196e-03 2.0e+02 3.41e-02  2e-10  3e-01 0:00.9
  321   6420 -4.267141293184241e-03 4.6e+02 2.30e-02  5e-11  2e-01 0:01.0
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:41 2025)
final/bestever f-value = -4.267141e-03 -4.934191e-03 after 6421/260 evaluations
incumbent solution: [-3.50865581e-09, -2.83049821e+00, 4.54570615e-10, 8.353142
50e-01, -2.29370013e+01, -5.15320318e+00, -2.69702986e+00]
std deviation: [4.61924981e-10, 3.17839826e-02, 5.34881642e-11, 3.18790949e-02,
1.53150651e-01, 4.19588384e-02, 1.04660797e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:41 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.728126209098651e-02 1.0e+00 2.32e-01  2e-01  2e-01 0:00.0
    2     40 -1.827617667037628e-02 1.5e+00 2.79e-01  2e-01  3e-01 0:00.0
    3     60 -1.829209202111777e-02 1.5e+00 2.84e-01  2e-01  3e-01 0:00.0
/tmp/ipython-input-3-2769142098.py:103: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1452:
RuntimeWarning: invalid value encountered in subtract
  a = op(a[slice1], a[slice2])
```

```
  100   2000 -1.823326254745948e-02 8.4e+03 5.46e-02  2e-05  1e-01 0:00.3
NOTE (module=cma, iteration=104):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.6e+08 to 1.0e+02
NOTE (module=cma, iteration=198):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.9e+08 to 5.2e+03
  200   4000 -1.823327205566508e-02 7.7e+01 1.89e-02  2e-09  2e-01 0:00.6
  300   6000 -2.387583352889450e-02 9.1e+03 1.98e-03  6e-13  5e-02 0:00.9
  400   8000 -2.469008733198852e-02 2.1e+04 6.73e-04  1e-13  8e-03 0:01.2
  500  10000 -2.450325195095755e-02 3.9e+04 3.11e-04  4e-14  2e-03 0:01.5
  570  11400 -2.476961911223755e-02 9.6e+04 9.47e-04  9e-14  4e-03 0:01.7
termination on tolstagnation=192 (Tue Jul 22 17:23:44 2025)
final/bestever f-value = -2.035872e-02 -2.494925e-02 after 11401/8628 evaluatio
ns
incumbent solution: [ 2.17393065e-11, -3.04586878e+00, 4.00533650e-13, 2.851913
86e+00, -1.95153321e+01, 1.32476227e+01, -7.05097697e+00]
std deviation: [9.01222858e-14, 4.41647506e-04, 8.74171059e-14, 5.62662037e-04,
3.71204279e-03, 3.79398460e-03, 2.65233073e-03]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:44 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -7.810960926098443e-03 1.0e+00 2.15e-01  2e-01  2e-01 0:00.0
    2     40 -7.949486261353588e-03 1.3e+00 2.66e-01  2e-01  3e-01 0:00.0
    3     60 -7.950363668349992e-03 1.3e+00 2.81e-01  3e-01  3e-01 0:00.0
  100   2000 -8.017996190768581e-03 9.8e+03 1.54e-01  4e-05  3e-01 0:00.3
NOTE (module=cma, iteration=101):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 3.2e+01
  194   3880 -8.018007079837830e-03 6.5e+03 4.58e-02  5e-09  2e-01 0:00.6
termination on tolfun=1e-11 (Tue Jul 22 17:23:45 2025)
final/bestever f-value = -8.018007e-03 -8.262067e-03 after 3881/476 evaluations
incumbent solution: [-4.22369559e-08, -1.61761352e+00, 1.64604074e-10, 8.892317
93e-01, -1.61341794e+01, -3.56958966e+00, -3.36826281e+00]
std deviation: [3.04312438e-08, 6.18039296e-02, 5.14025830e-09, 6.71237568e-02,
1.99566844e-01, 1.56609482e-01, 7.94417606e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:45 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -5.740487691136553e-04 1.0e+00 2.31e-01  2e-01  2e-01 0:00.0
    2     40 -7.518111775735111e-03 1.4e+00 2.59e-01  2e-01  3e-01 0:00.0
    3     60 -7.476070491166595e-03 1.4e+00 2.81e-01  2e-01  3e-01 0:00.0
  100   2000 -7.787534344413925e-03 1.0e+04 1.20e-01  3e-05  3e-01 0:00.3
NOTE (module=cma, iteration=101):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 2.0e+02
NOTE (module=cma, iteration=187):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 2.8e+08 to 1.4e+04
  200   4000 -7.787566143409416e-03 1.3e+02 5.13e-02  4e-09  5e-01 0:00.7
  228   4560 -7.787566145776498e-03 1.3e+02 2.56e-02  2e-10  3e-01 0:00.9
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:46 2025)
final/bestever f-value = -7.787566e-03 -7.787566e-03 after 4561/4522 evaluation
s
```

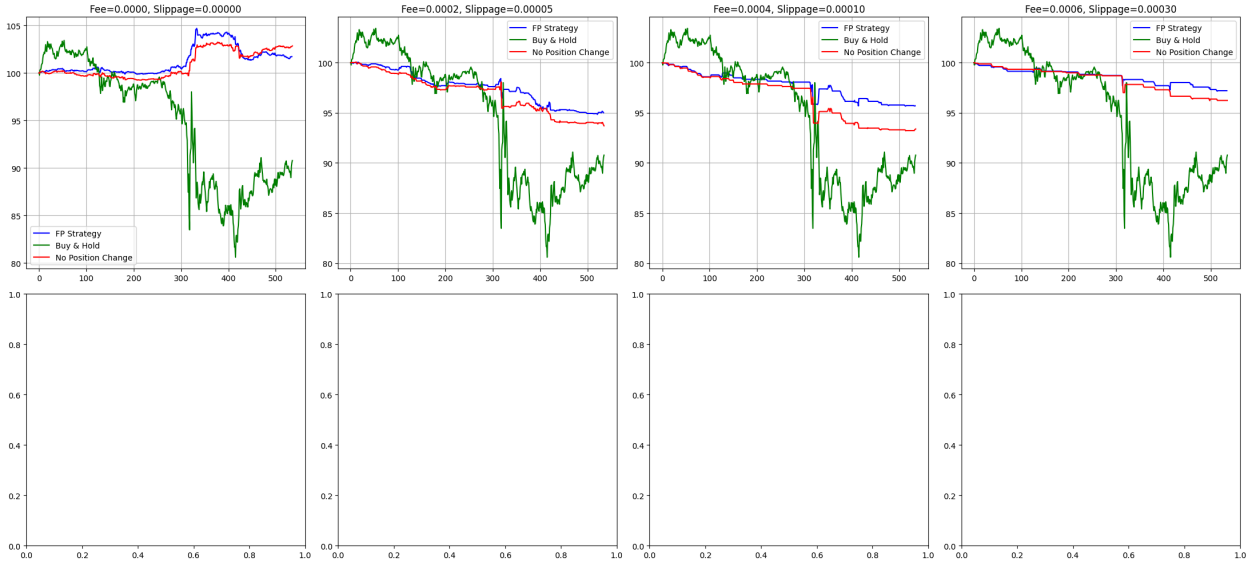
```
incumbent solution: [ 1.34912379e-09, -6.10641245e+00, -6.56596850e-12, 5.21701
268e+00, -1.41691237e+01, 1.93131679e+01, -9.24357261e+00]
std deviation: [1.06773318e-09, 9.42320232e-02, 1.95615716e-10, 8.39813648e-02,
1.93256233e-01, 2.90029629e-01, 1.13335184e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:46 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 1.568849720780446e-03 1.0e+00 2.43e-01  2e-01  3e-01 0:00.0
    2     40 2.055876857497106e-04 1.5e+00 2.96e-01  3e-01  3e-01 0:00.0
    3     60 1.962758432991017e-04 1.5e+00 3.35e-01  3e-01  4e-01 0:00.0
NOTE (module=cma, iteration=93):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 9.3e+02
  100   2000 7.819941766829680e-05 3.7e+01 1.46e-01  2e-05  5e-01 0:00.4
NOTE (module=cma, iteration=190):
condition in coordinate system exceeded 1.5e+08, rescaled to 1.0e+00,
condition changed from 2.7e+08 to 6.8e+04
  200   4000 7.817459908860021e-05 3.3e+02 5.47e-02  4e-09  1e+00 0:00.9
  224   4480 7.817459751848291e-05 2.2e+02 2.28e-02  3e-10  3e-01 0:01.0
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:47 2025)
final/bestever f-value = 7.817460e-05 7.817460e-05 after 4481/4471 evaluations
incumbent solution: [ 1.35944539e-09, -1.14081203e+01, -3.28917102e-11, 1.08038
754e+01, -2.97744600e+01, 3.26079256e+00, 7.44120770e+00]
std deviation: [1.29936225e-09, 1.05579857e-01, 2.70190101e-10, 8.79332459e-02,
3.42141992e-01, 9.13355170e-02, 1.51522379e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:47 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 3.730773364475599e-03 1.0e+00 2.19e-01  2e-01  2e-01 0:00.0
    2     40 1.359668168489285e-03 1.4e+00 3.07e-01  3e-01  4e-01 0:00.0
    3     60 1.223200674254979e-03 1.7e+00 4.01e-01  3e-01  5e-01 0:00.0
  100   2000 -5.612453919966075e-03 4.8e+02 7.68e-02  9e-04  1e-01 0:00.5
  200   4000 -8.866615014936539e-03 1.5e+03 2.20e-01  9e-04  2e-01 0:00.8
  300   6000 -3.916086667114525e-03 9.9e+02 3.12e-02  4e-05  6e-03 0:01.1
  400   8000 -2.923799186367592e-03 8.2e+03 4.82e-02  4e-05  7e-03 0:01.4
  500  10000 -2.923427208526525e-03 1.7e+04 5.01e-02  1e-05  4e-03 0:01.7
  530  10600 -3.634815205750679e-03 2.1e+04 2.78e-02  5e-06  2e-03 0:01.8
termination on tolstagnation=192 (Tue Jul 22 17:23:50 2025)
final/bestever f-value = 3.072559e-04 -1.223805e-02 after 10601/5648 evaluation
s
incumbent solution: [-1.62089568e-03, -6.54033066e-01, 9.59428830e-04, 6.312363
17e-01, -4.74114290e+00, -1.31022805e+00, -9.17494719e-01]
std deviation: [5.19824559e-06, 1.97929491e-03, 3.05795357e-05, 1.59798195e-03,
1.26612498e-03, 1.74633341e-03, 1.10016484e-03]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:50 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.370093532274543e-02 1.0e+00 2.44e-01  2e-01  3e-01 0:00.0
    2     40 -1.572006123979251e-02 1.5e+00 3.01e-01  3e-01  3e-01 0:00.0
    3     60 -1.574100464970266e-02 1.5e+00 3.35e-01  3e-01  4e-01 0:00.0
NOTE (module=cma, iteration=93):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.7e+08 to 1.7e+02
  100   2000 -1.603323417405541e-02 1.2e+01 1.77e-01  5e-05  8e-01 0:00.3
```

```
NOTE (module=cma, iteration=196):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.5e+08 to 4.9e+02
  200   4000 -1.603327205203871e-02 2.5e+01 4.49e-02  3e-09  4e-01 0:00.6
  300   6000 -2.256961911223531e-02 1.9e+04 1.95e-02  3e-12  5e-01 0:00.9
NOTE (module=cma, iteration=302):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 3.8e+08 to 1.5e+04
  321   6420 -2.266514398711642e-02 9.5e+01 6.30e-03  4e-13  1e-01 0:00.9
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:51 2025)
final/bestever f-value = -2.266514e-02 -2.266514e-02 after 6421/6421 evaluation
s
incumbent solution: [ 9.05599724e-12, 9.26335059e-01, 1.16314282e-14, -9.764097
89e-01, -3.71593446e+01, -7.95105763e+00, 8.79977705e+00]
std deviation: [4.15769474e-13, 1.72579264e-03, 5.57719871e-13, 7.22709024e-03,
1.20910770e-01, 7.63222011e-02, 4.46586021e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:51 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -5.389542079480574e-03 1.0e+00 2.24e-01  2e-01  2e-01 0:00.0
    2     40 -5.565889741080096e-03 1.3e+00 2.46e-01  2e-01  3e-01 0:00.0
    3     60 -5.565928389933939e-03 1.6e+00 2.73e-01  3e-01  3e-01 0:00.0
  100   2000 -5.817920589501663e-03 1.1e+04 4.77e-01  2e-04  1e+00 0:00.3
NOTE (module=cma, iteration=109):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 2.4e+08 to 1.4e+02
  200   4000 -5.818007076308833e-03 7.6e+03 7.78e-02  8e-09  3e-01 0:00.6
NOTE (module=cma, iteration=211):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.5e+08 to 7.0e+03
  245   4900 -5.818007082232551e-03 1.3e+02 4.31e-02  1e-10  4e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:52 2025)
final/bestever f-value = -5.818007e-03 -5.818007e-03 after 4901/4869 evaluation
s
incumbent solution: [-1.17038190e-09, 7.48360262e+00, 2.04782956e-11, -8.210349
48e+00, 9.88954012e-01, -7.88559565e-01, -2.40942383e+01]
std deviation: [9.66912868e-10, 1.98970867e-01, 1.48563987e-10, 1.81333239e-01,
1.57333253e-01, 1.50168350e-01, 3.97919116e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:52 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.789831964725559e-03 1.0e+00 2.34e-01  2e-01  3e-01 0:00.0
    2     40 -4.813074867289689e-03 1.4e+00 2.54e-01  2e-01  3e-01 0:00.0
    3     60 -5.151786133207187e-03 1.3e+00 2.83e-01  3e-01  3e-01 0:00.0
  100   2000 -5.587261762319906e-03 4.7e+03 3.16e-01  2e-04  6e-01 0:00.3
NOTE (module=cma, iteration=112):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.8e+08 to 3.9e+02
  200   4000 -5.587566116966029e-03 5.2e+03 1.07e-01  2e-08  7e-01 0:00.6
NOTE (module=cma, iteration=212):
condition in coordinate system exceeded 1.5e+08, rescaled to 1.0e+00,
condition changed from 1.8e+08 to 2.3e+03
  231   4620 -5.587566144859869e-03 4.8e+01 3.80e-02  5e-10  3e-01 0:00.7
termination on tolfun=1e-11 (Tue Jul 22 17:23:53 2025)
```

```
final/bestever f-value = -5.587566e-03 -5.587566e-03 after 4621/4545 evaluation
s
incumbent solution: [ 3.35961726e-09, -5.03529759e+00, -1.28162266e-11, 3.72537
482e+00, -1.87560375e+00, 9.28525840e+00, -2.08903406e+01]
std deviation: [3.36135514e-09, 5.51074366e-02, 4.79901509e-10, 6.23149118e-02,
6.01810157e-02, 2.00493805e-01, 3.02033286e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:53 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 4.496367410405613e-03 1.0e+00 2.36e-01  2e-01  2e-01 0:00.0
    2     40 2.661680607358819e-03 1.4e+00 3.36e-01  3e-01  4e-01 0:00.0
    3     60 2.500254738615253e-03 1.6e+00 4.26e-01  4e-01  5e-01 0:00.0
  100   2000 2.278341824445099e-03 2.6e+03 8.95e-02  6e-05  1e-01 0:00.3
NOTE (module=cma, iteration=121):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 4.2e+01
  200   4000 2.278174635564317e-03 2.0e+03 3.07e-02  7e-09  1e-01 0:00.6
NOTE (module=cma, iteration=214):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 1.1e+03
  243   4860 2.278174600495213e-03 2.7e+01 2.30e-02  2e-10  1e-01 0:00.7
termination on tolfun=1e-11 (Tue Jul 22 17:23:54 2025)
final/bestever f-value = 2.278175e-03 -0.000000e+00 after 4861/166 evaluations
incumbent solution: [ 4.36659246e-09, -1.77783273e+00, -5.77481891e-11, -2.6265
9255e-02, -1.66106948e+01, -5.45952673e+00, -5.29880325e+00]
std deviation: [1.60730813e-09, 1.19486377e-02, 2.32505568e-10, 1.79283384e-02,
1.08040215e-01, 4.86880692e-02, 3.63681540e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:54 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 1.417748891165777e-02 1.0e+00 2.28e-01  2e-01  3e-01 0:00.0
    2     40 3.715446041938934e-03 1.5e+00 2.57e-01  2e-01  3e-01 0:00.0
    3     60 3.860210561463052e-03 1.6e+00 2.71e-01  3e-01  3e-01 0:00.0
/tmp/ipython-input-3-2769142098.py:126: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

```
  100   2000 3.358874652563645e-03 8.8e+03 1.02e-01  4e-05  2e-01 0:00.3
NOTE (module=cma, iteration=105):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.3e+08 to 3.7e+02
  200   4000 -8.795691007668204e-03 8.0e+03 9.70e-02  4e-07  5e-01 0:00.6
  300   6000 -9.509241999129652e-03 4.5e+05 1.41e-01  1e-07  8e-01 0:00.9
NOTE (module=cma, iteration=306):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.0e+11 to 2.0e+08
  400   8000 -9.510879354540732e-03 2.3e+04 2.61e-02  1e-11  1e-01 0:01.2
NOTE (module=cma, iteration=451):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 2.7e+10 to 1.6e+09
  463   9260 -9.510879939580056e-03 4.2e+04 2.11e-03  1e-13  9e-03 0:01.4
termination on tolfunhist=1e-12 (Tue Jul 22 17:23:56 2025)
final/bestever f-value = -9.510880e-03 -9.510880e-03 after 9261/9030 evaluation
s
incumbent solution: [ 1.18906699e-11, -2.04090433e+01, -2.86786806e-11, 2.04644
059e+01, -1.88467383e+01, -2.48250925e+01, -1.39072221e+01]
std deviation: [1.05514550e-13, 9.21551343e-03, 2.48923777e-13, 9.21633681e-03,
3.37987677e-03, 8.71953806e-03, 3.84322029e-03]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
3:56 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.050442713856297e-02 1.0e+00 2.40e-01  2e-01  3e-01 0:00.0
    2     40 -1.276911894272874e-02 1.4e+00 2.87e-01  3e-01  3e-01 0:00.0
    3     60 -1.335322683032443e-02 1.5e+00 3.18e-01  3e-01  4e-01 0:00.0
NOTE (module=cma, iteration=95):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 1.0e+02
  100   2000 -1.383320528489543e-02 9.9e+00 1.36e-01  3e-05  4e-01 0:00.3
NOTE (module=cma, iteration=190):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 9.8e+02
  200   4000 -1.383327205149176e-02 2.9e+01 3.81e-02  4e-09  4e-01 0:00.6
  300   6000 -2.224261121522252e-02 6.9e+03 2.72e-02  9e-12  5e-01 0:00.9
NOTE (module=cma, iteration=336):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 5.1e+08 to 6.6e+03
  400   8000 -2.230326514516330e-02 4.0e+02 2.61e-02  1e-12  5e-01 0:01.2
  500  10000 -2.230329209811342e-02 2.7e+03 1.27e-02  1e-13  2e-01 0:01.5
  600  12000 -2.230329069747900e-02 1.2e+04 4.61e-03  4e-14  5e-02 0:02.0
  700  14000 -2.230329221726020e-02 2.7e+04 4.91e-04  2e-15  2e-03 0:02.4
  800  16000 -2.230329208778087e-02 6.0e+04 1.68e-03  7e-15  3e-03 0:02.8
  850  17000 -2.230329220584974e-02 1.1e+05 3.96e-04  1e-15  6e-04 0:03.1
termination on tolstagnation=192 (Tue Jul 22 17:24:01 2025)
final/bestever f-value = 3.550330e-02 -2.230329e-02 after 17001/9276 evaluation
s
incumbent solution: [-1.53521220e-11, -2.41908886e+00, -2.04197146e-10, 2.53170
325e+00, -3.98411063e+01, -2.88041665e+00, 8.85079572e+00]
std deviation: [1.12438506e-15, 1.84484070e-04, 1.54965335e-14, 1.84806296e-04,
5.72506148e-04, 1.38142529e-04, 3.08490049e-04]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:01 2025)
```

```
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.516089002863638e-03 1.0e+00 2.21e-01  2e-01  2e-01 0:00.0
    2     40 -2.691723303957102e-03 1.3e+00 2.40e-01  2e-01  3e-01 0:00.0
    3     60 -3.133820051550948e-03 1.6e+00 2.62e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 6.0e+01
  100   2000 -3.617957775899247e-03 7.5e+00 1.28e-01  3e-05  3e-01 0:00.4
NOTE (module=cma, iteration=198):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.6e+08 to 1.2e+02
  200   4000 -3.618007074668397e-03 1.4e+01 4.45e-02  3e-09  3e-01 0:00.7
  218   4360 -3.618007080534528e-03 2.0e+01 4.31e-02  7e-10  4e-01 0:00.7
termination on tolfun=1e-11 (Tue Jul 22 17:24:02 2025)
final/bestever f-value = -3.618007e-03 -3.618007e-03 after 4361/4221 evaluation
s
incumbent solution: [-7.80069777e-09, -2.34756652e+00, -1.72146783e-10, 9.00493
436e-01, -8.90169917e+00, 3.39451958e+00, -1.38647277e+01]
std deviation: [4.71309463e-09, 5.79503682e-02, 6.79270009e-10, 6.34888961e-02,
1.53980579e-01, 1.23021906e-01, 3.61390061e-01]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           101.73           1.73           90.78
-9.22   102.85            2.85
0.0002   0.00005            95.01          -4.99           90.78
-9.22    93.70           -6.30
0.0004   0.00010            95.70          -4.30           90.78
-9.22    93.39           -6.61
0.0006   0.00030            97.21          -2.79           90.78
-9.22    96.23           -3.77
```

In [ ]:
```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt
from numba import njit
```

```
from sklearn.preprocessing import RobustScaler

np.random.seed(42)
random_seed = 42

# Load 1-minute data
df = pd.read_csv("ADA_1min.csv", parse_dates=['system_time'], index_col='syste

# Feature engineering with noise reduction
for j in range(15):
    noise_factor = 0.05  # Reduced noise for 1min data
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}'] * (1 + nc
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}'] * (1 + nc

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]

# Enhanced OBI calculation with smoothing
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().rolling(5, min_periods=1).mean().fillna(0)  # 5-
df['depth'] = np.log1p(df[bid_cols + ask_cols].sum(axis=1))
df['queue_slope'] = (df['bids_notional_0'] - df['bids_notional_5']) / (df['bid

# Feature scaling
scaler = RobustScaler()
features = ['obi', 'dobi', 'depth', 'queue_slope']
df[features] = scaler.fit_transform(df[features])

# Adjusted time splits for 1min data (more recent test set)
train_end = int(len(df) * 0.5)  # 50% training
cv_end = int(len(df) * 0.75)    # 25% validation
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

# Returns calculation
for df_part in [df_train, df_cv, df_test]:
    df_part['log_mid'] = np.log(df_part['midpoint'])
    df_part['returns'] = df_part['log_mid'].diff().fillna(0)

# Trading strategy with position smoothing
@njit
def trading_strategy(signal, threshold, volatility):
    positions = np.zeros(len(signal))
    for i in range(1, len(signal)):
        z_score = signal[i] / (volatility[i] + 1e-8)
        if z_score > threshold:
            positions[i] = min(positions[i-1] + 0.05, 1)  # Slower position bu
        elif z_score < -threshold:
            positions[i] = max(positions[i-1] - 0.05, -1)
        else:
            positions[i] = positions[i-1] * 0.98  # Slower decay
    # Manual diff calculation
```

```
    trades = np.zeros(len(positions))
    trades[0] = positions[0]
    for i in range(1, len(positions)):
        trades[i] = positions[i] - positions[i-1]
    return positions, trades

@njit
def apply_trading_costs(positions, trades, returns, fee, slip, volatility):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip * volatility[1:len(positions)][trade_mask]
    net_pnl = raw_pnl - costs
    return net_pnl

@njit
def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2, a3 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1] + a3 * np.tanh(x[t-1]/3.0)  # Sm
        sigma = np.exp(b0 + b1 * np.log1p(np.abs(x[t-1])/10.0) + b2)  # More s
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * np.random.randn()
    return x

def optimize_threshold(signal, returns, fee, slip, volatility):
    thresholds = np.geomspace(0.0005, 0.05, 25)  # Wider range for 1min
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t, volatility)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip, vola
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0

    def objective(params):
        mu_params = params[:4]
        sigma_params = params[4:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, 0.005, np.ones_like(signal)*vol
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip, np
```

```
    res = fmin(objective, [0, -0.3, 0.3, 0.05, -1.5, 0.05, 0.005],
               sigma0=0.15, options={'seed':random_seed, 'popsize':25, 'maxite
    return res[0][:4], res[0][4:]

# Adjusted fee structure for 1min trading
fees = [0, 0.0001, 0.0002, 0.0003]
slippages = [0, 0.00002, 0.00005, 0.0001]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    # Adjusted training segments for 1min (smaller windows)
    train_segments = [(i*500, (i+1)*500) for i in range(6)]  # 500-min (8.3hr)
    segment_models = []
    segment_thresholds = []

    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    # Adjusted window size for 1min (30-min windows)
    window_size = 30
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []

    for start in range(0, len(cv_returns) - window_size + 1, window_size//2):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            volatility = np.sqrt(np.mean(np.diff(signal)**2))
            pos, trades = trading_strategy(signal, segment_thresholds[i], np.c
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    test_volatility = []
```

```
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, threshold, np.ones_like(signal)
        test_positions.append(pos)
        test_trades.append(trades)
        test_volatility.extend([volatility]*len(pos))

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_volatility = np.array(test_volatility[:len(fp_positions)])
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns), len(fp_volatility))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_volatility = fp_volatility[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    npc_returns = fp_positions * bh_returns - (fee + slip * fp_volatility) * (
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee:.4f}, Slippage={slip:.5f}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
```

```
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:06 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.365431808352879e-02 1.0e+00 1.48e-01  1e-01  2e-01 0:04.1
    2     50 -1.936758076394668e-02 1.4e+00 1.29e-01  1e-01  1e-01 0:04.1
    3     75 -1.021499957248044e-02 1.5e+00 1.17e-01  1e-01  1e-01 0:04.1
  100   2500 -1.709353331571612e-02 3.7e+01 4.60e-02  7e-03  5e-02 0:05.6
  150   3750 -1.749436134818990e-02 7.4e+01 6.01e-02  6e-03  6e-02 0:06.7
termination on maxiter=150 (Tue Jul 22 17:24:14 2025)
final/bestever f-value = -1.734501e-02 -2.543328e-02 after 3751/634 evaluations
incumbent solution: [ 0.1533341, -0.23095783, 0.43038381, 0.19997015, -1.457454
26, -0.16361781, -0.47700191]
std deviation: [0.00638229, 0.01740922, 0.01739116, 0.05761631, 0.02313635, 0.0
1429886, 0.0459249, ]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:14 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.355555802662260e-02 1.0e+00 1.37e-01  1e-01  1e-01 0:00.0
    2     50 -1.575738415077563e-02 1.2e+00 1.28e-01  1e-01  1e-01 0:00.0
    3     75 -1.375788459586265e-02 1.4e+00 1.31e-01  1e-01  1e-01 0:00.0
   68   1700 -1.451251960951334e-02 8.6e+02 1.26e-01  4e-03  4e-01 0:00.9
termination on tolflatfitness=1 (Tue Jul 22 17:24:15 2025)
final/bestever f-value = -1.451252e-02 -1.575738e-02 after 1701/28 evaluations
incumbent solution: [ 0.20216683, -0.39404682, 4.70092472, -0.26066381, -4.5590
8997, 1.73708294, -3.56398722]
std deviation: [0.01740791, 0.0039342, 0.39129226, 0.01994415, 0.20533154, 0.19
634937, 0.3590924, ]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:15 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.580264963096686e-02 1.0e+00 1.81e-01  2e-01  2e-01 0:00.0
    2     50 -1.839891821794505e-02 1.4e+00 2.11e-01  2e-01  2e-01 0:00.0
    3     75 -1.150882575183627e-02 1.6e+00 2.49e-01  2e-01  3e-01 0:00.0
   59   1475 -1.601115943622496e-02 1.2e+03 1.47e-01  2e-04  2e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:24:16 2025)
final/bestever f-value = -1.601116e-02 -1.839892e-02 after 1476/41 evaluations
incumbent solution: [ 0.41296524, 0.06768857, 0.35078188, -0.04966325, -0.99935
088, -0.17086782, 0.39405722]
std deviation: [6.77805157e-02, 2.31017059e-04, 1.42367946e-01, 1.81563955e-01,
1.50552361e-01, 2.37163623e-01, 2.35897077e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:16 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.059782182495282e-02 1.0e+00 1.54e-01  1e-01  2e-01 0:00.0
    2     50 -5.129413460403919e-03 1.2e+00 1.60e-01  1e-01  2e-01 0:00.0
    3     75 -9.893004750384491e-03 1.6e+00 1.60e-01  1e-01  2e-01 0:00.0
```

```
/tmp/ipython-input-4-271812216.py:108: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

```
   58   1450 -1.132326954880009e-02 5.5e+02 1.30e-01  4e-04  2e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:24:17 2025)
final/bestever f-value = -1.132327e-02 -1.250433e-02 after 1451/447 evaluations
incumbent solution: [-0.63566922, 0.07264503, 0.64227053, 0.10295902, -1.379446
23, 0.42086458, 0.74697036]
std deviation: [0.11555049, 0.00043224, 0.10274995, 0.11233451, 0.13201783, 0.1
837371, 0.12457919]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:17 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.033564736744372e-02 1.0e+00 1.37e-01  1e-01  1e-01 0:00.0
    2     50 -1.967668081740020e-02 1.3e+00 1.27e-01  1e-01  1e-01 0:00.0
    3     75 -2.111067573420511e-02 1.5e+00 1.25e-01  1e-01  1e-01 0:00.0
  100   2500 -2.420488717748520e-02 4.5e+01 3.11e-01  6e-02  3e-01 0:01.0
  150   3750 -2.379931832210402e-02 1.1e+02 1.87e-01  2e-02  2e-01 0:01.6
termination on maxiter=150 (Tue Jul 22 17:24:19 2025)
final/bestever f-value = -2.258947e-02 -2.822404e-02 after 3751/1001 evaluation
s
incumbent solution: [-0.75365064, -0.29832756, 2.22274213, 0.03647023, -1.66214
81, -0.31274381, -1.13459387]
std deviation: [0.03906392, 0.02151623, 0.11432501, 0.06699883, 0.09163207, 0.1
5130473, 0.10929518]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:19 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.026648528919431e-02 1.0e+00 1.85e-01  2e-01  2e-01 0:00.0
    2     50 -7.326660112845145e-03 1.4e+00 1.81e-01  2e-01  2e-01 0:00.0
    3     75 -9.461442864933516e-03 1.4e+00 1.87e-01  1e-01  2e-01 0:00.0
  100   2500 -1.128504819071571e-02 3.7e+01 2.45e-02  2e-03  2e-02 0:00.9
  150   3750 -1.226889844301061e-02 9.9e+01 8.08e-03  4e-04  5e-03 0:01.4
termination on maxiter=150 (Tue Jul 22 17:24:21 2025)
final/bestever f-value = -7.332833e-03 -1.486670e-02 after 3751/228 evaluations
incumbent solution: [ 0.01498788, -0.90642757, 1.18530063, -0.5294389, -1.09967
117, 0.2963554, 0.82177817]
std deviation: [0.00041313, 0.00196172, 0.00225485, 0.00533443, 0.00453085, 0.0
0441772, 0.0037885, ]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:21 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 3.770758593631118e-03 1.0e+00 1.63e-01  2e-01  2e-01 0:02.0
    2     50 1.332054894409877e-03 1.3e+00 1.82e-01  2e-01  2e-01 0:02.0
    3     75 6.755261966987034e-04 1.7e+00 2.19e-01  2e-01  3e-01 0:02.0
NOTE (module=cma, iteration=95):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 4.6e+01
  100   2500 5.938316066788837e-04 7.7e+00 2.60e-01  3e-05  7e-01 0:02.9
  150   3750 5.938253138517095e-04 5.8e+02 1.94e-01  2e-07  7e-01 0:03.6
termination on maxiter=150 (Tue Jul 22 17:24:26 2025)
final/bestever f-value = 5.938253e-04 5.938253e-04 after 3751/3680 evaluations
incumbent solution: [ 1.33774711e-06, -7.20882147e-01, -6.21766350e-08, 2.16031
916e-01, -7.98175026e+00, 1.05810589e+01, -8.78378344e+00]
std deviation: [1.31946894e-06, 9.37310643e-02, 1.74571767e-07, 2.07340670e-01,
2.89260074e-01, 6.99942820e-01, 4.47481963e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
```

```
4:26 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -4.747335952140776e-03 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     50 -4.760939648881937e-03 1.4e+00 1.98e-01  2e-01  2e-01 0:00.0
    3     75 -4.787320276857382e-03 1.4e+00 2.13e-01  2e-01  2e-01 0:00.0
/tmp/ipython-input-4-271812216.py:108: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
NOTE (module=cma, iteration=86):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 1.2e+02
  100   2500 -4.842687287384228e-03 1.4e+01 1.23e-01  9e-06  4e-01 0:01.4
  150   3750 -4.842689336921165e-03 1.3e+03 9.27e-02  6e-08  6e-01 0:02.2
termination on maxiter=150 (Tue Jul 22 17:24:29 2025)
final/bestever f-value = -4.842689e-03 -4.842689e-03 after 3751/3714 evaluation
s
incumbent solution: [ 5.41824501e-07, -1.66386179e+00, -1.32611078e-08, 5.30246
343e-01, -1.51453341e+01, 2.98371810e+00, -4.04115073e+00]
std deviation: [5.66517503e-07, 8.56881389e-02, 6.01745344e-08, 2.59788283e-01,
5.51585896e-01, 3.13468681e-01, 2.76790939e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:29 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -7.456069161402457e-03 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     50 -7.467482881931515e-03 1.5e+00 2.28e-01  2e-01  3e-01 0:00.0
    3     75 -7.498456248657127e-03 1.5e+00 2.92e-01  2e-01  3e-01 0:00.0
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1452:
RuntimeWarning: invalid value encountered in subtract
  a = op(a[slice1], a[slice2])
```

```
NOTE (module=cma, iteration=89):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 3.3e+01
  100   2500 -7.563217149487111e-03 1.2e+01 2.46e-01  2e-05  1e+00 0:00.9
  150   3750 -7.563219001517178e-03 1.2e+03 6.09e-02  3e-08  3e-01 0:01.2
termination on maxiter=150 (Tue Jul 22 17:24:30 2025)
final/bestever f-value = -7.563219e-03 -7.563219e-03 after 3751/3608 evaluation
s
incumbent solution: [ 2.45713953e-07, -2.44224933e+00, 3.66563538e-09, 4.042874
46e+00, -1.46000622e+01, 3.52486099e+00, -4.14619988e+00]
std deviation: [2.89643741e-07, 6.19642766e-02, 3.12080652e-08, 1.30059610e-01,
3.23401911e-01, 1.22288264e-01, 1.08709328e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:30 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 2.401340992325408e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     50 1.427610447471746e-04 1.4e+00 2.08e-01  2e-01  2e-01 0:00.0
    3     75 1.215289880622255e-03 1.6e+00 2.20e-01  2e-01  3e-01 0:00.0
   21    525 -0.000000000000000e+00 4.7e+00 1.49e+00  1e+00  2e+00 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 17:24:30 2025)
final/bestever f-value = -0.000000e+00 -0.000000e+00 after 526/303 evaluations
incumbent solution: [-1.8372714, 2.75912818, -3.27375199, 1.21399351, -2.383663
94, 4.07030144, 1.15325009]
std deviation: [1.47717411, 1.39204071, 1.95238086, 1.23525495, 1.82526169, 2.0
412208, 1.74221437]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:30 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -4.819291425564191e-03 1.0e+00 1.71e-01  1e-01  2e-01 0:00.0
    2     50 -4.823860879900090e-03 1.5e+00 1.97e-01  2e-01  2e-01 0:00.0
    3     75 -4.853660216289888e-03 1.7e+00 2.32e-01  2e-01  3e-01 0:00.0
/tmp/ipython-input-4-271812216.py:134: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

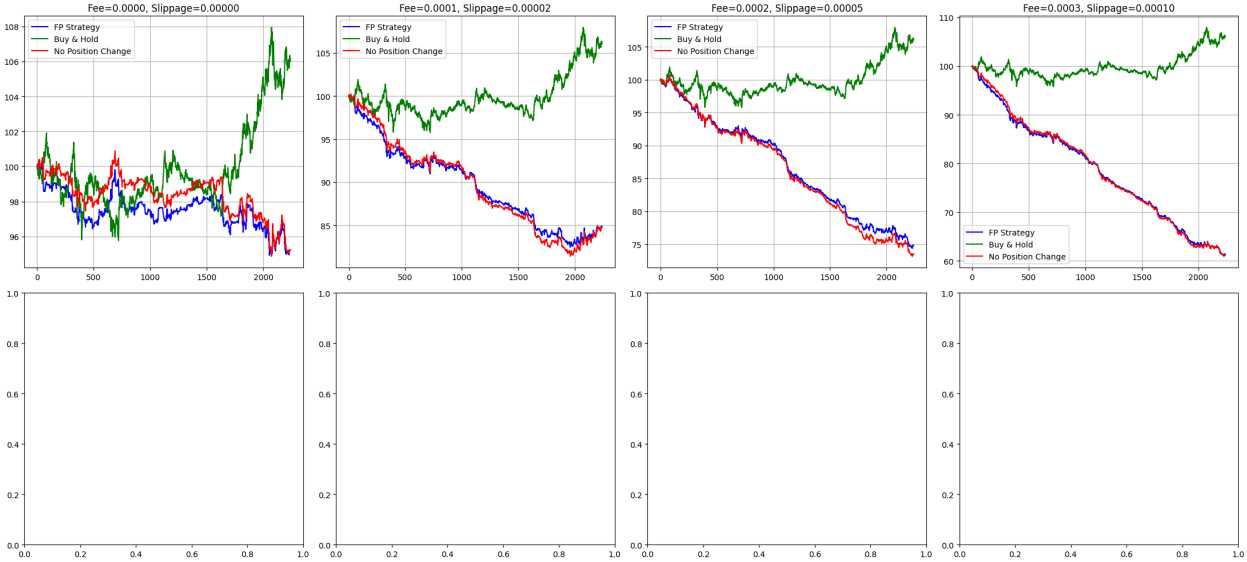
```
NOTE (module=cma, iteration=93):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.8e+08 to 5.9e+01
  100   2500 -4.934858450040668e-03 9.0e+00 1.43e-01  1e-05  4e-01 0:00.5
  150   3750 -4.934861076634326e-03 4.4e+02 1.03e-01  1e-07  3e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:31 2025)
final/bestever f-value = -4.934861e-03 -8.084663e-03 after 3751/165 evaluations
incumbent solution: [ 8.55336748e-07, 4.03393485e-01, 1.53734586e-08, -3.826178
33e+00, -1.16785061e+01, -4.62005159e+00, -6.05332443e+00]
std deviation: [6.72348396e-07, 1.54711154e-01, 1.21612927e-07, 1.65724932e-01,
3.38245980e-01, 2.64490306e-01, 1.76409772e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:31 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 8.229404572109952e-03 1.0e+00 1.66e-01  2e-01  2e-01 0:00.0
    2     50 -1.717204238471659e-06 1.4e+00 1.66e-01  2e-01  2e-01 0:00.0
    3     75 -9.211962656189262e-04 1.5e+00 1.79e-01  2e-01  2e-01 0:00.0
NOTE (module=cma, iteration=88):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 2.0e+02
  100   2500 -1.606939347886145e-04 1.2e+01 4.40e-01  6e-05  1e+00 0:00.5
  150   3750 -1.606965043377973e-04 1.4e+03 1.12e-01  1e-07  8e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:32 2025)
final/bestever f-value = -1.606965e-04 -9.211963e-04 after 3751/64 evaluations
incumbent solution: [ 8.35594999e-07, 1.19206454e+00, 3.30182690e-08, -5.150520
45e+00, 5.44989485e+00, -4.60871703e+00, -2.28175357e+01]
std deviation: [6.34450482e-07, 1.19664980e-01, 1.23316830e-07, 3.11718995e-01,
4.09897580e-01, 2.70862831e-01, 7.53088901e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:32 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 9.903653030125074e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     50 4.540398276550797e-03 1.4e+00 2.22e-01  2e-01  3e-01 0:00.0
    3     75 3.054941660905437e-03 1.5e+00 2.55e-01  2e-01  3e-01 0:00.0
  100   2500 2.593861165409253e-03 7.9e+03 2.34e-01  6e-05  4e-01 0:00.5
NOTE (module=cma, iteration=104):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 9.3e+07 to 4.5e+01
  150   3750 2.593825395731027e-03 3.2e+02 1.70e-01  3e-07  5e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:33 2025)
final/bestever f-value = 2.593825e-03 -2.087502e-02 after 3751/226 evaluations
incumbent solution: [ 2.46829999e-06, 2.21234563e+00, 3.34807910e-08, -7.670866
98e+00, -6.03655228e+00, 6.93186041e+00, -1.01992720e+01]
std deviation: [2.61117729e-06, 1.57123125e-01, 3.42118400e-07, 4.79155147e-01,
1.43241988e-01, 4.09465238e-01, 3.16257406e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:33 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.609948420209150e-03 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     50 -2.654629453253562e-03 1.4e+00 2.14e-01  2e-01  2e-01 0:00.0
    3     75 -2.602833718437992e-03 1.5e+00 2.59e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.7e+08 to 2.1e+02
```

```
  100   2500 -2.842681771136158e-03 1.3e+01 1.50e-01  3e-05  3e-01 0:00.5
  150   3750 -2.842689306815960e-03 4.0e+02 7.43e-02  1e-07  3e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:34 2025)
final/bestever f-value = -2.842689e-03 -3.249143e-03 after 3751/2401 evaluation
s
incumbent solution: [ 7.31881297e-07, -2.97271040e+00, 9.61723231e-09, 7.741905
05e+00, -8.11641216e+00, 4.98369535e+00, -9.21205694e+00]
std deviation: [5.39379994e-07, 8.13735372e-02, 1.18470090e-07, 1.85792063e-01,
1.27324001e-01, 1.63827410e-01, 2.74108375e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:34 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -5.303909527718118e-03 1.0e+00 1.68e-01  2e-01  2e-01 0:00.0
    2     50 -5.346906947737599e-03 1.5e+00 1.73e-01  2e-01  2e-01 0:00.0
    3     75 -5.422011598283047e-03 1.5e+00 2.31e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=90):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 3.1e+01
  100   2500 -5.563209686717228e-03 5.3e+00 1.84e-01  2e-05  4e-01 0:00.4
  150   3750 -5.563218937772216e-03 4.4e+02 2.39e-01  2e-07  5e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:35 2025)
final/bestever f-value = -5.563219e-03 -5.563219e-03 after 3751/3737 evaluation
s
incumbent solution: [ 1.20656245e-06, -2.16254332e+00, -5.05865223e-08, 5.70499
197e+00, -1.91352921e+01, -7.04680710e+00, 2.31746177e+00]
std deviation: [1.38154183e-06, 1.00405631e-01, 2.48176833e-07, 2.80917969e-01,
5.41105345e-01, 3.24977811e-01, 3.31929486e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:35 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 8.521727052534048e-03 1.0e+00 1.67e-01  2e-01  2e-01 0:00.0
    2     50 3.327678968904558e-03 1.4e+00 1.79e-01  2e-01  2e-01 0:00.0
    3     75 3.315659217692508e-03 1.5e+00 1.88e-01  2e-01  2e-01 0:00.0
   14    350 -0.000000000000000e+00 3.7e+00 2.00e+00  1e+00  2e+00 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 17:24:35 2025)
final/bestever f-value = -0.000000e+00 -0.000000e+00 after 351/198 evaluations
incumbent solution: [ 0.49297908, 3.33089118, 1.98924006, 1.75450067, -1.681691
57, 2.59184626, 1.00919918]
std deviation: [2.24497308, 2.13279779, 1.30572028, 1.96194412, 2.1049182, 2.27
090876, 1.84984156]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:35 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.679081603143499e-03 1.0e+00 1.81e-01  2e-01  2e-01 0:00.0
    2     50 -2.733876150085199e-03 1.5e+00 2.16e-01  2e-01  2e-01 0:00.0
    3     75 -2.814523965496698e-03 1.6e+00 2.61e-01  2e-01  3e-01 0:00.0
/usr/local/lib/python3.11/dist-packages/numpy/_core/_methods.py:127: RuntimeWar
ning: overflow encountered in reduce
  ret = umr_sum(arr, axis, dtype, out, keepdims, where=where)
```

```
NOTE (module=cma, iteration=88):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 3.3e+01
  100   2500 -2.934848322968820e-03 6.8e+00 1.57e-01  1e-05  4e-01 0:00.4
  150   3750 -2.934861051101724e-03 7.7e+02 1.10e-01  1e-07  5e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:36 2025)
final/bestever f-value = -2.934861e-03 -2.934861e-03 after 3751/3714 evaluation
s
incumbent solution: [ 6.16582863e-07, -1.68140405e+00, 3.97894863e-08, 4.373490
45e+00, 2.49071029e+00, 3.53320087e-01, -1.99058064e+01]
std deviation: [7.52457044e-07, 8.09852085e-02, 1.38277268e-07, 1.31948210e-01,
1.75593307e-01, 3.55598575e-01, 5.24890118e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:36 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 1.152879105654605e-02 1.0e+00 1.62e-01  2e-01  2e-01 0:00.0
    2     50 6.387950010511971e-03 1.3e+00 1.71e-01  1e-01  2e-01 0:00.0
    3     75 2.514944726827338e-03 1.6e+00 1.79e-01  1e-01  2e-01 0:00.0
NOTE (module=cma, iteration=98):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 3.6e+01
  100   2500 1.839315089321454e-03 6.6e+00 1.50e-01  2e-05  3e-01 0:00.5
  150   3750 1.839303513026510e-03 2.8e+02 6.24e-02  9e-08  1e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:37 2025)
final/bestever f-value = 1.839304e-03 1.839304e-03 after 3751/3729 evaluations
incumbent solution: [ 6.17332343e-07, -9.52420214e-01, -2.19565719e-08, 8.63975
342e-01, -7.83004708e+00, -8.64758770e+00, -9.33791935e+00]
std deviation: [7.95281834e-07, 1.86867954e-02, 9.43815500e-08, 1.47418646e-01,
1.42219725e-01, 1.31376831e-01, 1.32473822e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:37 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 1.481340131856787e-02 1.0e+00 1.61e-01  1e-01  2e-01 0:00.0
    2     50 5.222768611376376e-03 1.3e+00 1.77e-01  2e-01  2e-01 0:00.0
    3     75 6.569551021391194e-03 1.5e+00 1.93e-01  2e-01  2e-01 0:00.0
NOTE (module=cma, iteration=90):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.5e+08 to 3.1e+01
  100   2500 4.593845428052858e-03 5.7e+00 1.93e-01  2e-05  5e-01 0:00.5
  150   3750 4.593825367295207e-03 1.1e+03 1.01e-01  9e-08  4e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:38 2025)
final/bestever f-value = 4.593825e-03 4.593825e-03 after 3751/3749 evaluations
incumbent solution: [ 9.18280884e-07, 1.85971395e+00, 1.23973956e-08, -8.997455
49e+00, -3.24069385e+00, -8.60554536e+00, -1.40848658e+01]
std deviation: [8.36511756e-07, 9.43876836e-02, 9.04455175e-08, 3.61425783e-01,
1.02833972e-01, 4.08833756e-01, 3.79794436e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:38 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -3.635482829102941e-04 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     50 -4.250547828622327e-04 1.4e+00 1.93e-01  2e-01  2e-01 0:00.0
    3     75 -4.103834074424863e-04 1.5e+00 2.49e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=91):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
```

```
condition changed from 1.2e+08 to 1.6e+01
  100   2500 -8.426803486902487e-04 6.0e+00 1.60e-01  1e-05  4e-01 0:00.5
  150   3750 -8.426892451392429e-04 6.8e+02 1.61e-01  1e-07  8e-01 0:00.8
termination on maxiter=150 (Tue Jul 22 17:24:39 2025)
final/bestever f-value = -8.426892e-04 -8.426893e-04 after 3751/3702 evaluation
s
incumbent solution: [ 1.14226011e-06, -1.45770994e+00, 7.69863989e-09, 8.802676
96e-01, -1.39986858e+01, -2.22910577e+00, -4.67713118e+00]
std deviation: [7.22224383e-07, 1.08153322e-01, 1.40610395e-07, 2.18891556e-01,
7.51804915e-01, 2.55762833e-01, 2.55830535e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:39 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -3.035261879332785e-03 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     50 -3.130576759130083e-03 1.5e+00 2.09e-01  2e-01  2e-01 0:00.0
    3     75 -3.248875128586463e-03 1.6e+00 2.54e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=91):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.1e+08 to 1.5e+01
  100   2500 -3.563187959900993e-03 4.8e+00 3.98e-01  3e-05  8e-01 0:00.7
  150   3750 -3.563218885605041e-03 6.1e+02 1.75e-01  1e-07  5e-01 0:01.0
termination on maxiter=150 (Tue Jul 22 17:24:40 2025)
final/bestever f-value = -3.563219e-03 -3.563219e-03 after 3751/3750 evaluation
s
incumbent solution: [ 1.05308648e-06, -1.20568405e+00, -9.46808823e-10, 1.20684
611e+00, -5.77116107e+00, 6.84081573e+00, -1.13437217e+01]
std deviation: [8.22116654e-07, 1.00138802e-01, 1.48582994e-07, 1.11923790e-01,
4.71351256e-01, 2.70324571e-01, 1.91539871e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:40 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 1.598371421348892e-02 1.0e+00 1.72e-01  2e-01  2e-01 0:00.0
    2     50 5.406962194587576e-03 1.4e+00 2.02e-01  2e-01  2e-01 0:00.0
    3     75 5.400767021847948e-03 1.6e+00 2.20e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=91):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 2.0e+01
  100   2500 5.116431150071793e-03 5.7e+00 2.10e-01  2e-05  6e-01 0:00.6
  150   3750 5.116415549030452e-03 8.1e+02 1.03e-01  9e-08  4e-01 0:01.0
termination on maxiter=150 (Tue Jul 22 17:24:41 2025)
final/bestever f-value = 5.116416e-03 5.116416e-03 after 3751/3729 evaluations
incumbent solution: [-5.12099406e-07, -2.15068433e+00, -1.88141964e-09, 3.12958
930e+00, -9.04869895e+00, 1.29356797e+00, -8.86828671e+00]
std deviation: [8.35002351e-07, 7.29431655e-02, 8.55335160e-08, 2.25350346e-01,
2.01978437e-01, 3.58354133e-01, 1.84879847e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:41 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -4.301063952168779e-04 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     50 -5.384086413010910e-04 1.4e+00 1.97e-01  2e-01  2e-01 0:00.0
    3     75 -6.060008672262134e-04 1.5e+00 2.28e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=89):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.8e+08 to 3.6e+01
```

```
  100   2500 -9.348546933383856e-04 9.9e+00 1.16e-01  8e-06  4e-01 0:00.5
  150   3750 -9.348610617165420e-04 8.5e+02 5.08e-02  4e-08  2e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:42 2025)
final/bestever f-value = -9.348610e-04 -9.348611e-04 after 3751/3750 evaluation
s
incumbent solution: [ 3.87381725e-07, -2.16766350e+00, -3.90800861e-09, 2.13662
939e+00, -7.99071775e+00, 3.28469714e+00, -1.06783419e+01]
std deviation: [3.63800841e-07, 2.93108098e-02, 3.76952247e-08, 5.82529959e-02,
1.17761541e-01, 1.09703591e-01, 2.48660895e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
4:42 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 1.993226418590004e-02 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     50 8.630850617006585e-03 1.4e+00 1.86e-01  2e-01  2e-01 0:00.0
    3     75 4.367251602044271e-03 1.5e+00 1.88e-01  2e-01  2e-01 0:00.0
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.7e+08 to 1.1e+02
  100   2500 3.839322296171838e-03 1.1e+01 1.67e-01  3e-05  6e-01 0:00.5
  150   3750 3.839303588350737e-03 3.0e+02 4.73e-02  1e-07  2e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:24:43 2025)
final/bestever f-value = 3.839304e-03 3.839304e-03 after 3751/3680 evaluations
incumbent solution: [ 8.03353768e-07, 1.24044709e-01, 7.55652252e-10, -3.107101
24e+00, -7.72911134e+00, 8.98935507e-01, -9.22342092e+00]
std deviation: [7.33274374e-07, 4.75432210e-02, 9.89991832e-08, 1.29257027e-01,
5.89111848e-02, 3.07187886e-02, 1.79751314e-01]
```



Final Portfolio Values and Returns for Different Fee/Slippage Configurations:

Fee	Slippage	FP Strategy ($)	FP Return (%)	Buy & Hold ($)	Buy & Hold Return (%)	NPC ($)	NPC Return (%)
0.0000	0.00000	95.22	-4.78	106.15	6.15	95.24	-4.76
0.0001	0.00002	84.89	-15.11	106.15	6.15	84.76	-15.24
0.0002	0.00005	74.81	-25.19	106.15	6.15	73.47	-26.53
0.0003	0.00010	61.37	-38.63	106.15	6.15	61.17	-38.83

In [ ]:

```python
import pandas as pd
import numpy as np
from skopt import gp_minimize
from skopt.space import Real
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from typing import Tuple, List, Dict

# Configuration
class Config:
    RANDOM_SEED = 42
    TRAIN_RATIO = 0.6
    CV_RATIO = 0.2
    TEST_RATIO = 0.2
    INITIAL_CAPITAL = 100
    FEE_SLIPPAGE_COMBOS = [
        (0, 0),
        (0.0002, 0.00005),
        (0.0004, 0.0001),
        (0.0006, 0.0003)
    ]
    WINDOW_SIZE = 3
    N_MODEL_SEGMENTS = 5

np.random.seed(Config.RANDOM_SEED)

# Data Preparation
def prepare_data(filepath: str) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFr
    """Load and preprocess the data"""
    df = pd.read_csv(filepath)

    # Calculate price levels
    for j in range(15):
        df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
        df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

    # Calculate features
    bid_cols = [f"bids_notional_{i}" for i in range(15)]
    ask_cols = [f"asks_notional_{i}" for i in range(15)]

    df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (
        df[bid_cols].sum(axis=1) + df[ask_cols].sum(axis=1) + 1e-8)
    df['dobi'] = df['obi'].diff().fillna(0)
    df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
    df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
    df['spread'] = df['ask_price_0'] - df['bid_price_0']

    # Log returns
    df['log_mid'] = np.log(df['midpoint'])
    df['returns'] = df['log_mid'].diff().fillna(0)

    # Train/Validation/Test split
    train_end = int(len(df) * Config.TRAIN_RATIO)
```

```
    cv_end = int(len(df) * (Config.TRAIN_RATIO + Config.CV_RATIO))

    df_train = df.iloc[:train_end].copy().reset_index(drop=True)
    df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
    df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

    # Feature scaling
    scaler = StandardScaler()
    scale_cols = ['obi', 'depth', 'queue_slope', 'spread']
    df_train[scale_cols] = scaler.fit_transform(df_train[scale_cols])
    df_cv[scale_cols] = scaler.transform(df_cv[scale_cols])
    df_test[scale_cols] = scaler.transform(df_test[scale_cols])

    return df_train, df_cv, df_test

# Trading Strategy Components
def trading_strategy(signal: np.ndarray, threshold: float) -> Tuple[np.ndarray
    """Generate positions from trading signals"""
    positions = np.zeros_like(signal)
    positions[signal > threshold] = 1
    positions[signal < -threshold] = -1
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(
    positions: np.ndarray,
    trades: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float,
    trade_sizes: np.ndarray = None
) -> np.ndarray:
    """Calculate PnL with realistic trading costs"""
    raw_pnl = positions[:-1] * returns[1:len(positions)]

    # Dynamic slippage based on trade size and liquidity
    if trade_sizes is None:
        costs = np.abs(trades[1:len(positions)]) * (fee + slip)
    else:
        liquidity_impact = 0.0001 * (trade_sizes / 1e6)  # Assume liquidity in
        costs = np.abs(trades[1:len(positions)]) * (fee + slip + liquidity_imp

    return raw_pnl - costs

# Signal Generation Model
def simulate_fp(
    mu_params: List[float],
    sigma_params: List[float],
    x0: float,
    obi: np.ndarray,
    timesteps: int,
    dt: float = 1.0
) -> np.ndarray:
```

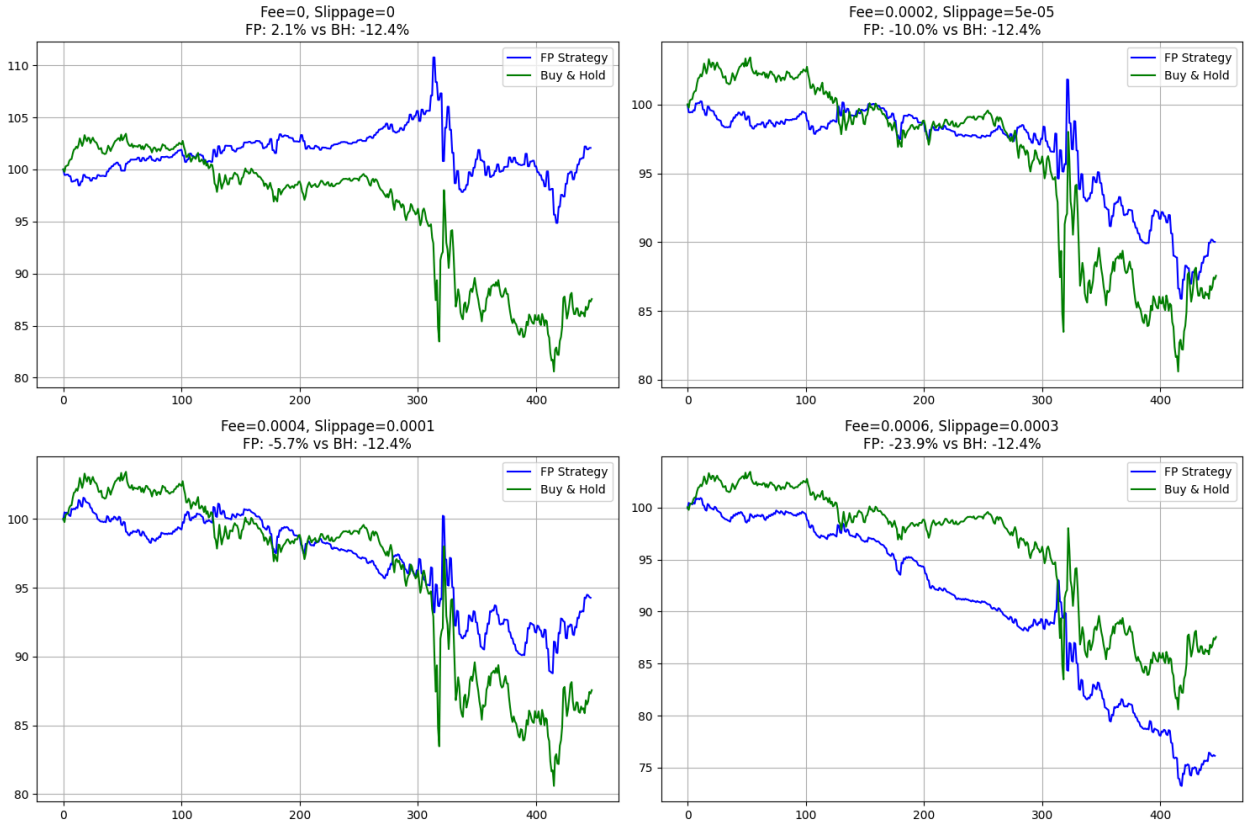
```
    """Fokker-Planck inspired signal generation"""
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params

    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(Config.RANDOM_SEED)

    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()

    return x

# Optimization
def optimize_threshold(
    signal: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float
) -> float:
    """Find optimal trading threshold"""
    thresholds = np.linspace(0.001, 0.01, 20)
    best_pnl = -np.inf
    best_thresh = 0.005

    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))

        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t

    return best_thresh

def train_fp_model(
    df_slice: pd.DataFrame,
    fee: float,
    slip: float
) -> Tuple[List[float], List[float]]:
    """Train model using Bayesian optimization"""
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0

    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
```

```
    space = [
        Real(-1, 1, name='a0'),
        Real(-1, 1, name='a1'),
        Real(-1, 1, name='a2'),
        Real(0.0001, 0.1, name='b0'),
        Real(0.0001, 0.1, name='b1')
    ]

    res = gp_minimize(objective, space, n_calls=50, random_state=Config.RANDOM
    return res.x[:3], res.x[3:]

# Backtest Framework
def run_backtest(
    df_train: pd.DataFrame,
    df_cv: pd.DataFrame,
    df_test: pd.DataFrame,
    fee: float,
    slip: float
) -> Dict:
    """Complete backtest pipeline for one fee/slippage combo"""
    # 1. Train multiple models on different segments
    segment_size = len(df_train) // Config.N_MODEL_SEGMENTS
    segment_models = []
    segment_thresholds = []

    for i in range(Config.N_MODEL_SEGMENTS):
        start = i * segment_size
        end = (i + 1) * segment_size
        if end > len(df_train):
            continue

        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             df_train.iloc[start:end]['obi'].values,
                             end - start)
        threshold = optimize_threshold(signal,
                                       df_train.iloc[start:end]['returns'].value
                                       fee, slip)
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    # 2. Model selection on CV data
    selected_models = []
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values

    for start in range(0, len(cv_returns) - Config.WINDOW_SIZE, Config.WINDOW_
        end = start + Config.WINDOW_SIZE
        best_pnl = -np.inf
        best_index = 0

        for i, (mu_p, sigma_p) in enumerate(segment_models):
```

```
            signal = simulate_fp(mu_p, sigma_p, 0.0,
                                 cv_obi[start:end],
                                 Config.WINDOW_SIZE)
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades,
                                             cv_returns[start:end],
                                             fee, slip))
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i

        selected_models.append(best_index)

    # 3. Test on out-of-sample data
    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []

    for i, start in enumerate(range(0, len(test_returns) - Config.WINDOW_SIZE
        end = start + Config.WINDOW_SIZE
        model_idx = selected_models[min(i, len(selected_models) - 1)]
        mu_p, sigma_p = segment_models[model_idx]
        threshold = segment_thresholds[model_idx]

        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             test_obi[start:end],
                             min(Config.WINDOW_SIZE, len(test_returns) - start))
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    # Combine results
    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    # Calculate PnLs
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(bh_returns))

    # Calculate metrics
    def calculate_metrics(returns):
        total_return = (np.exp(np.sum(returns)) - 1) * 100
        sharpe = np.mean(returns) / np.std(returns) * np.sqrt(365*24*12)  # 5m
```

```
        max_drawdown = (np.exp(np.min(returns.cumsum())) - 1) * 100
        return total_return, sharpe, max_drawdown

    fp_metrics = calculate_metrics(fp_net_returns)
    bh_metrics = calculate_metrics(bh_returns)

    return {
        'fee': fee,
        'slippage': slip,
        'fp_pnl': fp_pnl,
        'bh_pnl': bh_pnl,
        'fp_return_pct': fp_metrics[0],
        'fp_sharpe': fp_metrics[1],
        'fp_drawdown_pct': fp_metrics[2],
        'bh_return_pct': bh_metrics[0],
        'bh_sharpe': bh_metrics[1],
        'bh_drawdown_pct': bh_metrics[2]
    }

# Main Execution
if __name__ == "__main__":
    # Load and prepare data
    df_train, df_cv, df_test = prepare_data("ADA_5min.csv")

    # Run backtests for all fee/slippage combinations
    results = []
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    axes = axes.flatten()

    for idx, (fee, slip) in enumerate(Config.FEE_SLIPPAGE_COMBOS):
        result = run_backtest(df_train, df_cv, df_test, fee, slip)
        results.append(result)

        # Plotting
        ax = axes[idx]
        ax.plot(result['fp_pnl'], label='FP Strategy', color='blue')
        ax.plot(result['bh_pnl'], label='Buy & Hold', color='green')
        ax.set_title(f"Fee={fee}, Slippage={slip}\n"
                     f"FP: {result['fp_return_pct']:.1f}% vs BH: {result['bh_re
        ax.grid(True)
        ax.legend()

    plt.tight_layout()
    plt.show()

    # Results table
    results_df = pd.DataFrame([{
        'Fee': r['fee'],
        'Slippage': r['slippage'],
        'FP Return (%)': r['fp_return_pct'],
        'FP Sharpe': r['fp_sharpe'],
        'FP Drawdown (%)': r['fp_drawdown_pct'],
        'BH Return (%)': r['bh_return_pct'],
```

```
        'BH Sharpe': r['bh_sharpe'],
        'BH Drawdown (%)': r['bh_drawdown_pct']
    } for r in results])

    print("\nPerformance Metrics Across Different Cost Scenarios:")
    print(results_df.to_string(index=False, float_format="%.2f"))
```




```
Performance Metrics Across Different Cost Scenarios:
 Fee  Slippage  FP Return (%)  FP Sharpe  FP Drawdown (%)  BH Return (%)  BH Sh
arpe  BH Drawdown (%)
0.00      0.00           2.07       2.55            -5.13         -12.44      -1
1.01           -19.40
0.00      0.00          -9.98     -13.07           -14.12         -12.44      -1
1.01           -19.40
0.00      0.00          -5.73      -7.32           -11.22         -12.44      -1
1.01           -19.40
0.00      0.00         -23.87     -33.85           -26.76         -12.44      -1
1.01           -19.40
```

In [ ]:

```
import pandas as pd
import numpy as np
from skopt import gp_minimize
from skopt.space import Real
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from typing import Tuple, List, Dict

# Configuration
class Config:
```

```
    RANDOM_SEED = 42
    TRAIN_RATIO = 0.6
    CV_RATIO = 0.2
    TEST_RATIO = 0.2
    INITIAL_CAPITAL = 100
    FEE_SLIPPAGE_COMBOS = [
        (0, 0),
        (0.0002, 0.00005),
        (0.0004, 0.0001),
        (0.0006, 0.0003)
    ]
    WINDOW_SIZE = 3
    N_MODEL_SEGMENTS = 5

np.random.seed(Config.RANDOM_SEED)

# Data Preparation
def prepare_data(filepath: str) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFr
    """Load and preprocess the data"""
    df = pd.read_csv(filepath)

    # Calculate price levels
    for j in range(15):
        df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
        df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

    # Calculate features
    bid_cols = [f"bids_notional_{i}" for i in range(15)]
    ask_cols = [f"asks_notional_{i}" for i in range(15)]

    df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (
        df[bid_cols].sum(axis=1) + df[ask_cols].sum(axis=1) + 1e-8)
    df['dobi'] = df['obi'].diff().fillna(0)
    df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
    df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
    df['spread'] = df['ask_price_0'] - df['bid_price_0']

    # Log returns
    df['log_mid'] = np.log(df['midpoint'])
    df['returns'] = df['log_mid'].diff().fillna(0)

    # Train/Validation/Test split
    train_end = int(len(df) * Config.TRAIN_RATIO)
    cv_end = int(len(df) * (Config.TRAIN_RATIO + Config.CV_RATIO))

    df_train = df.iloc[:train_end].copy().reset_index(drop=True)
    df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
    df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

    # Feature scaling
    scaler = StandardScaler()
    scale_cols = ['obi', 'depth', 'queue_slope', 'spread']
    df_train[scale_cols] = scaler.fit_transform(df_train[scale_cols])
```

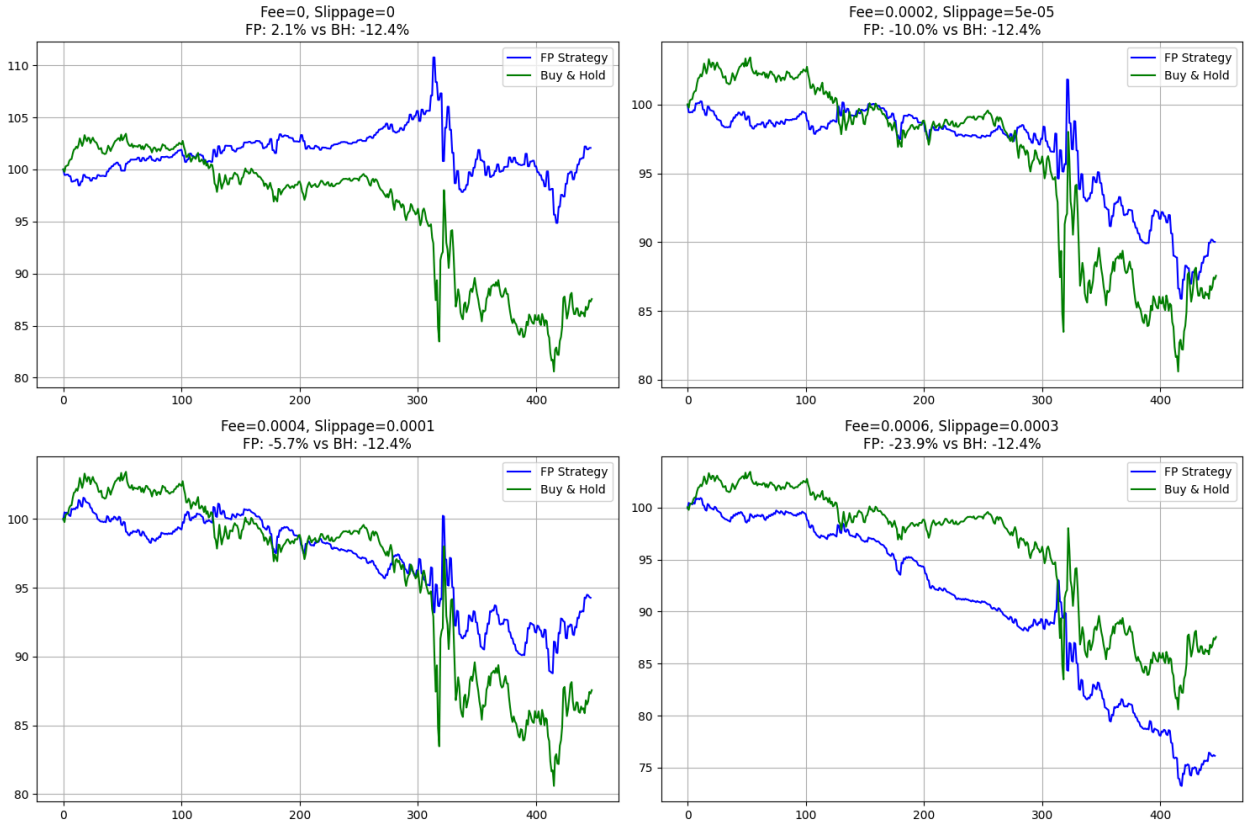
```python
    df_cv[scale_cols] = scaler.transform(df_cv[scale_cols])
    df_test[scale_cols] = scaler.transform(df_test[scale_cols])

    return df_train, df_cv, df_test

# Trading Strategy Components
def trading_strategy(signal: np.ndarray, threshold: float) -> Tuple[np.ndarray
    """Generate positions from trading signals"""
    positions = np.zeros_like(signal)
    positions[signal > threshold] = 1
    positions[signal < -threshold] = -1
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(
    positions: np.ndarray,
    trades: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float,
    trade_sizes: np.ndarray = None
) -> np.ndarray:
    """Calculate PnL with realistic trading costs"""
    raw_pnl = positions[:-1] * returns[1:len(positions)]

    # Dynamic slippage based on trade size and liquidity
    if trade_sizes is None:
        costs = np.abs(trades[1:len(positions)]) * (fee + slip)
    else:
        liquidity_impact = 0.0001 * (trade_sizes / 1e6)  # Assume liquidity ir
        costs = np.abs(trades[1:len(positions)]) * (fee + slip + liquidity_imp

    return raw_pnl - costs

# Signal Generation Model
def simulate_fp(
    mu_params: List[float],
    sigma_params: List[float],
    x0: float,
    obi: np.ndarray,
    timesteps: int,
    dt: float = 1.0
) -> np.ndarray:
    """Fokker-Planck inspired signal generation"""
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params

    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(Config.RANDOM_SEED)

    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
```

```
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()

    return x

# Optimization
def optimize_threshold(
    signal: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float
) -> float:
    """Find optimal trading threshold"""
    thresholds = np.linspace(0.001, 0.01, 20)
    best_pnl = -np.inf
    best_thresh = 0.005

    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))

        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t

    return best_thresh

def train_fp_model(
    df_slice: pd.DataFrame,
    fee: float,
    slip: float
) -> Tuple[List[float], List[float]]:
    """Train model using Bayesian optimization"""
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0

    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))

    space = [
        Real(-1, 1, name='a0'),
        Real(-1, 1, name='a1'),
        Real(-1, 1, name='a2'),
        Real(0.0001, 0.1, name='b0'),
        Real(0.0001, 0.1, name='b1')
    ]

    res = gp_minimize(objective, space, n_calls=50, random_state=Config.RANDOM
```

```
    return res.x[:3], res.x[3:]

# Backtest Framework
def run_backtest(
    df_train: pd.DataFrame,
    df_cv: pd.DataFrame,
    df_test: pd.DataFrame,
    fee: float,
    slip: float
) -> Dict:
    """Complete backtest pipeline for one fee/slippage combo"""
    # 1. Train multiple models on different segments
    segment_size = len(df_train) // Config.N_MODEL_SEGMENTS
    segment_models = []
    segment_thresholds = []

    for i in range(Config.N_MODEL_SEGMENTS):
        start = i * segment_size
        end = (i + 1) * segment_size
        if end > len(df_train):
            continue

        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             df_train.iloc[start:end]['obi'].values,
                             end - start)
        threshold = optimize_threshold(signal,
                                       df_train.iloc[start:end]['returns'].value
                                       fee, slip)
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    # 2. Model selection on CV data
    selected_models = []
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values

    for start in range(0, len(cv_returns) - Config.WINDOW_SIZE, Config.WINDOW_
        end = start + Config.WINDOW_SIZE
        best_pnl = -np.inf
        best_index = 0

        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0,
                                 cv_obi[start:end],
                                 Config.WINDOW_SIZE)
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades,
                                             cv_returns[start:end],
                                             fee, slip))
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
```

```
        selected_models.append(best_index)

    # 3. Test on out-of-sample data
    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []

    for i, start in enumerate(range(0, len(test_returns) - Config.WINDOW_SIZE
        end = start + Config.WINDOW_SIZE
        model_idx = selected_models[min(i, len(selected_models) - 1)]
        mu_p, sigma_p = segment_models[model_idx]
        threshold = segment_thresholds[model_idx]

        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             test_obi[start:end],
                             min(Config.WINDOW_SIZE, len(test_returns) - start))
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    # Combine results
    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    # Calculate PnLs
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(bh_returns))

    # Calculate metrics
    def calculate_metrics(returns):
        total_return = (np.exp(np.sum(returns)) - 1) * 100
        sharpe = np.mean(returns) / np.std(returns) * np.sqrt(365*24*12)  # 5m
        max_drawdown = (np.exp(np.min(returns.cumsum())) - 1) * 100
        return total_return, sharpe, max_drawdown

    fp_metrics = calculate_metrics(fp_net_returns)
    bh_metrics = calculate_metrics(bh_returns)

    return {
        'fee': fee,
        'slippage': slip,
        'fp_pnl': fp_pnl,
```

```
        'bh_pnl': bh_pnl,
        'fp_return_pct': fp_metrics[0],
        'fp_sharpe': fp_metrics[1],
        'fp_drawdown_pct': fp_metrics[2],
        'bh_return_pct': bh_metrics[0],
        'bh_sharpe': bh_metrics[1],
        'bh_drawdown_pct': bh_metrics[2]
    }

# Main Execution
if __name__ == "__main__":
    # Load and prepare data
    df_train, df_cv, df_test = prepare_data("ADA_5min.csv")

    # Run backtests for all fee/slippage combinations
    results = []
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    axes = axes.flatten()

    for idx, (fee, slip) in enumerate(Config.FEE_SLIPPAGE_COMBOS):
        result = run_backtest(df_train, df_cv, df_test, fee, slip)
        results.append(result)

        # Plotting
        ax = axes[idx]
        ax.plot(result['fp_pnl'], label='FP Strategy', color='blue')
        ax.plot(result['bh_pnl'], label='Buy & Hold', color='green')
        ax.set_title(f"Fee={fee}, Slippage={slip}\n"
                     f"FP: {result['fp_return_pct']:.1f}% vs BH: {result['bh_re
        ax.grid(True)
        ax.legend()

    plt.tight_layout()
    plt.show()

    # Results table
    results_df = pd.DataFrame([{
        'Fee': r['fee'],
        'Slippage': r['slippage'],
        'FP Return (%)': r['fp_return_pct'],
        'FP Sharpe': r['fp_sharpe'],
        'FP Drawdown (%)': r['fp_drawdown_pct'],
        'BH Return (%)': r['bh_return_pct'],
        'BH Sharpe': r['bh_sharpe'],
        'BH Drawdown (%)': r['bh_drawdown_pct']
    } for r in results])

    print("\nPerformance Metrics Across Different Cost Scenarios:")
    print(results_df.to_string(index=False, float_format="%.2f"))
```




```
Performance Metrics Across Different Cost Scenarios:
 Fee  Slippage  FP Return (%)  FP Sharpe  FP Drawdown (%)  BH Return (%)  BH Sh
arpe  BH Drawdown (%)
0.00      0.00           2.07       2.55            -5.13         -12.44      -1
1.01           -19.40
0.00      0.00          -9.98     -13.07           -14.12         -12.44      -1
1.01           -19.40
0.00      0.00          -5.73      -7.32           -11.22         -12.44      -1
1.01           -19.40
0.00      0.00         -23.87     -33.85           -26.76         -12.44      -1
1.01           -19.40
```

In [ ]:

```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
```

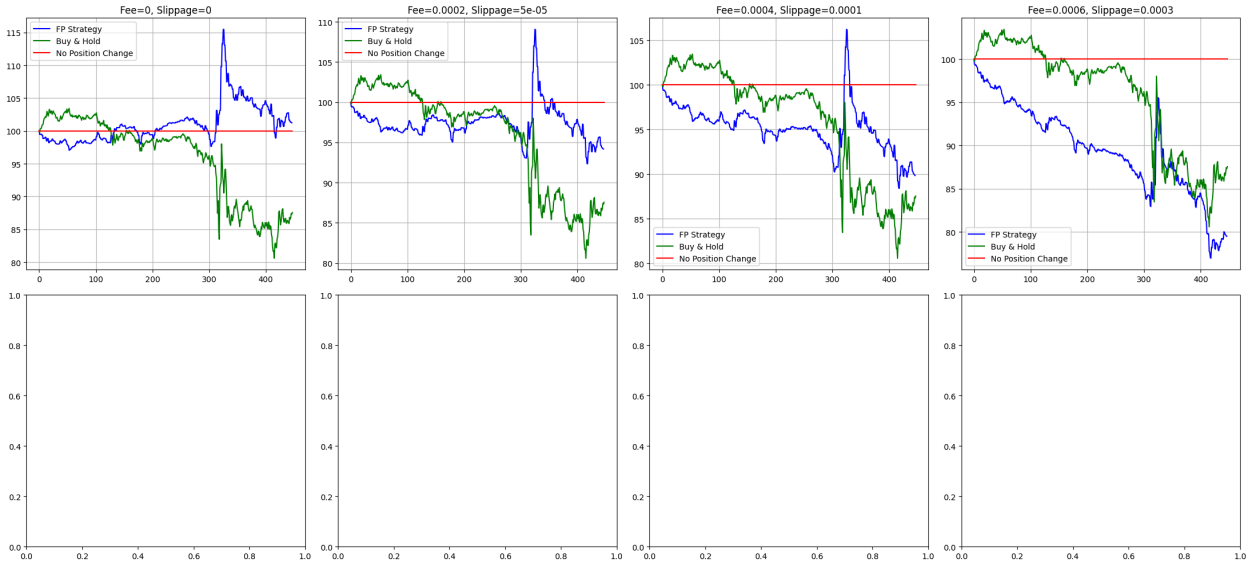
```
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi_t = features['obi'].iloc[t-1]
        dobi_t = features['dobi'].iloc[t-1]
        depth_t = features['depth'].iloc[t-1]
        slope_t = features['queue_slope'].iloc[t-1]
        spread_t = features['spread'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
```

```
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:7]
        sigma_params = params[7:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:7], res[0][7:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:enc
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
```

```
    test_features = df_test[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
```

```
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-3-907504385.py:21: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           101.25           1.25           87.56
-12.44    100.0            0.0
0.0002   0.00005            94.18          -5.82           87.56
-12.44    100.0            0.0
0.0004   0.00010            89.85         -10.15           87.56
-12.44    100.0            0.0
0.0006   0.00030            79.48         -20.52           87.56
-12.44    100.0            0.0
```

In [ ]:

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_1min.csv")
```

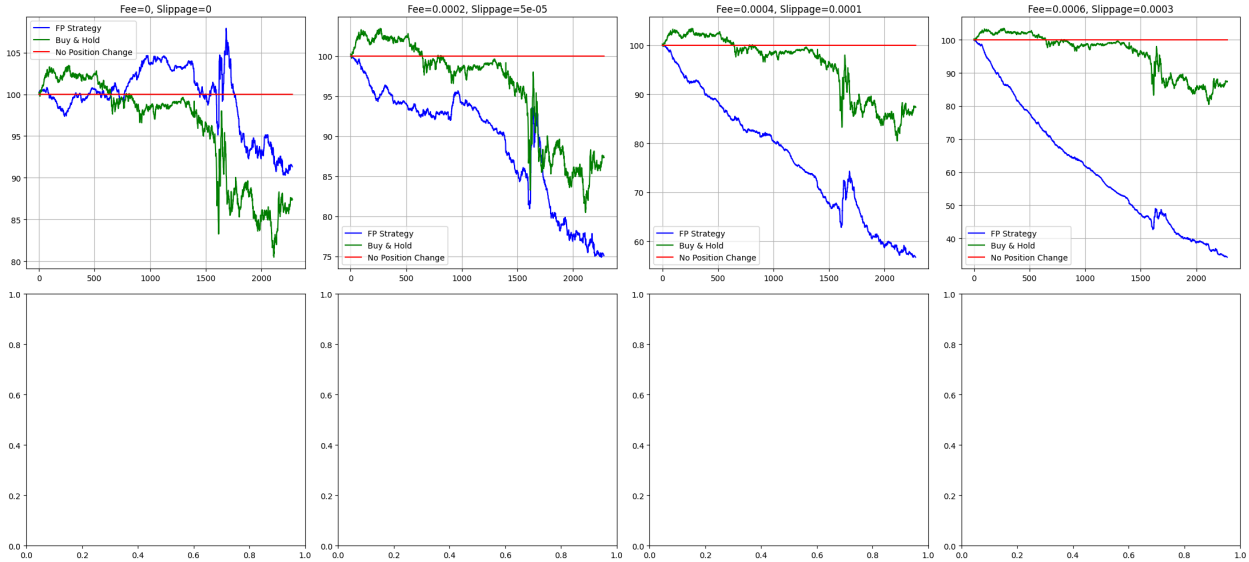
```
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi_t = features['obi'].iloc[t-1]
        dobi_t = features['dobi'].iloc[t-1]
        depth_t = features['depth'].iloc[t-1]
        slope_t = features['queue_slope'].iloc[t-1]
        spread_t = features['spread'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x
```

```
def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:7]
        sigma_params = params[7:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:7], res[0][7:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+500) for i in range(0, len(df_train)-500, 500)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
```

```
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
```

```
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-5-3781500835.py:21: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            91.32          -8.68           87.38
-12.62    100.0             0.0
0.0002   0.00005            75.08         -24.92           87.38
-12.62    100.0             0.0
0.0004   0.00010            56.72         -43.28           87.38
-12.62    100.0             0.0
0.0006   0.00030            34.31         -65.69           87.38
-12.62    100.0             0.0
```

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt
```

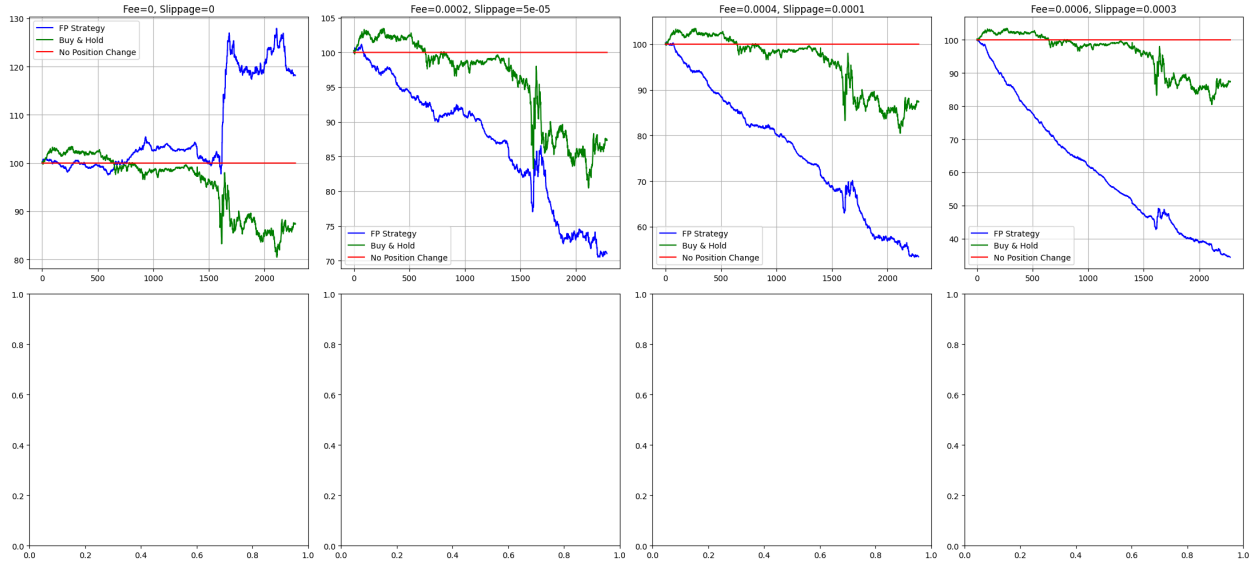
```
np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_1min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['net_queue_slope'] = (df['bids_notional_0'] - df['bids_notional_5']) - (df[
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = np.abs(df['dobi'])

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi_t = features['obi'].iloc[t-1]
        dobi_t = features['dobi'].iloc[t-1]
        depth_t = features['depth'].iloc[t-1]
```

```
        slope_t = features['net_queue_slope'].iloc[t-1]
        spread_t = features['spread'].iloc[t-1]
        dv_t = features['depth_variance'].iloc[t-1]
        abs_dobi_t = features['abs_dobi'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5
        sigma = np.abs(b0 + b1 * (dv_t + abs_dobi_t))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'net_queue_slope', 'spread',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:7]
        sigma_params = params[7:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:7], res[0][7:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+500) for i in range(0, len(df_train)-500, 500)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
```

```
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'net_queue_slope', 'sprea
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))
```

```
    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-6-703127033.py:21: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```



```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           118.16          18.16           87.38
-12.62    100.0             0.0
0.0002   0.00005            71.03         -28.97           87.38
-12.62    100.0             0.0
0.0004   0.00010            53.41         -46.59           87.38
-12.62    100.0             0.0
0.0006   0.00030            34.39         -65.61           87.38
-12.62    100.0             0.0
```

In [ ]:

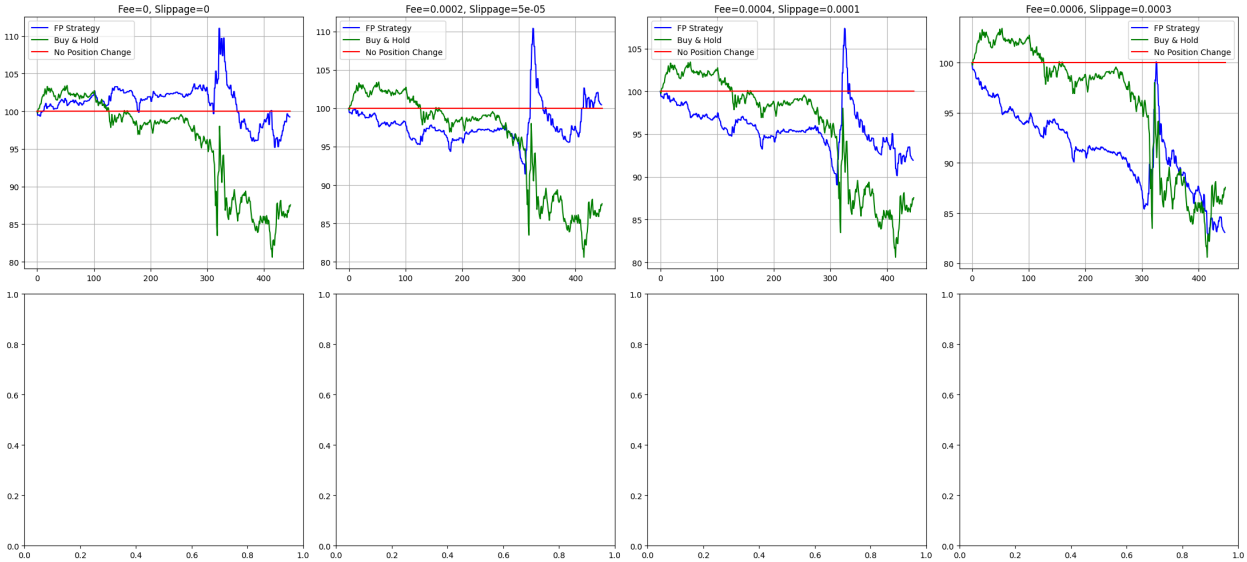
```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]

df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope_bid'] = df['bids_notional_0'] - df['bids_notional_5']
df['queue_slope_ask'] = df['asks_notional_0'] - df['asks_notional_5']
df['net_queue_slope'] = df['queue_slope_bid'] - df['queue_slope_ask']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = df['dobi'].abs()

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades
```

```
def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6, a7, a8, a9 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi = features['obi'].iloc[t-1]
        dobi = features['dobi'].iloc[t-1]
        depth = features['depth'].iloc[t-1]
        net_slope = features['net_queue_slope'].iloc[t-1]
        spread = features['spread'].iloc[t-1]
        depth_var = features['depth_variance'].iloc[t-1]
        abs_dobi = features['abs_dobi'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'net_queue_slope', 'spread',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:10]
        sigma_params = params[10:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*10 + [0.005, 0.005, 0.005], sigma0=0.2, options=
    return res[0][:10], res[0][10:]
```

```
fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'net_queue_slope', 'sprea
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]
```

```
    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-8-3118677534.py:24: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```



Final Portfolio Values and Returns for Different Fee/Slippage Configurations:

Fee	Slippage	FP Strategy ($)	FP Return (%)	Buy & Hold ($)	Buy & Hold Return (%)	NPC ($)	NPC Return (%)
0.0000	0.00000	99.25	-0.75	87.56	-12.44	100.0	0.0
0.0002	0.00005	100.46	0.46	87.56	-12.44	100.0	0.0
0.0004	0.00010	91.94	-8.06	87.56	-12.44	100.0	0.0
0.0006	0.00030	83.07	-16.93	87.56	-12.44	100.0	0.0

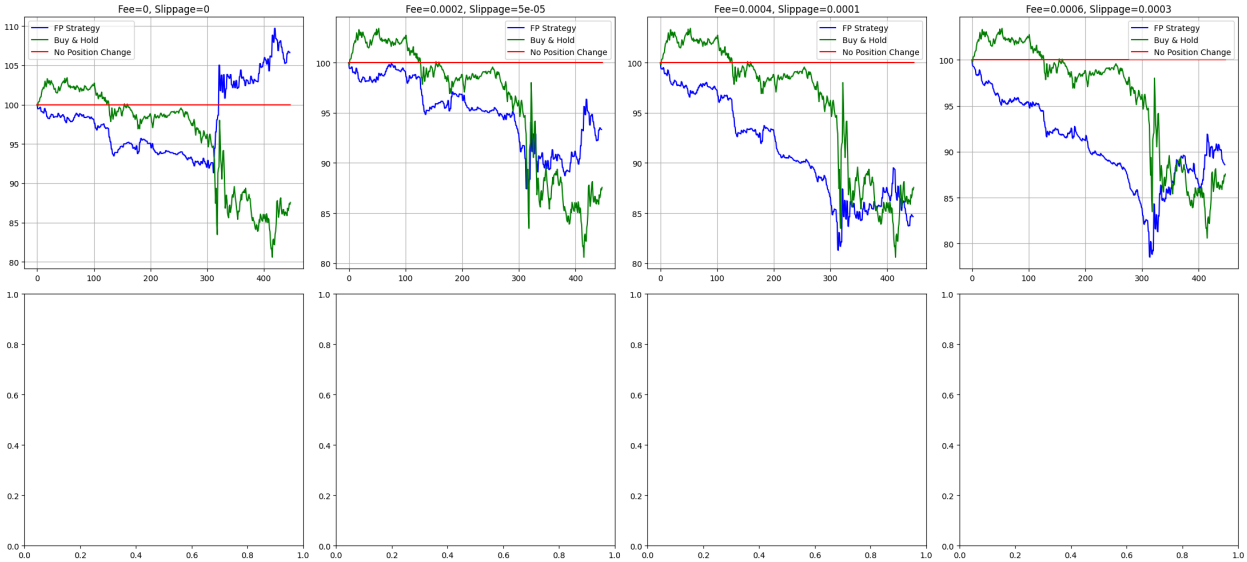
In [ ]:

```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ADA_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['abs_dobi'] = df['dobi'].abs()
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope_bid'] = df['bids_notional_0'] - df['bids_notional_5']
df['queue_slope_ask'] = df['asks_notional_5'] - df['asks_notional_0']
df['net_queue_slope'] = df['queue_slope_bid'] + df['queue_slope_ask']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].ffill().fillna(0)
```

```
train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        row = features.iloc[t-1]
        mu = sum([mu_params[i] * row[f] for i, f in enumerate(features.columns
        sigma = np.abs(sum([sigma_params[i] * row[f] for i, f in enumerate(fea
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'queue_slope_bid', 'queue_slo
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:10]
```

```
        sigma_params = params[10:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*10 + [0.005]*3, sigma0=0.2, options={'seed':rand
    return res[0][:10], res[0][10:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        fset = df_train.iloc[start:end][['obi', 'dobi', 'depth', 'queue_slope_
        signal = simulate_fp(mu_p, sigma_p, 0.0, fset, end-start, 1.0)
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            fset = df_cv.iloc[start:end][['obi', 'dobi', 'depth', 'queue_slope
            signal = simulate_fp(mu_p, sigma_p, 0.0, fset, window_size, 1.0)
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'queue_slope_bid', 'queue
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
```

```
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           106.63           6.63           87.56
-12.44    100.0             0.0
0.0002   0.00005            93.34          -6.66           87.56
-12.44    100.0             0.0
0.0004   0.00010            84.64         -15.36           87.56
-12.44    100.0             0.0
0.0006   0.00030            88.58         -11.42           87.56
-12.44    100.0             0.0
```

In [ ]:

In [ ]:
```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

# Load and preprocess data
df = pd.read_csv("ADA_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]

df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope_bid'] = df['bids_notional_0'] - df['bids_notional_5']
df['queue_slope_ask'] = df['asks_notional_0'] - df['asks_notional_5']
df['net_queue_slope'] = df['queue_slope_bid'] - df['queue_slope_ask']
```

```
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = df['dobi'].abs()

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.tanh(signal / threshold)
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.abs(trades[1:len(positions)]) * (fee + slip)
    costs[~trade_mask] = 0
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6, a7, a8, a9 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi = features['obi'].iloc[t-1]
        dobi = features['dobi'].iloc[t-1]
        depth = features['depth'].iloc[t-1]
        net_slope = features['net_queue_slope'].iloc[t-1]
        spread = features['spread'].iloc[t-1]
        depth_var = features['depth_variance'].iloc[t-1]
        abs_dobi = features['abs_dobi'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
```

```
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'net_queue_slope', 'spread',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:10]
        sigma_params = params[10:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*10 + [0.005, 0.005, 0.005], sigma0=0.2, options=
    return res[0][:10], res[0][10:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)
```

```
    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'net_queue_slope', 'sprea
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = np.abs(first_position) * (fee + slip) if first_positi
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
```

```
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-17-1945131386.py:23: FutureWarning: Series.fillna with 'meth
od' is deprecated and will raise in a future version. Use obj.ffill() or obj.bf
ill() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```