In [ ]:

```
pip install cma
```

```
Collecting cma
  Downloading cma-4.2.0-py3-none-any.whl.metadata (7.7 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from cma) (2.0.2)
Downloading cma-4.2.0-py3-none-any.whl (288 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 288.2/288.2 kB 4.4 MB/s eta 0:00:00
Installing collected packages: cma
Successfully installed cma-4.2.0
```

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("ETH_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
```

```python
    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], window
            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))
```

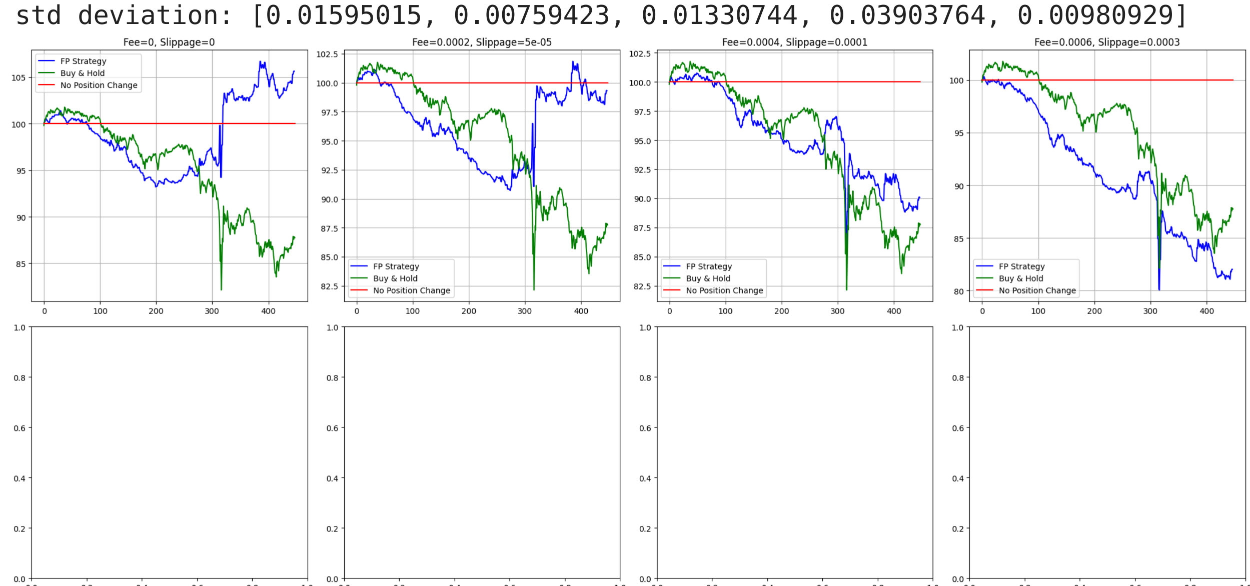
```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:13
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.606676083059089e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -3.160671368341905e-02 1.3e+00 1.61e-01  1e-01  2e-01 0:00.0
    3     24 -1.969515552675905e-02 1.3e+00 1.55e-01  1e-01  2e-01 0:00.0
   80    640 -1.105012840781914e-01 2.5e+01 4.12e-03  3e-04  3e-03 0:00.7
termination on tolflatfitness=1 (Tue Jul 22 12:53:14 2025)
final/bestever f-value = -1.105013e-01 -1.116381e-01 after 641/506 evaluations
incumbent solution: [ 0.09753337, -0.36569446, -0.63472271, -0.25374184, -0.066
15709]
std deviation: [0.00030539, 0.00105977, 0.00141874, 0.00132591, 0.00312751]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:14
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.283712782242841e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -1.283712782242841e-02 1.1e+00 1.72e-01  2e-01  2e-01 0:00.0
    3     24 -1.349697412644613e-02 1.2e+00 1.75e-01  2e-01  2e-01 0:00.0
    8     64 -1.283712782242841e-02 2.2e+00 1.52e-01  1e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:53:14 2025)
final/bestever f-value = -1.283713e-02 -1.875734e-02 after 65/25 evaluations
incumbent solution: [0.29959549, 0.05202989, 0.09048512, 0.04845959, 0.1305949
3]
std deviation: [0.13949195, 0.16365975, 0.13438779, 0.15509893, 0.14063303]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:14
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.550943731129273e-02 1.0e+00 2.04e-01  2e-01  2e-01 0:00.0
    2     16 -3.435428784481598e-02 1.4e+00 2.09e-01  2e-01  2e-01 0:00.0
    3     24 -4.615272472322296e-02 1.5e+00 2.47e-01  2e-01  3e-01 0:00.0
   93    744 -7.171024163342832e-02 1.6e+01 2.09e-02  2e-03  2e-02 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:53:15 2025)
final/bestever f-value = -7.171024e-02 -7.171024e-02 after 745/642 evaluations
incumbent solution: [-0.14418196, -0.76592522, -1.45345156, -0.49081354, 0.0892
0684]
std deviation: [0.0020892, 0.0038805, 0.01980952, 0.00606777, 0.00835662]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.544915980934494e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -4.544915980934494e-02 1.2e+00 1.60e-01  1e-01  2e-01 0:00.0
    3     24 -4.544915980934494e-02 1.2e+00 1.57e-01  1e-01  2e-01 0:00.0
   35    280 -4.686458290720896e-02 5.8e+00 8.05e-02  2e-02  1e-01 0:00.3
termination on tolflatfitness=1 (Tue Jul 22 12:53:15 2025)
final/bestever f-value = -4.686458e-02 -5.505383e-02 after 281/136 evaluations
incumbent solution: [ 0.39978579, 0.31442136, 1.5739091, -1.32947814, 0.5911784
8]
std deviation: [0.02617449, 0.07565209, 0.0960854, 0.07360357, 0.02317744]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.621556452625875e-02 1.0e+00 2.16e-01  2e-01  2e-01 0:00.0
    2     16 -7.000895168403432e-02 1.3e+00 2.79e-01  3e-01  3e-01 0:00.0
    3     24 -5.460604609137931e-02 1.5e+00 2.82e-01  2e-01  3e-01 0:00.0
```

```
   61    488 -7.804274405255818e-02 1.3e+01 1.01e-01  2e-02  1e-01 0:00.5
termination on tolflatfitness=1 (Tue Jul 22 12:53:16 2025)
final/bestever f-value = -7.804274e-02 -7.918006e-02 after 489/240 evaluations
incumbent solution: [-2.790271, -0.32384324, 1.8759848, -4.76401075, -1.2149471
3]
std deviation: [0.08003318, 0.02122773, 0.06975771, 0.11300429, 0.03663213]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:16
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.581676083059089e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -1.581676083059089e-02 1.3e+00 1.81e-01  2e-01  2e-01 0:00.0
    3     24 -1.581676083059089e-02 1.5e+00 1.81e-01  2e-01  2e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:53:16 2025)
final/bestever f-value = -1.581676e-02 -1.581676e-02 after 25/5 evaluations
incumbent solution: [ 0.31319718, -0.01987204, -0.13968355, -0.33869647, -0.075
00157]
std deviation: [0.1739219, 0.15347958, 0.17732414, 0.19236438, 0.17359208]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:16
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.258712782242841e-02 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -1.258712782242841e-02 1.3e+00 1.92e-01  2e-01  2e-01 0:00.0
    3     24 -1.258712782242841e-02 1.5e+00 2.06e-01  2e-01  2e-01 0:00.0
    5     40 -1.258712782242841e-02 2.0e+00 2.06e-01  2e-01  2e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:53:16 2025)
final/bestever f-value = -1.258713e-02 -1.258713e-02 after 41/1 evaluations
incumbent solution: [ 0.28933438, 0.10041964, 0.50890086, -0.00232154, -0.16434
962]
std deviation: [0.21505225, 0.1663962, 0.23424511, 0.19810563, 0.17864623]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:16
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.140327779131345e-02 1.0e+00 2.00e-01  2e-01  2e-01 0:00.0
    2     16 -1.172409127960264e-02 1.4e+00 1.93e-01  2e-01  2e-01 0:00.0
    3     24 -2.300426631209412e-02 1.3e+00 1.85e-01  2e-01  2e-01 0:00.0
  100    800 -7.657374786392873e-02 5.3e+01 8.35e-03  3e-04  1e-02 0:01.2
  107    856 -7.657374786392873e-02 6.6e+01 5.41e-03  2e-04  8e-03 0:01.3
termination on tolflatfitness=1 (Tue Jul 22 12:53:18 2025)
final/bestever f-value = -7.657375e-02 -7.661715e-02 after 857/480 evaluations
incumbent solution: [ 0.01676381, -0.04368776, -1.46250458, 0.26875438, -0.2444
3963]
std deviation: [0.0001862, 0.0004025, 0.00767504, 0.00197912, 0.00078243]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:18
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.519915980934494e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -4.519915980934494e-02 1.2e+00 1.60e-01  1e-01  2e-01 0:00.0
    3     24 -4.519915980934494e-02 1.2e+00 1.57e-01  1e-01  2e-01 0:00.0
   14    112 -4.519915980934494e-02 2.3e+00 3.06e-01  2e-01  3e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:53:18 2025)
final/bestever f-value = -4.519916e-02 -5.840246e-02 after 113/66 evaluations
incumbent solution: [ 0.80073338, -0.03199459, 0.2710671, -0.15362778, -0.09335
138]
std deviation: [0.23684875, 0.32098294, 0.30464524, 0.2834977, 0.26130622]
```

```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:18
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.871556452625874e-02 1.0e+00 2.16e-01  2e-01  2e-01 0:00.0
    2     16 -5.181245339099905e-02 1.3e+00 2.38e-01  2e-01  3e-01 0:00.0
    3     24 -4.385004396597231e-02 1.5e+00 2.42e-01  2e-01  3e-01 0:00.1
   75    600 -6.027954776273134e-02 6.0e+00 5.56e-02  8e-03  4e-02 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:53:19 2025)
final/bestever f-value = -6.027955e-02 -6.027955e-02 after 601/202 evaluations
incumbent solution: [-0.60220022, -0.32122395, -0.09714682, -0.86467986, -1.417
54776]
std deviation: [0.03130126, 0.0082413, 0.0356498, 0.03048169, 0.01276729]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:19
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.556676083059089e-02 1.0e+00 1.85e-01  2e-01  2e-01 0:00.0
    2     16 -2.422327434335109e-02 1.4e+00 1.92e-01  2e-01  2e-01 0:00.0
    3     24 -5.470793898895576e-02 1.6e+00 1.67e-01  1e-01  2e-01 0:00.0
   53    424 -9.821607769303697e-02 1.8e+01 7.54e-03  1e-03  7e-03 0:00.5
termination on tolflatfitness=1 (Tue Jul 22 12:53:20 2025)
final/bestever f-value = -9.821608e-02 -9.821608e-02 after 425/263 evaluations
incumbent solution: [ 0.10208978, -0.20482088, -0.69554513, -0.27007868, -0.151
48691]
std deviation: [0.00138279, 0.00288839, 0.00686096, 0.00498708, 0.00479373]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:20
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.233712782242841e-02 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -1.233712782242841e-02 1.3e+00 1.92e-01  2e-01  2e-01 0:00.0
    3     24 -1.233712782242841e-02 1.5e+00 2.06e-01  2e-01  2e-01 0:00.0
    5     40 -1.233712782242841e-02 2.0e+00 2.06e-01  2e-01  2e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:53:20 2025)
final/bestever f-value = -1.233713e-02 -1.233713e-02 after 41/1 evaluations
incumbent solution: [ 0.28933438, 0.10041964, 0.50890086, -0.00232154, -0.16434
962]
std deviation: [0.21505225, 0.1663962, 0.23424511, 0.19810563, 0.17864623]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:20
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.040327779131345e-02 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     16 -2.129250934769578e-02 1.4e+00 1.61e-01  1e-01  2e-01 0:00.0
    3     24 -1.154293122991168e-02 1.4e+00 1.54e-01  1e-01  2e-01 0:00.0
   92    736 -8.291815150165227e-02 3.8e+01 7.53e-03  7e-04  6e-03 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:53:21 2025)
final/bestever f-value = -8.291815e-02 -8.374577e-02 after 737/484 evaluations
incumbent solution: [ 0.00821358, -0.04922176, -1.63841035, 0.03218449, -0.0327
4894]
std deviation: [0.0006652, 0.00085322, 0.00648899, 0.00423753, 0.00378925]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:21
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.494915980934494e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -4.494915980934494e-02 1.2e+00 1.60e-01  1e-01  2e-01 0:00.0
    3     24 -4.494915980934494e-02 1.2e+00 1.57e-01  1e-01  2e-01 0:00.0
```

```
    9     72 -4.494915980934494e-02 2.0e+00 1.37e-01  1e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:53:21 2025)
final/bestever f-value = -4.494916e-02 -4.625986e-02 after 73/29 evaluations
incumbent solution: [ 0.29575742, -0.32860441, -0.12848303, -0.03716374, 0.1150
256, ]
std deviation: [0.10009795, 0.15420034, 0.13171876, 0.11035383, 0.11838169]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:21
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.121556452625874e-02 1.0e+00 2.16e-01  2e-01  2e-01 0:00.0
    2     16 -4.481245339099904e-02 1.3e+00 2.42e-01  2e-01  3e-01 0:00.0
    3     24 -3.877802199828263e-02 1.5e+00 2.38e-01  2e-01  3e-01 0:00.0
   53    424 -5.495873747618587e-02 1.0e+01 9.04e-03  3e-03  6e-03 0:00.4
termination on tolflatfitness=1 (Tue Jul 22 12:53:21 2025)
final/bestever f-value = -5.495874e-02 -5.495874e-02 after 425/333 evaluations
incumbent solution: [-0.3469809, -0.40648175, -0.26969078, -0.85867479, -0.5373
5128]
std deviation: [0.00322126, 0.0029739, 0.0049485, 0.00589114, 0.00460218]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:21
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.516676083059089e-02 1.0e+00 1.85e-01  2e-01  2e-01 0:00.0
    2     16 -2.400373219591850e-02 1.3e+00 1.92e-01  2e-01  2e-01 0:00.0
    3     24 -4.544199525507479e-02 1.6e+00 1.72e-01  2e-01  2e-01 0:00.0
   65    520 -7.834809752179546e-02 2.6e+01 2.42e-02  7e-03  3e-02 0:00.5
termination on tolflatfitness=1 (Tue Jul 22 12:53:22 2025)
final/bestever f-value = -7.834810e-02 -7.952527e-02 after 521/441 evaluations
incumbent solution: [ 0.21357452, -0.317897, -1.14594445, -0.54155891, 0.252988
07]
std deviation: [0.00938086, 0.00745084, 0.02681571, 0.0130788, 0.01084494]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:22
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.193712782242841e-02 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -1.193712782242841e-02 1.3e+00 1.52e-01  1e-01  2e-01 0:00.0
    3     24 -1.193712782242841e-02 1.4e+00 1.41e-01  1e-01  1e-01 0:00.0
    5     40 -1.193712782242841e-02 1.5e+00 1.37e-01  1e-01  1e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:53:22 2025)
termination on tolflatfitness=1 (Tue Jul 22 12:53:22 2025)
final/bestever f-value = -1.193713e-02 -1.193713e-02 after 41/1 evaluations
incumbent solution: [0.16972447, 0.04994999, 0.10155209, 0.05705084, 0.2613291
1]
std deviation: [0.12964896, 0.11927461, 0.13357529, 0.11649359, 0.13126631]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:22
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.803277791313451e-03 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     16 -1.623700185973941e-02 1.3e+00 1.73e-01  1e-01  2e-01 0:00.0
    3     24 -1.891736625704424e-02 1.5e+00 1.74e-01  1e-01  2e-01 0:00.0
   66    528 -3.609292095870512e-02 4.1e+01 1.71e-02  7e-03  2e-02 0:00.6
termination on tolflatfitness=1 (Tue Jul 22 12:53:23 2025)
final/bestever f-value = -3.609292e-02 -5.012505e-02 after 529/317 evaluations
incumbent solution: [-0.03033599, 0.02916577, -0.2622613, 0.00862751, -0.055604
7, ]
```

```
std deviation: [0.00656438, 0.00814481, 0.01853476, 0.01337391, 0.01392918]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:23
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.454915980934494e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.0
    2     16 -4.454915980934494e-02 1.2e+00 1.60e-01  1e-01  2e-01 0:00.0
    3     24 -4.454915980934494e-02 1.2e+00 1.57e-01  1e-01  2e-01 0:00.0
    9     72 -4.454915980934494e-02 2.0e+00 1.37e-01  1e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:53:23 2025)
final/bestever f-value = -4.454916e-02 -4.465986e-02 after 73/29 evaluations
incumbent solution: [ 0.29575742, -0.32860441, -0.12848303, -0.03716374, 0.1150
256, ]
std deviation: [0.10009795, 0.15420034, 0.13171876, 0.11035383, 0.11838169]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:53:23
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.921556452625875e-02 1.0e+00 2.16e-01  2e-01  2e-01 0:00.0
    2     16 -3.361245339099905e-02 1.3e+00 2.42e-01  2e-01  3e-01 0:00.0
    3     24 -2.247540146877340e-02 1.6e+00 2.56e-01  2e-01  3e-01 0:00.0
   53    424 -3.079251644222380e-02 1.1e+01 2.96e-02  8e-03  4e-02 0:00.4
termination on tolflatfitness=1 (Tue Jul 22 12:53:23 2025)
final/bestever f-value = -3.079252e-02 -3.361245e-02 after 425/12 evaluations
incumbent solution: [-0.80720427, -0.28150654, -0.12448537, -1.57268585, -0.348
13291]
std deviation: [0.01595015, 0.00759423, 0.01330744, 0.03903764, 0.00980929]
```




```
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           105.63           5.63           87.74
-12.26    100.0             0.0
0.0002   0.00005            99.31          -0.69           87.74
-12.26    100.0             0.0
0.0004   0.00010            90.08          -9.92           87.74
-12.26    100.0             0.0
0.0006   0.00030            82.03         -17.97           87.74
-12.26    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("ETH_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], window
            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')
```

```python
    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 12:57:42
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.894983897072436e-02 1.0e+00 2.00e-01  2e-01  2e-01 0:00.1
    2     16 -3.552290459335072e-02 1.3e+00 2.03e-01  2e-01  2e-01 0:00.1
    3     24 -5.761652171783460e-02 1.5e+00 2.12e-01  2e-01  2e-01 0:00.2
   81    648 -8.908035578437623e-02 2.7e+01 7.25e-03  1e-03  6e-03 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:44 2025)
final/bestever f-value = -8.908036e-02 -8.908036e-02 after 649/500 evaluations
incumbent solution: [ 0.53969945, -0.76235711, 0.10500557, -1.05157196, 0.00399
708]
std deviation: [0.00291881, 0.00111565, 0.00178586, 0.00594, 0.00289382]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:44
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.605747241975177e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -1.050364613459021e-02 1.2e+00 1.64e-01  1e-01  2e-01 0:00.0
    3     24 -3.090171764645255e-02 1.3e+00 1.61e-01  1e-01  2e-01 0:00.1
   62    496 -6.307305202493119e-02 2.5e+01 6.91e-03  5e-04  9e-03 0:01.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:45 2025)
final/bestever f-value = -6.307305e-02 -6.307305e-02 after 497/398 evaluations
incumbent solution: [ 0.11523076, -0.44369617, -0.18451746, -1.86396143, 0.2093
7478]
std deviation: [0.00086631, 0.0005004, 0.00291932, 0.00852058, 0.0021692, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:45
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.722122584990167e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.776233600899513e-02 1.1e+00 1.87e-01  2e-01  2e-01 0:00.0
    3     24 -6.203424804451352e-03 1.3e+00 2.00e-01  2e-01  2e-01 0:00.1
   47    376 -3.307480467940582e-02 1.8e+01 3.25e-02  1e-02  3e-02 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:57:46 2025)
final/bestever f-value = -3.307480e-02 -3.307480e-02 after 377/195 evaluations
incumbent solution: [ 0.46129709, 1.71042723, -0.27283174, 0.63398587, -1.26784
188]
std deviation: [0.02216435, 0.03232654, 0.01227324, 0.02471831, 0.01734029]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:46
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.482619576473621e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -3.036109630086781e-02 1.2e+00 1.70e-01  2e-01  2e-01 0:00.0
    3     24 -1.975704312233795e-02 1.2e+00 1.66e-01  2e-01  2e-01 0:00.1
  100    800 -5.637955367805514e-02 5.0e+01 1.10e-02  7e-04  1e-02 0:01.8
  137   1096 -5.637955367805514e-02 7.7e+01 1.75e-02  5e-04  9e-03 0:02.4
termination on tolfunhist=1e-12 (Tue Jul 22 12:57:49 2025)
final/bestever f-value = -5.637955e-02 -5.637955e-02 after 1097/792 evaluations
incumbent solution: [ 0.18757114, 0.15555046, 0.24217544, 0.17210945, -0.756062
68]
std deviation: [0.00610421, 0.00051977, 0.00679522, 0.00898921, 0.0017521, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.481921606700087e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -1.351517617220299e-02 1.2e+00 2.16e-01  2e-01  2e-01 0:00.0
```

```
    3     24 -1.258175880323709e-02 1.4e+00 2.13e-01  2e-01  2e-01 0:00.1
   71    568 -4.668081691945858e-02 9.1e+00 6.32e-03  1e-03  3e-03 0:01.2
termination on tolflatfitness=1 (Tue Jul 22 12:57:50 2025)
final/bestever f-value = -4.668082e-02 -4.668082e-02 after 569/468 evaluations
incumbent solution: [ 0.54288854, -1.06600096, -0.07984083, -0.38624687, -0.257
78827]
std deviation: [0.00302952, 0.0012396, 0.00178658, 0.00193837, 0.00164246]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:51
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.334006204771777e-02 1.0e+00 2.02e-01  2e-01  2e-01 0:00.0
    2     16 -2.672648279255035e-03 1.4e+00 1.83e-01  2e-01  2e-01 0:00.0
    3     24 -1.581577344848151e-02 1.3e+00 1.84e-01  1e-01  2e-01 0:00.1
  100    800 -8.628436751938043e-02 8.7e+00 5.30e-03  7e-04  3e-03 0:02.0
  103    824 -8.628436751938043e-02 1.0e+01 3.79e-03  5e-04  2e-03 0:02.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:54 2025)
final/bestever f-value = -8.628437e-02 -9.093184e-02 after 825/420 evaluations
incumbent solution: [ 0.04095431, 0.09149711, -0.25110293, -0.12768326, -0.9124
2088]
std deviation: [0.00050988, 0.00069157, 0.00203696, 0.00102237, 0.00056037]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:54
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.025364613459021e-02 1.0e+00 1.84e-01  2e-01  2e-01 0:00.0
    2     16 -1.025364613459021e-02 1.3e+00 1.91e-01  2e-01  2e-01 0:00.1
    3     24 -2.104413144969558e-02 1.4e+00 1.92e-01  2e-01  2e-01 0:00.1
   98    784 -5.135197628168339e-02 5.2e+01 6.48e-03  3e-04  6e-03 0:02.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:56 2025)
final/bestever f-value = -5.135198e-02 -5.327471e-02 after 785/446 evaluations
incumbent solution: [ 0.5215914, 0.07014214, 0.25687136, -0.34140009, -0.660417
09]
std deviation: [0.00447557, 0.00032636, 0.00623144, 0.00199518, 0.00070415]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:56
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.472122584990167e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.037722404938277e-02 1.3e+00 1.61e-01  1e-01  2e-01 0:00.0
    3     24 -1.569529680870982e-02 1.3e+00 1.63e-01  1e-01  2e-01 0:00.0
   61    488 -2.490392957865759e-02 1.8e+01 9.74e-03  3e-03  5e-03 0:01.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:57 2025)
final/bestever f-value = -2.490393e-02 -2.490393e-02 after 489/284 evaluations
incumbent solution: [ 0.4471484, -0.37824907, 0.36550431, -0.44874341, -0.08148
173]
std deviation: [0.00435651, 0.00299344, 0.00489482, 0.00507351, 0.00531531]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:57
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.439422842317514e-02 1.0e+00 1.82e-01  2e-01  2e-01 0:00.0
    2     16 -1.439422842317514e-02 1.2e+00 2.08e-01  2e-01  2e-01 0:00.0
    3     24 -1.439422842317514e-02 1.6e+00 2.29e-01  2e-01  3e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:57:57 2025)
final/bestever f-value = -1.439423e-02 -1.439423e-02 after 25/5 evaluations
incumbent solution: [ 0.60312204, 0.07980727, -0.12373699, -0.28328214, -0.1440
1976]
```

```
std deviation: [0.27308386, 0.21287883, 0.21505166, 0.25654304, 0.20300922]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:57
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.256921606700087e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.086480241680507e-02 1.2e+00 1.90e-01  2e-01  2e-01 0:00.0
    3     24 -1.086480241680507e-02 1.5e+00 2.02e-01  2e-01  2e-01 0:00.1
   47    376 -1.624422815621757e-02 1.1e+01 8.12e-02  2e-02  8e-02 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:57:58 2025)
final/bestever f-value = -1.624423e-02 -1.624423e-02 after 377/54 evaluations
incumbent solution: [0.3714085, 0.92631717, 0.77970515, 0.66124393, 0.60235205]
std deviation: [0.05126576, 0.04553291, 0.06739031, 0.08300655, 0.01801135]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:59
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.340062047717779e-03 1.0e+00 2.07e-01  2e-01  2e-01 0:00.0
    2     16 -2.224885825807979e-02 1.3e+00 2.78e-01  2e-01  3e-01 0:00.0
    3     24 -1.213431018530056e-02 1.5e+00 2.97e-01  3e-01  3e-01 0:00.1
```

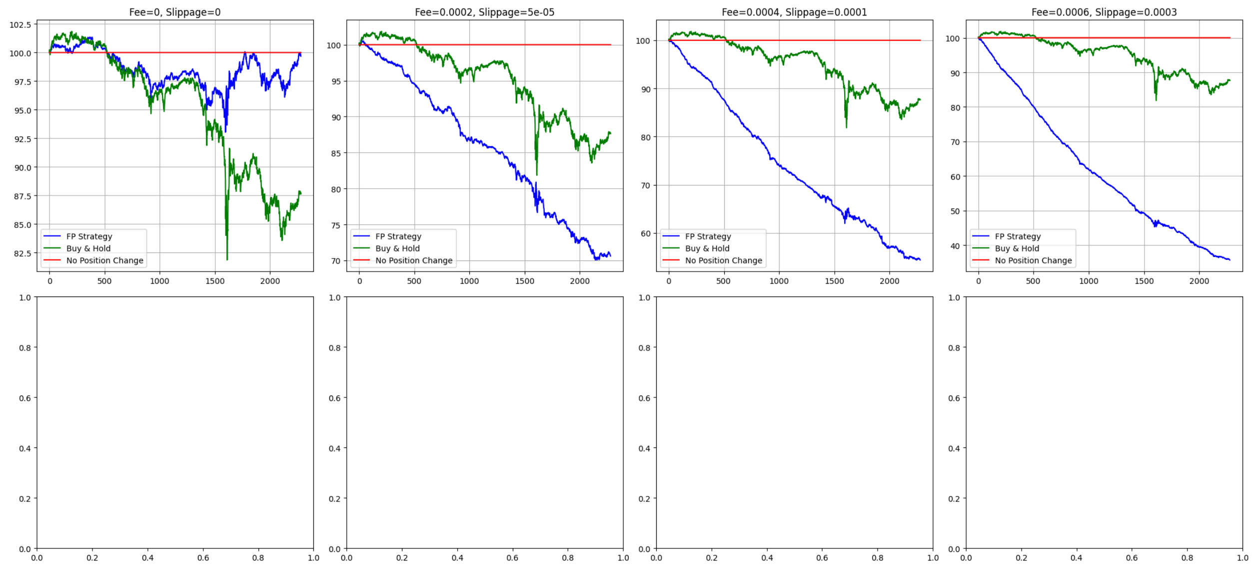
```
/tmp/ipython-input-7-1564720100.py:55: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-1564720100.py:55: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-1564720100.py:53: RuntimeWarning: overflow encountered in
scalar multiply
  mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
/tmp/ipython-input-7-1564720100.py:54: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
```

```
   46    368 -4.214896014232656e-02 1.9e+01 1.81e-01  1e-01  2e-01 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:58:00 2025)
final/bestever f-value = -4.214896e-02 -4.604630e-02 after 369/92 evaluations
incumbent solution: [-2.18457177, 3.59730805, 5.36567384, -2.78438605, -3.80496
125]
std deviation: [0.14083593, 0.1567434, 0.22762503, 0.15296439, 0.13858823]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:00
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.000364613459021e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     16 -1.000364613459021e-02 1.3e+00 1.82e-01  2e-01  2e-01 0:00.0
    3     24 -1.000364613459021e-02 1.5e+00 1.88e-01  2e-01  2e-01 0:00.1
```

```
/tmp/ipython-input-7-1564720100.py:55: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
```

```
termination on tolfun=1e-11 (Tue Jul 22 12:58:00 2025)
termination on tolflatfitness=1 (Tue Jul 22 12:58:00 2025)
final/bestever f-value = -1.000365e-02 -1.000365e-02 after 25/5 evaluations
incumbent solution: [ 0.47046759, -0.1125464, 0.03856429, -0.21772953, -0.04419
997]
std deviation: [0.20211133, 0.18070851, 0.16534958, 0.19023226, 0.17123874]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:00
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.222122584990167e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -9.222122584990167e-03 1.3e+00 1.76e-01  2e-01  2e-01 0:00.0
    3     24 -9.345346386404775e-03 1.4e+00 1.96e-01  2e-01  2e-01 0:00.1
   32    256 -1.298142413695930e-02 5.6e+00 6.29e-02  3e-02  7e-02 0:00.5
termination on tolflatfitness=1 (Tue Jul 22 12:58:01 2025)
final/bestever f-value = -1.298142e-02 -1.597591e-02 after 257/30 evaluations
incumbent solution: [ 1.15216015, -0.34446713, -0.04745465, -0.40921011, -0.220
11521]
std deviation: [0.07463395, 0.03261134, 0.03455281, 0.04074981, 0.03019944]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:01
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.414422842317514e-02 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     16 -1.414422842317514e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.0
    3     24 -2.886109630086781e-02 1.5e+00 1.74e-01  1e-01  2e-01 0:00.1
   32    256 -2.886109630086781e-02 4.0e+00 6.77e-02  3e-02  6e-02 0:00.6
termination on tolflatfitness=1 (Tue Jul 22 12:58:02 2025)
final/bestever f-value = -2.886110e-02 -2.886110e-02 after 257/24 evaluations
incumbent solution: [ 0.51343848, 0.53428756, -0.20669012, -0.25084156, -0.5365
9826]
std deviation: [0.05558083, 0.05020521, 0.03945684, 0.05964965, 0.0292034, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:02
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.061480241680507e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -1.061480241680507e-02 1.3e+00 1.82e-01  2e-01  2e-01 0:00.0
    3     24 -1.061480241680507e-02 1.5e+00 1.72e-01  1e-01  2e-01 0:00.0
    6     48 -1.061480241680507e-02 1.9e+00 2.03e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:58:02 2025)
final/bestever f-value = -1.061480e-02 -1.061480e-02 after 49/5 evaluations
incumbent solution: [ 0.46180302, 0.33257986, 0.34519206, -0.43568774, -0.02518
47, ]
std deviation: [0.20004243, 0.19823176, 0.1991084, 0.19899656, 0.21259361]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:03
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.022648279255038e-03 1.0e+00 1.86e-01  2e-01  2e-01 0:00.0
    2     16 -2.022648279255038e-03 1.3e+00 1.96e-01  2e-01  2e-01 0:00.0
    3     24 -2.022648279255038e-03 1.5e+00 2.00e-01  2e-01  2e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:58:03 2025)
final/bestever f-value = -2.022648e-03 -2.022648e-03 after 25/5 evaluations
incumbent solution: [ 0.4209689, -0.01051846, -0.00919083, -0.36216649, 0.12199
728]
std deviation: [0.21344753, 0.18145524, 0.19112304, 0.21487733, 0.17890455]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:03
```

```
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.603646134590206e-03 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     16 -9.603646134590206e-03 1.3e+00 1.86e-01  2e-01  2e-01 0:00.0
    3     24 -9.603646134590206e-03 1.5e+00 1.96e-01  2e-01  2e-01 0:00.1
    6     48 -9.603646134590206e-03 2.1e+00 1.88e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:58:03 2025)
final/bestever f-value = -9.603646e-03 -1.195364e-02 after 49/25 evaluations
incumbent solution: [ 0.43172413, 0.03100178, -0.01496067, -0.4421665, -0.13785
281]
std deviation: [0.19285654, 0.17784977, 0.20715856, 0.18984424, 0.16751761]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:03
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.822122584990167e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -8.822122584990167e-03 1.3e+00 1.73e-01  2e-01  2e-01 0:00.0
    3     24 -8.822122584990167e-03 1.4e+00 1.80e-01  2e-01  2e-01 0:00.1
   11     88 -8.822122584990167e-03 2.3e+00 1.96e-01  1e-01  2e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:58:03 2025)
final/bestever f-value = -8.822123e-03 -1.132233e-02 after 89/32 evaluations
incumbent solution: [ 0.95732462, -0.14337598, -0.245554, 0.03564569, -0.124300
7, ]
std deviation: [0.21774592, 0.16703091, 0.21077044, 0.14862007, 0.14656386]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:03
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.374422842317514e-02 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     16 -1.374422842317514e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.0
    3     24 -2.766109630086781e-02 1.5e+00 1.74e-01  1e-01  2e-01 0:00.1
   11     88 -1.374422842317514e-02 2.1e+00 1.81e-01  1e-01  2e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:58:04 2025)
final/bestever f-value = -1.374423e-02 -2.766110e-02 after 89/24 evaluations
incumbent solution: [ 0.25535998, 0.00875839, 0.07345428, 0.0965565, -0.1358907
7]
std deviation: [0.15774178, 0.14855452, 0.21447032, 0.14079072, 0.15699442]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:04
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.021480241680507e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -1.021480241680507e-02 1.3e+00 1.82e-01  2e-01  2e-01 0:00.0
    3     24 -1.021480241680507e-02 1.5e+00 1.72e-01  1e-01  2e-01 0:00.1
    6     48 -1.021480241680507e-02 1.9e+00 2.03e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:58:04 2025)
final/bestever f-value = -1.021480e-02 -1.021480e-02 after 49/5 evaluations
incumbent solution: [ 0.46180302, 0.33257986, 0.34519206, -0.43568774, -0.02518
47, ]
std deviation: [0.20004243, 0.19823176, 0.1991084, 0.19899656, 0.21259361]
```




```
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            99.68          -0.32           87.65
-12.35    100.0            0.0
0.0002   0.00005            70.61         -29.39           87.65
-12.35    100.0            0.0
0.0004   0.00010            54.40         -45.60           87.65
-12.35    100.0            0.0
0.0006   0.00030            35.73         -64.27           87.65
-12.35    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("ETH_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], window
            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_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_trac
    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 12:58:43
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.624499488922940e-02 1.0e+00 2.00e-01  2e-01  2e-01 0:00.2
```

```
/tmp/ipython-input-8-2881762486.py:55: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-8-2881762486.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 -4.625334404713755e-02 1.3e+00 2.04e-01  2e-01  2e-01 0:00.3
    3     24 -4.429501649167111e-02 1.3e+00 1.76e-01  2e-01  2e-01 0:00.5
   23    184 -8.654376352954074e-02 3.4e+00 2.68e-01  1e-01  3e-01 0:03.5
   50    400 -1.189078819267451e-01 4.7e+00 5.60e-02  2e-02  4e-02 0:07.6
   81    648 -1.225266473595878e-01 1.3e+01 1.34e-02  3e-03  9e-03 0:12.7
  100    800 -1.226711261113627e-01 1.9e+01 4.58e-03  6e-04  3e-03 0:15.7
  138   1104 -1.228841363734423e-01 4.5e+01 1.30e-03  9e-05  7e-04 0:22.7
  143   1144 -1.228841363734423e-01 5.0e+01 1.04e-03  7e-05  4e-04 0:23.5
termination on tolflatfitness=1 (Tue Jul 22 12:59:06 2025)
final/bestever f-value = -1.228841e-01 -1.232340e-01 after 1145/748 evaluations
incumbent solution: [ 0.6580059, -1.01056512, 1.37594153, -0.1481399, 0.9711413
5]
std deviation: [2.18292031e-04, 6.57689410e-05, 4.36123386e-04, 8.41496186e-05,
1.31282446e-04]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:07
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.433888313134357e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.1
    2     16 -1.457901665486272e-02 1.2e+00 1.81e-01  2e-01  2e-01 0:00.3
    3     24 -2.034283006708559e-02 1.4e+00 1.73e-01  1e-01  2e-01 0:00.5
   24    192 -4.270745523714226e-02 5.7e+00 2.26e-01  8e-02  3e-01 0:03.6
   50    400 -5.473273775063348e-02 6.6e+00 7.54e-02  2e-02  8e-02 0:07.8
   82    656 -5.790920868806904e-02 1.2e+01 1.06e-02  1e-03  8e-03 0:12.9
  100    800 -5.815193754592496e-02 2.1e+01 4.15e-03  5e-04  3e-03 0:15.5
  136   1088 -5.835489878421818e-02 3.7e+01 8.97e-04  7e-05  4e-04 0:21.4
termination on tolflatfitness=1 (Tue Jul 22 12:59:28 2025)
final/bestever f-value = -5.835490e-02 -5.836983e-02 after 1089/956 evaluations
incumbent solution: [ 0.22262794, -0.5899061, 1.72267148, 0.28048429, -0.591445
94]
std deviation: [6.86330878e-05, 7.61545043e-05, 4.15931693e-04, 1.02614155e-04,
1.01049478e-04]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:28
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.783329689125868e-02 1.0e+00 1.98e-01  2e-01  2e-01 0:00.1
    2     16 -1.419006934921008e-02 1.3e+00 1.97e-01  2e-01  2e-01 0:00.3
    3     24 -1.447852824855644e-02 1.5e+00 1.99e-01  2e-01  2e-01 0:00.4
   24    192 -7.497812014970862e-02 3.9e+00 1.36e-01  7e-02  1e-01 0:03.6
   51    408 -8.037295372790698e-02 1.1e+01 2.07e-02  3e-03  2e-02 0:07.6
   80    640 -8.185130526138629e-02 2.1e+01 1.06e-02  5e-04  8e-03 0:12.7
  100    800 -8.191234640912715e-02 2.9e+01 7.01e-03  4e-04  5e-03 0:15.6
  137   1096 -8.207814400171642e-02 3.3e+01 1.35e-03  4e-05  5e-04 0:21.7
termination on tolflatfitness=1 (Tue Jul 22 12:59:50 2025)
final/bestever f-value = -8.207814e-02 -8.211606e-02 after 1097/839 evaluations
incumbent solution: [ 0.00867823, -1.03216231, 0.83303812, -0.01149252, 0.14850
349]
std deviation: [3.79335305e-05, 1.00541780e-04, 5.42496639e-04, 5.51656557e-05,
3.52289265e-04]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:50
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -6.532244644382068e-03 1.0e+00 1.78e-01  2e-01  2e-01 0:00.1
    2     16 -5.821418832256953e-03 1.1e+00 1.93e-01  2e-01  2e-01 0:00.3
    3     24 -9.128150324410278e-03 1.4e+00 1.89e-01  2e-01  2e-01 0:00.4
```

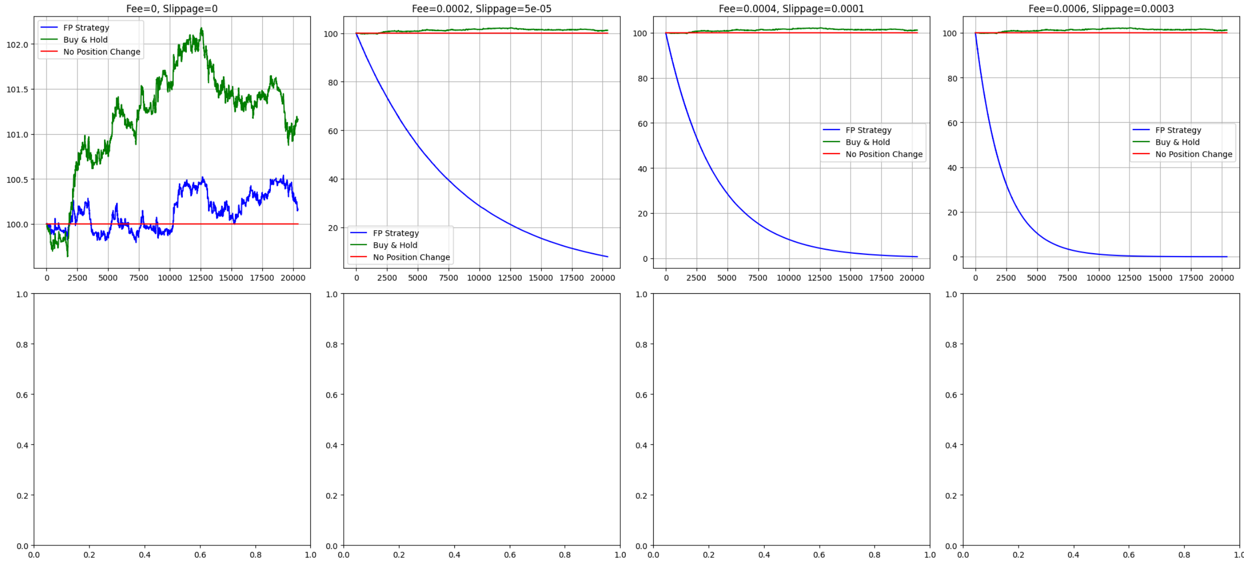
```
   24    192 -1.564832106462610e-02 3.2e+00 1.53e-01  7e-02  2e-01 0:03.6
   50    400 -7.667281219696243e-02 5.5e+00 2.15e-01  7e-02  3e-01 0:07.6
   79    632 -7.951890858192634e-02 1.9e+01 3.17e-02  5e-03  4e-02 0:12.7
  100    800 -8.066846665244487e-02 4.4e+01 1.78e-02  3e-03  3e-02 0:15.8
  144   1152 -8.209796969671768e-02 1.7e+02 2.99e-03  3e-04  4e-03 0:22.9
  159   1272 -8.221041305349530e-02 1.7e+02 1.20e-03  7e-05  1e-03 0:25.2
termination on tolflatfitness=1 (Tue Jul 22 13:00:15 2025)
final/bestever f-value = -8.221041e-02 -8.221041e-02 after 1273/1010 evaluation
s
incumbent solution: [ 0.19426715, -0.65165609, 1.14429051, 0.28199817, -1.07530
393]
std deviation: [2.37657203e-04, 6.78916586e-05, 1.45238761e-03, 3.36790304e-04,
1.01560493e-04]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.628203093512415e-02 1.0e+00 1.84e-01  2e-01  2e-01 0:00.1
    2     16 -1.911774877232020e-02 1.2e+00 1.94e-01  2e-01  2e-01 0:00.3
    3     24 -1.956328830399912e-02 1.5e+00 2.06e-01  2e-01  2e-01 0:00.4
   23    184 -2.213320085491066e-02 3.6e+00 6.01e-02  3e-02  7e-02 0:03.5
   49    392 -2.828172079791536e-02 6.3e+00 9.37e-03  2e-03  6e-03 0:07.6
   77    616 -2.930925957955388e-02 1.9e+01 1.02e-03  8e-05  7e-04 0:12.6
   81    648 -2.930925957955388e-02 2.0e+01 6.98e-04  5e-05  4e-04 0:13.2
termination on tolflatfitness=1 (Tue Jul 22 13:00:29 2025)
final/bestever f-value = -2.930926e-02 -2.937874e-02 after 649/400 evaluations
incumbent solution: [-0.2408623, 0.00398416, 0.1145548, -0.61979474, -0.5059085
1]
std deviation: [2.23023881e-04, 4.96544587e-05, 3.58501438e-04, 4.37194695e-04,
8.08868601e-05]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:43
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.745176153431678e-03 1.0e+00 2.08e-01  2e-01  2e-01 0:00.2
    2     16 -6.875797802492783e-03 1.5e+00 1.81e-01  2e-01  2e-01 0:00.3
    3     24 -6.875797802492783e-03 1.4e+00 1.74e-01  2e-01  2e-01 0:00.5
   14    112 -6.875797802492783e-03 2.6e+00 1.73e-01  1e-01  2e-01 0:02.1
termination on tolflatfitness=1 (Tue Jul 22 13:00:45 2025)
final/bestever f-value = -6.875798e-03 -2.538024e-02 after 113/80 evaluations
incumbent solution: [7.22121949e-01, 4.88680615e-02, 1.87115159e-01, 1.22101891
e-01, 7.31012990e-06]
std deviation: [0.24085065, 0.09931601, 0.13741908, 0.16705811, 0.14526097]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:45
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.300626293932647e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.1
    2     16 -1.300626293932647e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.3
    3     24 -1.300626293932647e-02 1.5e+00 1.87e-01  2e-01  2e-01 0:00.5
   10     80 -1.300626293932647e-02 3.3e+00 2.47e-01  2e-01  3e-01 0:01.9
termination on tolfunhist=1e-12 (Tue Jul 22 13:00:47 2025)
final/bestever f-value = -1.300626e-02 -1.300626e-02 after 81/5 evaluations
incumbent solution: [ 1.0419998, 0.01521702, 0.71696715, -0.28619781, -0.082453
88]
std deviation: [0.30399052, 0.17454078, 0.27594261, 0.25053517, 0.18779803]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:47
```

```
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.394006934921008e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.3
    2     16 -1.394006934921008e-02 1.5e+00 1.96e-01  2e-01  2e-01 0:00.6
    3     24 -1.394006934921008e-02 1.5e+00 1.84e-01  2e-01  2e-01 0:00.8
   10     80 -1.394006934921008e-02 1.8e+00 1.45e-01  1e-01  2e-01 0:01.8
termination on tolfun=1e-11 (Tue Jul 22 13:00:49 2025)
termination on tolfunhist=1e-12 (Tue Jul 22 13:00:49 2025)
termination on tolflatfitness=1 (Tue Jul 22 13:00:49 2025)
final/bestever f-value = -1.394007e-02 -1.394007e-02 after 81/8 evaluations
incumbent solution: [-0.58828352, -0.20972505, 0.22538923, -0.05458561, 0.14432
27, ]
std deviation: [0.14171051, 0.1230159, 0.15742793, 0.09816249, 0.10451324]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.571418832256952e-03 1.0e+00 1.79e-01  2e-01  2e-01 0:00.2
    2     16 -5.571418832256952e-03 1.3e+00 1.86e-01  2e-01  2e-01 0:00.3
    3     24 -5.571418832256952e-03 1.5e+00 1.96e-01  2e-01  2e-01 0:00.5
   13    104 -5.571418832256952e-03 2.4e+00 1.38e-01  9e-02  2e-01 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 13:00:51 2025)
final/bestever f-value = -5.571419e-03 -8.248220e-03 after 105/70 evaluations
incumbent solution: [ 0.70483487, 0.0557058, 0.21882486, -0.12430461, 0.1927769
9]
std deviation: [0.17244018, 0.09203424, 0.12648577, 0.11503198, 0.08952109]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:51
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.443326597354105e-02 1.0e+00 2.05e-01  2e-01  2e-01 0:00.2
    2     16 -1.469033178507634e-02 1.4e+00 1.90e-01  2e-01  2e-01 0:00.3
    3     24 -1.417187608867940e-02 1.4e+00 1.76e-01  2e-01  2e-01 0:00.5
   20    160 -1.417187608867940e-02 3.2e+00 1.90e-01  1e-01  2e-01 0:03.1
termination on tolflatfitness=1 (Tue Jul 22 13:00:54 2025)
final/bestever f-value = -1.417188e-02 -2.370415e-02 after 161/92 evaluations
incumbent solution: [-0.54813632, -0.22376561, -0.37912847, 0.28811544, -0.2160
6262]
std deviation: [0.15541341, 0.17649633, 0.22479645, 0.12024936, 0.17030554]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:09
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.245176153431678e-03 1.0e+00 2.08e-01  2e-01  2e-01 0:00.2
    2     16 -6.625797802492783e-03 1.5e+00 1.83e-01  2e-01  2e-01 0:00.3
    3     24 -1.356791059518370e-02 1.3e+00 1.61e-01  1e-01  2e-01 0:00.4
   18    144 -1.749069287964422e-02 2.9e+00 6.96e-02  3e-02  7e-02 0:03.5
   44    352 -1.969740263571607e-02 1.9e+01 2.73e-02  2e-03  3e-02 0:07.5
   75    600 -1.980877260297687e-02 2.5e+02 4.31e-03  8e-05  6e-03 0:12.2
termination on tolflatfitness=1 (Tue Jul 22 13:01:21 2025)
final/bestever f-value = -1.980877e-02 -1.980877e-02 after 601/373 evaluations
incumbent solution: [-0.37046939, 0.31152308, 0.30056152, 0.00354163, -0.010149
72]
std deviation: [3.69070111e-03, 8.09410120e-05, 5.55710675e-03, 3.00765227e-03,
2.21254334e-03]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:21
2025)
```

```
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.275626293932647e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.3
    2     16 -1.275626293932647e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.5
    3     24 -1.275626293932647e-02 1.5e+00 1.87e-01  2e-01  2e-01 0:00.8
   10     80 -1.275626293932647e-02 2.9e+00 2.53e-01  2e-01  3e-01 0:02.1
termination on tolfunhist=1e-12 (Tue Jul 22 13:01:23 2025)
final/bestever f-value = -1.275626e-02 -1.275626e-02 after 81/5 evaluations
incumbent solution: [ 0.86531894, -0.07998184, 0.90551278, -0.38991147, 0.04833
927]
std deviation: [0.27821192, 0.16955246, 0.2671653, 0.28470604, 0.18370363]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:23
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.369006934921008e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.1
    2     16 -1.369006934921008e-02 1.5e+00 1.96e-01  2e-01  2e-01 0:00.3
    3     24 -1.369006934921008e-02 1.5e+00 1.84e-01  2e-01  2e-01 0:00.4
    9     72 -1.369006934921008e-02 1.7e+00 1.27e-01  1e-01  1e-01 0:01.3
termination on tolflatfitness=1 (Tue Jul 22 13:01:25 2025)
final/bestever f-value = -1.369007e-02 -1.369007e-02 after 73/8 evaluations
incumbent solution: [-0.34035253, 0.10965543, 0.16543877, 0.00123723, 0.0347148
9]
std deviation: [0.1239624, 0.09544405, 0.13414226, 0.10373471, 0.09803747]
(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 -5.321418832256952e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.1
    2     16 -5.321418832256952e-03 1.4e+00 1.61e-01  1e-01  2e-01 0:00.3
    3     24 -5.321418832256952e-03 1.3e+00 1.54e-01  1e-01  2e-01 0:00.4
   23    184 -9.960786737017613e-03 5.1e+00 9.53e-02  3e-02  1e-01 0:03.4
   50    400 -9.706316034702289e-03 1.8e+01 4.33e-02  5e-03  5e-02 0:07.5
   79    632 -9.791146797381578e-03 1.8e+02 2.00e-02  2e-03  3e-02 0:12.6
   83    664 -9.791146797381578e-03 2.6e+02 2.57e-02  2e-03  4e-02 0:13.2
termination on tolfunhist=1e-12 (Tue Jul 22 13:01:38 2025)
final/bestever f-value = -9.791147e-03 -1.704053e-02 after 665/45 evaluations
incumbent solution: [ 0.20851973, 0.21782998, 0.07708991, -1.22691106, 0.256216
7, ]
std deviation: [0.00680494, 0.00236935, 0.04424797, 0.0343872, 0.01199571]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:38
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.393326597354105e-02 1.0e+00 2.05e-01  2e-01  2e-01 0:00.1
    2     16 -1.419033178507634e-02 1.4e+00 1.90e-01  2e-01  2e-01 0:00.3
    3     24 -1.392187608867940e-02 1.4e+00 1.76e-01  2e-01  2e-01 0:00.4
   24    192 -1.392187608867940e-02 3.0e+00 2.13e-01  1e-01  2e-01 0:03.5
   27    216 -1.392187608867940e-02 3.1e+00 2.16e-01  1e-01  2e-01 0:04.0
termination on tolflatfitness=1 (Tue Jul 22 13:01:42 2025)
final/bestever f-value = -1.392188e-02 -1.556587e-02 after 217/54 evaluations
incumbent solution: [-1.57872665, -0.13133292, -0.67859886, 0.97901433, -0.0810
0654]
std deviation: [0.21246903, 0.1324738, 0.19935086, 0.16738534, 0.09943087]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:56
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.445176153431679e-03 1.0e+00 2.01e-01  2e-01  2e-01 0:00.2
```

```
    2     16 -6.225797802492784e-03 1.4e+00 1.79e-01  2e-01  2e-01 0:00.5
    3     24 -6.225797802492784e-03 1.5e+00 1.52e-01  1e-01  2e-01 0:00.7
    9     72 -6.225797802492784e-03 2.3e+00 1.63e-01  1e-01  2e-01 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 13:01:58 2025)
final/bestever f-value = -6.225798e-03 -8.445176e-03 after 73/7 evaluations
incumbent solution: [ 0.52232733, -0.07684968, -0.11059524, 0.04449018, 0.11411
89, ]
std deviation: [0.18721964, 0.11908653, 0.15148295, 0.20927703, 0.11675418]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:58
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.235626293932647e-02 1.0e+00 1.79e-01  2e-01  2e-01 0:00.1
    2     16 -1.235626293932647e-02 1.3e+00 1.86e-01  2e-01  2e-01 0:00.3
    3     24 -1.235626293932647e-02 1.5e+00 1.87e-01  2e-01  2e-01 0:00.4
    9     72 -1.235626293932647e-02 2.3e+00 3.45e-01  3e-01  4e-01 0:01.3
termination on tolfun=1e-11 (Tue Jul 22 13:01:59 2025)
final/bestever f-value = -1.235626e-02 -1.235626e-02 after 73/5 evaluations
incumbent solution: [ 1.29771249, 0.10350327, 0.88081023, -0.55772687, 0.265712
03]
std deviation: [0.38095374, 0.27216868, 0.36902442, 0.38103614, 0.27709404]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:59
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.329006934921008e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.1
    2     16 -1.329006934921008e-02 1.5e+00 2.05e-01  2e-01  2e-01 0:00.3
    3     24 -1.329006934921008e-02 1.5e+00 1.96e-01  2e-01  2e-01 0:00.4
    9     72 -1.329006934921008e-02 1.9e+00 1.61e-01  1e-01  2e-01 0:01.3
termination on tolfun=1e-11 (Tue Jul 22 13:02:00 2025)
termination on tolflatfitness=1 (Tue Jul 22 13:02:00 2025)
final/bestever f-value = -1.329007e-02 -1.329007e-02 after 73/8 evaluations
incumbent solution: [-0.53659361, -0.30081474, -0.2691636, 0.08738865, -0.03137
28, ]
std deviation: [0.1377929, 0.16057964, 0.17170565, 0.14977745, 0.11961344]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:00
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.921418832256953e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.1
    2     16 -4.921418832256953e-03 1.4e+00 1.61e-01  1e-01  2e-01 0:00.3
    3     24 -4.921418832256953e-03 1.3e+00 1.54e-01  1e-01  2e-01 0:00.4
   19    152 -4.921418832256953e-03 3.1e+00 1.90e-01  1e-01  2e-01 0:02.8
termination on tolflatfitness=1 (Tue Jul 22 13:02:03 2025)
final/bestever f-value = -4.921419e-03 -8.906316e-03 after 153/58 evaluations
incumbent solution: [ 0.82469315, -0.2208162, -0.11615597, -0.25381292, -0.0685
241, ]
std deviation: [0.17385807, 0.14345683, 0.18524739, 0.19907112, 0.13843422]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:03
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.313326597354105e-02 1.0e+00 2.05e-01  2e-01  2e-01 0:00.1
    2     16 -1.352187608867940e-02 1.4e+00 1.73e-01  2e-01  2e-01 0:00.3
    3     24 -1.352187608867940e-02 1.5e+00 1.51e-01  1e-01  2e-01 0:00.4
   16    128 -1.352187608867940e-02 2.9e+00 1.32e-01  9e-02  2e-01 0:02.4
termination on tolflatfitness=1 (Tue Jul 22 13:02:06 2025)
final/bestever f-value = -1.352188e-02 -1.352780e-02 after 129/26 evaluations
```

```
incumbent solution: [-0.35252887, -0.36601629, -0.13885536, -0.26396282, 0.2545
356, ]
std deviation: [0.11502588, 0.16832825, 0.10266575, 0.10878295, 0.09070207]
```




```
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           100.16           0.16          101.16
1.16    100.0             0.0
0.0002   0.00005             7.85         -92.15          101.16
1.16    100.0             0.0
0.0004   0.00010             0.61         -99.39          101.16
1.16    100.0             0.0
0.0006   0.00030             0.01         -99.99          101.16
1.16    100.0             0.0
```

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
```

```python
    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)
```

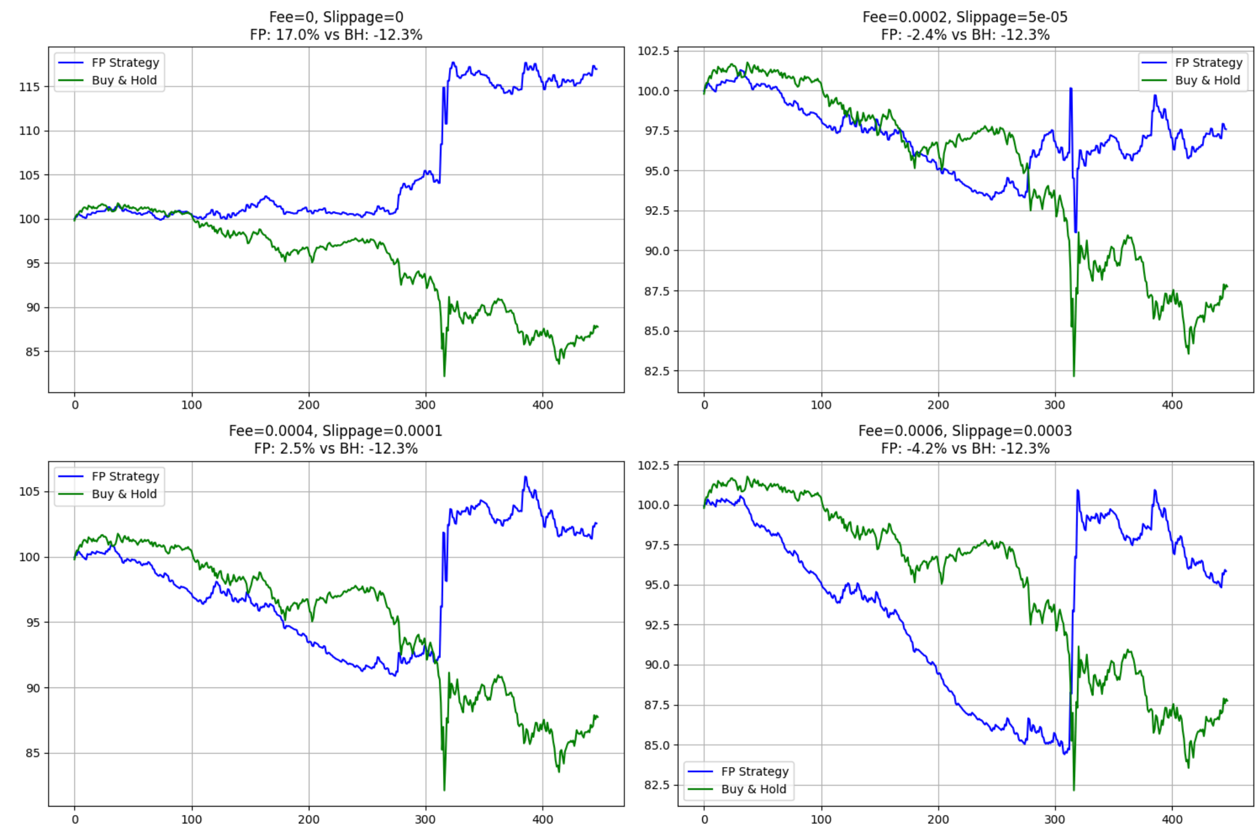
```
    return positions, trades

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
```

```python
    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_trac
    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
```

```python
df_train, df_cv, df_test = prepare_data("ETH_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          16.98      22.80            -0.12         -12.26     -1
5.51           -17.87
0.00      0.00          -2.42      -3.55            -8.87         -12.26     -1
5.51           -17.87
0.00      0.00           2.54       3.63            -9.12         -12.26     -1
5.51           -17.87
0.00      0.00          -4.16      -6.11           -15.61         -12.26     -1
5.51           -17.87
```

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("ETH_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)
```

```python
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
df['spread'] = df['ask_price_0'] - df['bid_price_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'][t-1]
        dobi_t = features['dobi'][t-1]
        depth_t = features['depth'][t-1]
        slope_t = features['queue_slope'][t-1]
        spread_t = features['spread'][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 = [(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][['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']]
```

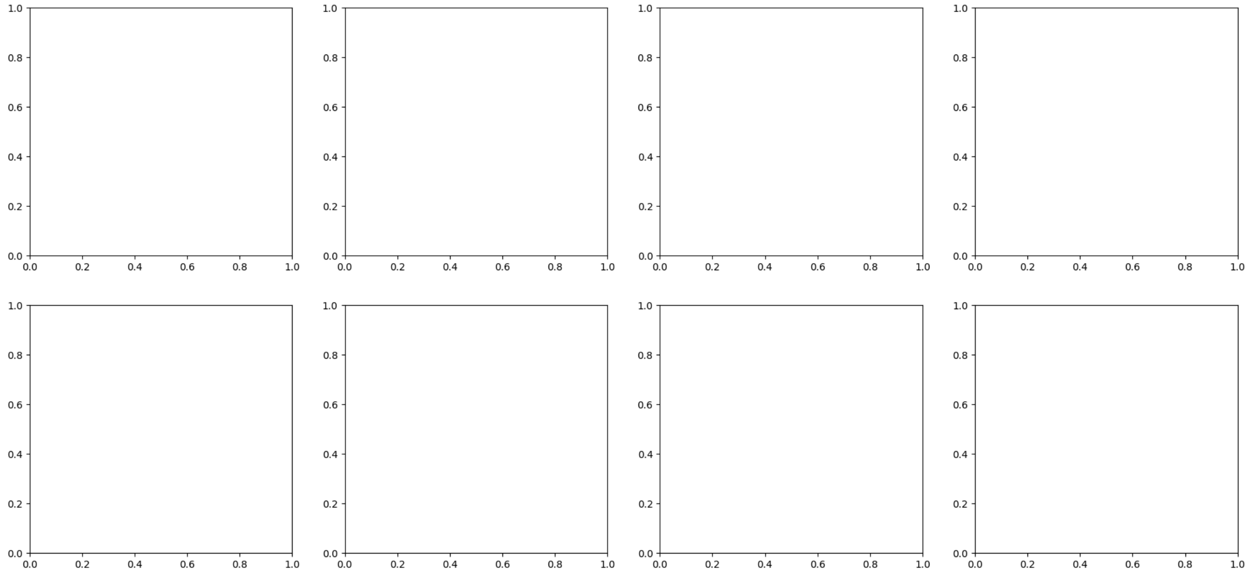
```python
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        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:
        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_trac
    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))
```

```
(5_w,10)-aCMA-ES (mu_w=3.2,w_1=45%) in dimension 9 (seed=42, Wed Jul 23 09:39:1
9 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     10 -1.606676083059089e-02 1.0e+00 1.87e-01  2e-01  2e-01 0:00.2
    2     20 -2.142526964787894e-02 1.1e+00 1.78e-01  2e-01  2e-01 0:00.4
    3     30 -2.142526964787894e-02 1.2e+00 1.67e-01  2e-01  2e-01 0:00.7
    5     50 -1.606676083059089e-02 1.4e+00 1.68e-01  2e-01  2e-01 0:01.0
termination on tolflatfitness=1 (Wed Jul 23 09:39:20 2025)
final/bestever f-value = -1.606676e-02 -2.142527e-02 after 51/16 evaluations
incumbent solution: [ 0.16910801  0.08138616  0.09185534 -0.05075645  0.3153814
6 -0.05509622
  0.00066397  0.4387675  ...]
std deviations: [0.16580245 0.154616   0.15855012 0.15352079 0.16454697 0.16337
627
 0.16196324 0.17747393 ...]
(5_w,10)-aCMA-ES (mu_w=3.2,w_1=45%) in dimension 9 (seed=42, Wed Jul 23 09:39:2
0 2025)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/range.py in get_lo
c(self, key)
    412             try:
--> 413                 return self._range.index(new_key)
    414             except ValueError as err:

ValueError: 0 is not in range

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/tmp/ipython-input-4-4003313612.py in <cell line: 0>()
     98         if end > len(df_train):
     99             continue
--> 100         mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, s
lip)
    101         signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:en
d][['obi', 'dobi', 'depth', 'queue_slope', 'spread']], end-start, 1.0)
    102         threshold = optimize_threshold(signal, df_train.iloc[start:en
d]['returns'].values, fee, slip)

/tmp/ipython-input-4-4003313612.py in train_fp_model(df_slice, fee, slip)
     82         pos, trades = trading_strategy(signal, 0.005)
     83         return -np.sum(apply_trading_costs(pos, trades, returns, fee, s
lip))
---> 84     res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, option
s={'seed':random_seed})
     85     return res[0][:7], res[0][7:]
     86

/usr/local/lib/python3.11/dist-packages/cma/evolution_strategy.py in fmin(objec
tive_function, x0, sigma0, options, args, gradf, restarts, restart_from_best, i
ncpopsize, eval_initial_x, parallel_objective, noise_handler, noise_change_sigm
a_exponent, noise_kappa_exponent, bipop, callback, init_callback)
   4227                 while not es.stop():  # iteration loop
   4228                     # X, fit = eval_in_parallel(lambda: es.ask(1)[0], e
s.popsize, args, repetitions=noisehandler.evaluations-1)
-> 4229                     X, fit = es.ask_and_eval(parallel_objective or obje
ctive_function,
   4230                                              args, gradf=gradf,
   4231                                              evaluations=noisehandler.e
valuations,

/usr/local/lib/python3.11/dist-packages/cma/evolution_strategy.py in ask_and_ev
al(self, func, args, gradf, number, xmean, sigma_fac, evaluations, aggregation,
kappa, parallel_mode)
   1917                     # self.more_to_write += [length_normalizer * 1e-3,
length_normalizer * self.mahalanobis_norm(x - xmean) * 1e2]
   1918
-> 1919                 f = func(x, *args) if kappa == 1 else \
   1920                     func(xmean + kappa * length_normalizer * (x - xmea
n),
```

```
   1921                              *args)

/tmp/ipython-input-4-4003313612.py in objective(params)
     79         mu_params = params[:7]
     80         sigma_params = params[7:]
---> 81         signal = simulate_fp(mu_params, sigma_params, x_init, feature
s, len(returns), dt)
     82         pos, trades = trading_strategy(signal, 0.005)
     83         return -np.sum(apply_trading_costs(pos, trades, returns, fee, s
lip))

/tmp/ipython-input-4-4003313612.py in simulate_fp(mu_params, sigma_params, x0,
features, timesteps, dt)
     49     rng = np.random.RandomState(random_seed)
     50     for t in range(1, timesteps):
---> 51         obi_t = features['obi'][t-1]
     52         dobi_t = features['dobi'][t-1]
     53         depth_t = features['depth'][t-1]

/usr/local/lib/python3.11/dist-packages/pandas/core/series.py in __getitem__(se
lf, key)
   1119
   1120         elif key_is_scalar:
-> 1121             return self._get_value(key)
   1122
   1123         # Convert generator to list before going through hashable part

/usr/local/lib/python3.11/dist-packages/pandas/core/series.py in _get_value(sel
f, label, takeable)
   1235
   1236         # Similar to Index.get_value, but we do not fall back to positi
onal
-> 1237         loc = self.index.get_loc(label)
   1238
   1239         if is_integer(loc):

/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/range.py in get_lo
c(self, key)
    413                 return self._range.index(new_key)
    414             except ValueError as err:
--> 415                 raise KeyError(key) from err
    416         if isinstance(key, Hashable):
    417             raise KeyError(key)

KeyError: 0
```



In [ ]:

```
for i in df.columns:
  print (i)
```

```
Unnamed: 0
system_time
midpoint
spread
buys
sells
bids_distance_0
bids_distance_1
bids_distance_2
bids_distance_3
bids_distance_4
bids_distance_5
bids_distance_6
bids_distance_7
bids_distance_8
bids_distance_9
bids_distance_10
bids_distance_11
bids_distance_12
bids_distance_13
bids_distance_14
bids_notional_0
bids_notional_1
bids_notional_2
bids_notional_3
bids_notional_4
bids_notional_5
bids_notional_6
bids_notional_7
bids_notional_8
bids_notional_9
bids_notional_10
bids_notional_11
bids_notional_12
bids_notional_13
bids_notional_14
bids_cancel_notional_0
bids_cancel_notional_1
bids_cancel_notional_2
bids_cancel_notional_3
bids_cancel_notional_4
bids_cancel_notional_5
bids_cancel_notional_6
bids_cancel_notional_7
bids_cancel_notional_8
bids_cancel_notional_9
bids_cancel_notional_10
bids_cancel_notional_11
bids_cancel_notional_12
bids_cancel_notional_13
bids_cancel_notional_14
bids_limit_notional_0
bids_limit_notional_1
bids_limit_notional_2
```

```
bids_limit_notional_3
bids_limit_notional_4
bids_limit_notional_5
bids_limit_notional_6
bids_limit_notional_7
bids_limit_notional_8
bids_limit_notional_9
bids_limit_notional_10
bids_limit_notional_11
bids_limit_notional_12
bids_limit_notional_13
bids_limit_notional_14
bids_market_notional_0
bids_market_notional_1
bids_market_notional_2
bids_market_notional_3
bids_market_notional_4
bids_market_notional_5
bids_market_notional_6
bids_market_notional_7
bids_market_notional_8
bids_market_notional_9
bids_market_notional_10
bids_market_notional_11
bids_market_notional_12
bids_market_notional_13
bids_market_notional_14
asks_distance_0
asks_distance_1
asks_distance_2
asks_distance_3
asks_distance_4
asks_distance_5
asks_distance_6
asks_distance_7
asks_distance_8
asks_distance_9
asks_distance_10
asks_distance_11
asks_distance_12
asks_distance_13
asks_distance_14
asks_notional_0
asks_notional_1
asks_notional_2
asks_notional_3
asks_notional_4
asks_notional_5
asks_notional_6
asks_notional_7
asks_notional_8
asks_notional_9
asks_notional_10
asks_notional_11
```

```
asks_notional_12
asks_notional_13
asks_notional_14
asks_cancel_notional_0
asks_cancel_notional_1
asks_cancel_notional_2
asks_cancel_notional_3
asks_cancel_notional_4
asks_cancel_notional_5
asks_cancel_notional_6
asks_cancel_notional_7
asks_cancel_notional_8
asks_cancel_notional_9
asks_cancel_notional_10
asks_cancel_notional_11
asks_cancel_notional_12
asks_cancel_notional_13
asks_cancel_notional_14
asks_limit_notional_0
asks_limit_notional_1
asks_limit_notional_2
asks_limit_notional_3
asks_limit_notional_4
asks_limit_notional_5
asks_limit_notional_6
asks_limit_notional_7
asks_limit_notional_8
asks_limit_notional_9
asks_limit_notional_10
asks_limit_notional_11
asks_limit_notional_12
asks_limit_notional_13
asks_limit_notional_14
asks_market_notional_0
asks_market_notional_1
asks_market_notional_2
asks_market_notional_3
asks_market_notional_4
asks_market_notional_5
asks_market_notional_6
asks_market_notional_7
asks_market_notional_8
asks_market_notional_9
asks_market_notional_10
asks_market_notional_11
asks_market_notional_12
asks_market_notional_13
asks_market_notional_14
bid_price_0
ask_price_0
bid_price_1
ask_price_1
bid_price_2
ask_price_2
```

```
bid_price_3
ask_price_3
bid_price_4
ask_price_4
bid_price_5
ask_price_5
bid_price_6
ask_price_6
bid_price_7
ask_price_7
bid_price_8
ask_price_8
bid_price_9
ask_price_9
bid_price_10
ask_price_10
bid_price_11
ask_price_11
bid_price_12
ask_price_12
bid_price_13
ask_price_13
bid_price_14
ask_price_14
obi
dobi
depth
queue_slope
```

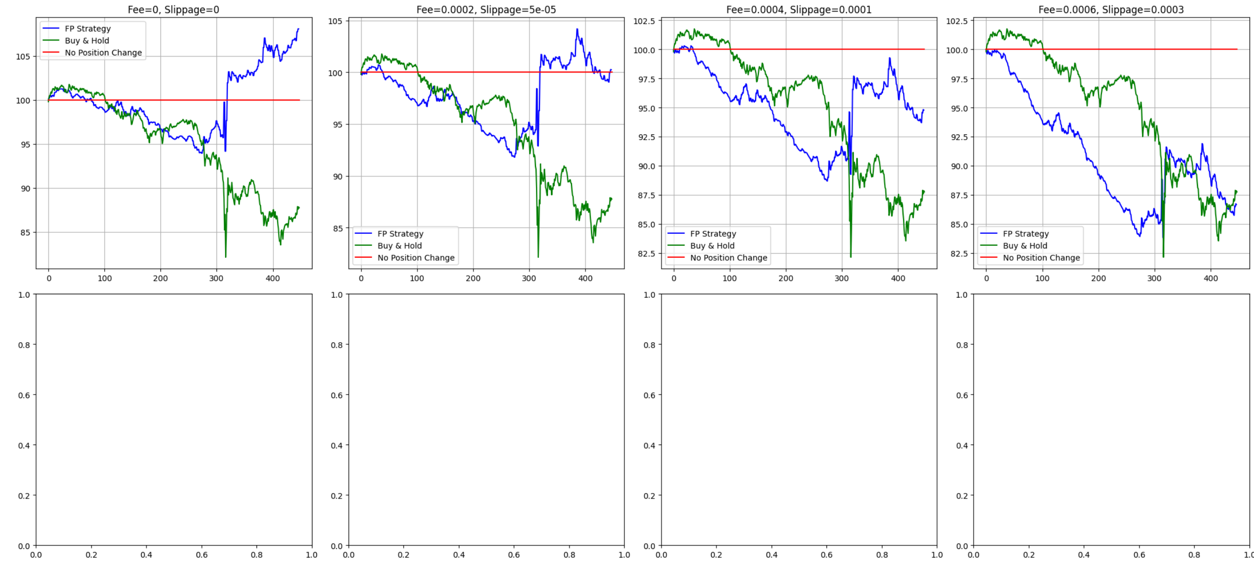
In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("ETH_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_trac
    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-8-1345527685.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           108.07           8.07           87.74
-12.26    100.0             0.0
0.0002   0.00005           100.24           0.24           87.74
-12.26    100.0             0.0
0.0004   0.00010            94.78          -5.22           87.74
-12.26    100.0             0.0
0.0006   0.00030            86.65         -13.35           87.74
-12.26    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("ETH_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)
```

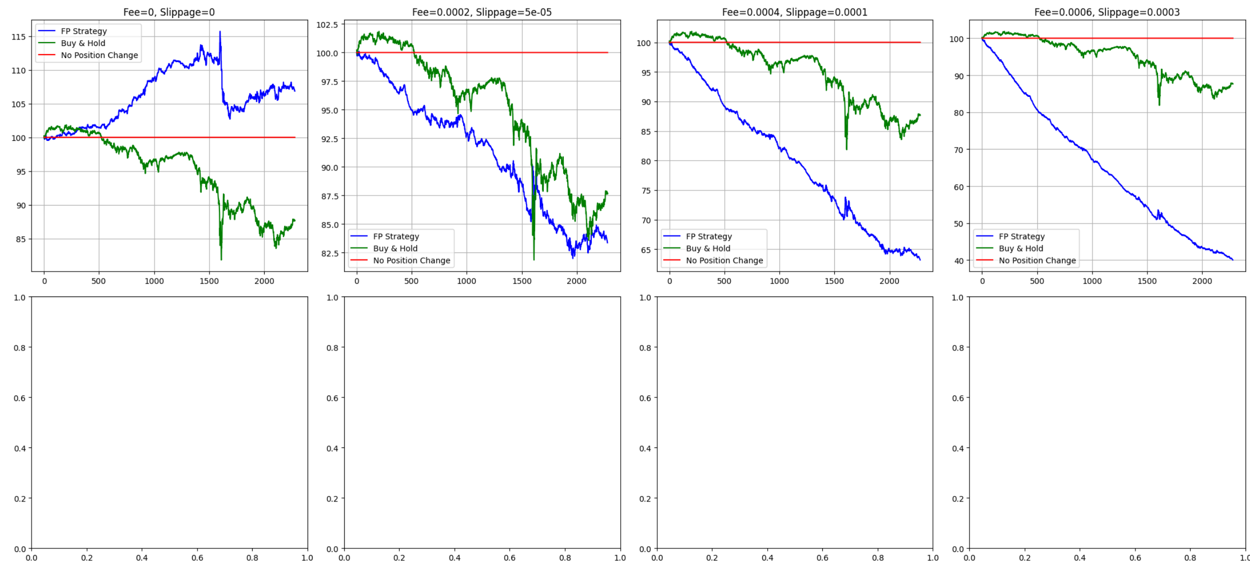
```
train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

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][['o
            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
```

```python
        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_trac
    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()
```

```python
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-12-2841790008.py:21: 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)
/tmp/ipython-input-12-2841790008.py:58: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5 * slop
e_t + a6 * spread_t)
/tmp/ipython-input-12-2841790008.py:59: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: invalid value encounter
ed in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:58: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5 * slop
e_t + a6 * spread_t)
/tmp/ipython-input-12-2841790008.py:59: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: invalid value encounter
ed in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: invalid value encounter
ed in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:58: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5 * slop
e_t + a6 * spread_t)
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-12-2841790008.py:60: RuntimeWarning: invalid value encounter
ed in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
```




```
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           106.84           6.84           87.65
-12.35    100.0             0.0
0.0002   0.00005            83.35         -16.65           87.65
-12.35    100.0             0.0
0.0004   0.00010            63.18         -36.82           87.65
-12.35    100.0             0.0
0.0006   0.00030            40.04         -59.96           87.65
-12.35    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("ETH_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)
```

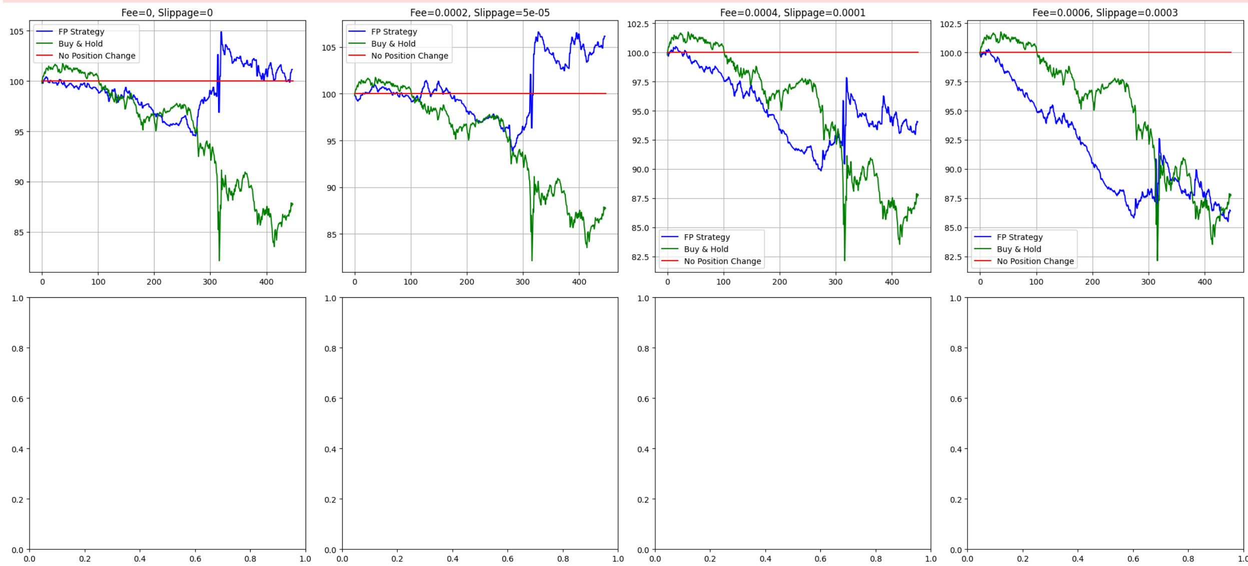
```
df['abs_dobi'] = df['dobi'].abs()

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][['o
            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', 'net_queue_slope', 'sprea
    test_positions = []
```

```python
    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_trac
    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-2-1122171351.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 Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           101.16           1.16           87.74
-12.26    100.0    0.0
0.0002   0.00005           106.15           6.15           87.74
-12.26    100.0    0.0
0.0004   0.00010            94.04          -5.96           87.74
-12.26    100.0    0.0
0.0006   0.00030            86.41         -13.59           87.74
-12.26    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("ETH_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}']
```

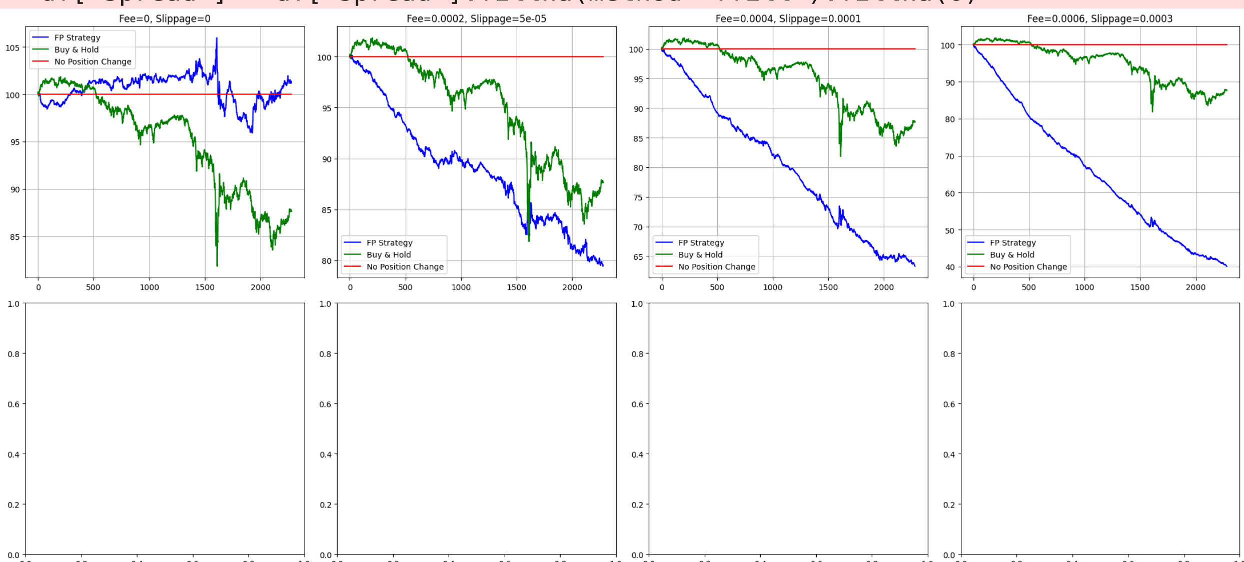
```
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['net_queue_slope'] = (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: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', '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_trac
    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-3-2079363087.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           101.36           1.36           87.65
-12.35    100.0             0.0
0.0002   0.00005            79.53         -20.47           87.65
-12.35    100.0             0.0
0.0004   0.00010            63.29         -36.71           87.65
-12.35    100.0             0.0
0.0006   0.00030            40.11         -59.89           87.65
-12.35    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("ETH_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_co
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)

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 = 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):
        f = features.iloc[t-1]
```

```
        mu = a0 + a1 * x[t-1] + a2 * f['obi'] + a3 * f['dobi'] + a4 * f['depth
        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', 'spread', 'queue_slope_bid',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:8]
        sigma_params = params[8:]
        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]*8 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:8], res[0][8:]

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', 'spread', 'queue_slope_bi
    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_trac
    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')
```

```python
    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-4-1011596047.py:23: 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)
```