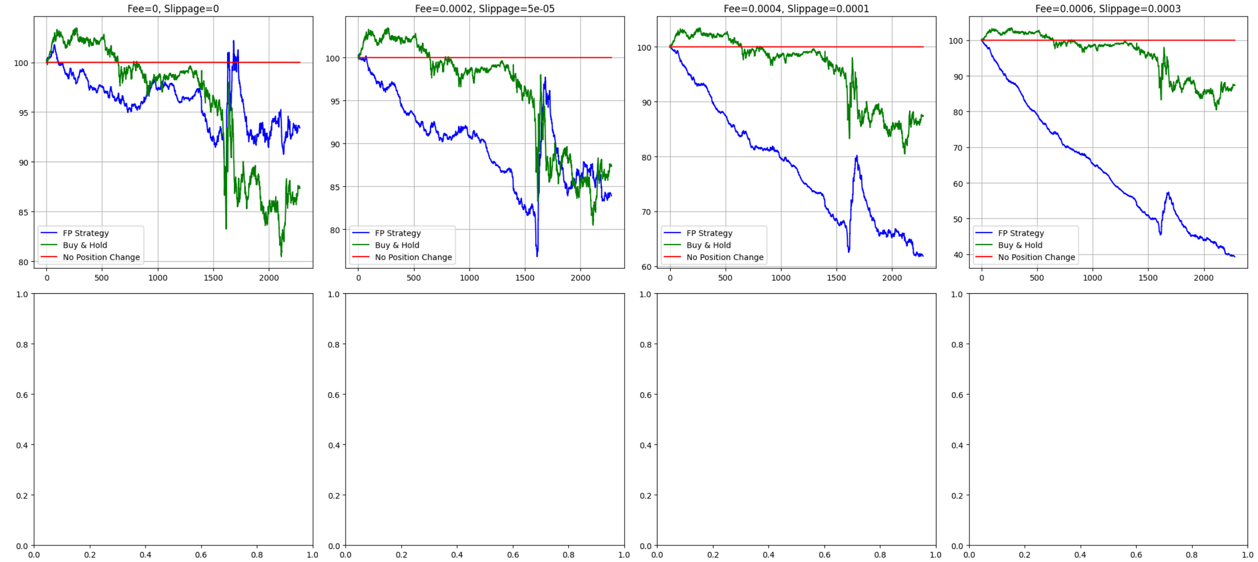
In [ ]:

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

np.random.seed(42)
random_seed = 42

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

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

df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_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)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = df['dobi'].abs()

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

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

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

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

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

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

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

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

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+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 = np.abs(first_position) * (fee + slip) if first_positi
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

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

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

plt.tight_layout()
plt.show()

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

```
/tmp/ipython-input-7-2599974331.py:25: 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)
/tmp/ipython-input-7-2599974331.py:66: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_slope +
a6 * spread + a7 * depth_var + a8 * abs_dobi + a9 * np.sign(x[t-1]))
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:40: RuntimeWarning: overflow encountered in
divide
  positions = np.tanh(signal / threshold)
/tmp/ipython-input-7-2599974331.py:67: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:40: RuntimeWarning: overflow encountered in
divide
  positions = np.tanh(signal / threshold)
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:66: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_slope +
a6 * spread + a7 * depth_var + a8 * abs_dobi + a9 * np.sign(x[t-1]))
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:40: RuntimeWarning: overflow encountered in
divide
  positions = np.tanh(signal / threshold)
/tmp/ipython-input-7-2599974331.py:66: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_slope +
a6 * spread + a7 * depth_var + a8 * abs_dobi + a9 * np.sign(x[t-1]))
/tmp/ipython-input-7-2599974331.py:67: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered in
scalar add
```

```
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:40: RuntimeWarning: overflow encountered in
divide
  positions = np.tanh(signal / threshold)
/tmp/ipython-input-7-2599974331.py:66: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_slope +
a6 * spread + a7 * depth_var + a8 * abs_dobi + a9 * np.sign(x[t-1]))
/tmp/ipython-input-7-2599974331.py:67: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:40: RuntimeWarning: overflow encountered in
divide
  positions = np.tanh(signal / threshold)
/tmp/ipython-input-7-2599974331.py:66: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_slope +
a6 * spread + a7 * depth_var + a8 * abs_dobi + a9 * np.sign(x[t-1]))
/tmp/ipython-input-7-2599974331.py:67: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-2599974331.py:68: RuntimeWarning: overflow encountered 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            93.40          -6.60           87.38
-12.62    100.0             0.0
0.0002   0.00005            83.89         -16.11           87.38
-12.62    100.0             0.0
0.0004   0.00010            61.82         -38.18           87.38
-12.62    100.0             0.0
0.0006   0.00030            39.26         -60.74           87.38
-12.62    100.0             0.0
```

In [ ]:

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

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

df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope_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()
df
```

```
/tmp/ipython-input-6-3011815859.py:17: 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)
```

Out[ ]:

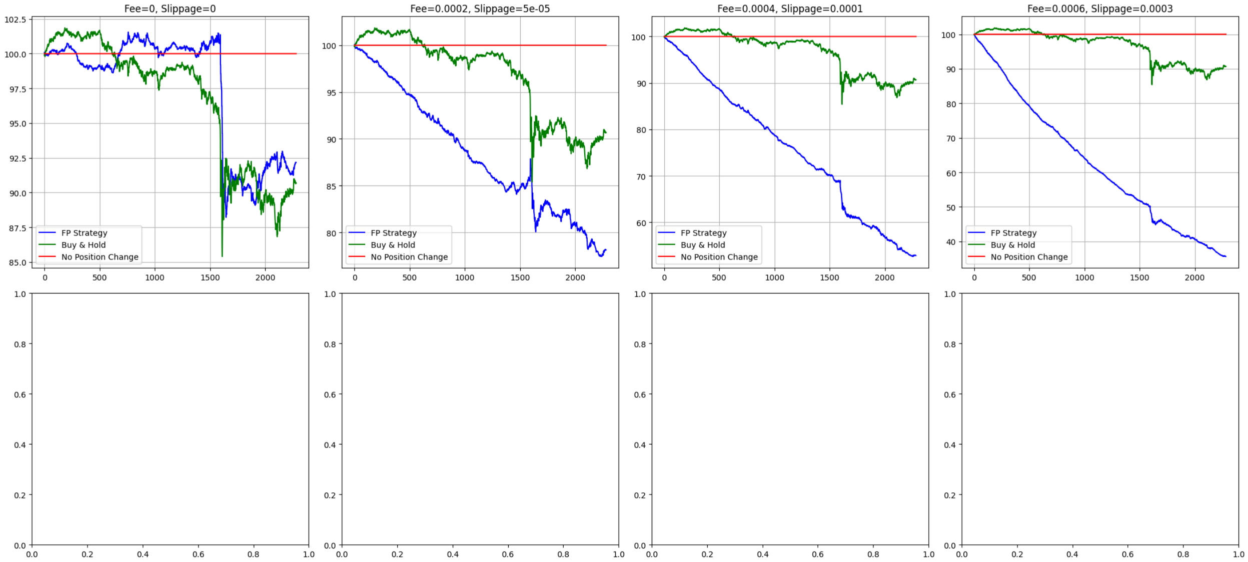
```
       Unnamed:
              0                 system_time  midpoint  spread           buys
    0         0                  2021-04-07
                   11:33:59.055697+00:00   1.16205     0.0   56936.467913  2582
    1         1                  2021-04-07
                   11:34:59.055697+00:00   1.16800     0.0   56491.336799   786
    2         2                  2021-04-07
                   11:35:59.055697+00:00   1.17530     0.0   52859.493359   484
    3         3                  2021-04-07
                   11:36:59.055697+00:00   1.16585     0.0   50772.386336   326
    4         4                  2021-04-07
                   11:37:59.055697+00:00   1.17255     0.0  113579.364184   825
  ...       ...                         ...       ...     ...            ...
17104     17104                  2021-04-19
                   09:45:00.442103+00:00   1.27325     0.0   13671.251598   253
17105     17105                  2021-04-19
                   09:46:00.442103+00:00   1.27200     0.0    9916.946518   336
17106     17106                  2021-04-19
                   09:47:00.442103+00:00   1.27255     0.0   32589.054204   434
17107     17107                  2021-04-19
                   09:48:00.442103+00:00   1.27305     0.0    3437.251449    79
17108     17108                  2021-04-19
                   09:49:00.442103+00:00   1.27105     0.0   10510.439494    68

17109 rows × 194 columns
```

In [ ]:

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

np.random.seed(42)
random_seed = 42

# Load and preprocess data
df = pd.read_csv("BTC_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_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)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = df['dobi'].abs()

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

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

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

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

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

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

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

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

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+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:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

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

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_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 = np.abs(first_position) * (fee + slip) if first_positi
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

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

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

plt.tight_layout()
plt.show()

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

```
/tmp/ipython-input-4-4261587749.py:25: 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            92.16          -7.84           90.67
-9.33    100.0             0.0
0.0002   0.00005            78.13         -21.87           90.67
-9.33    100.0             0.0
0.0004   0.00010            52.83         -47.17           90.67
-9.33    100.0             0.0
0.0006   0.00030            35.62         -64.38           90.67
-9.33    100.0             0.0
```

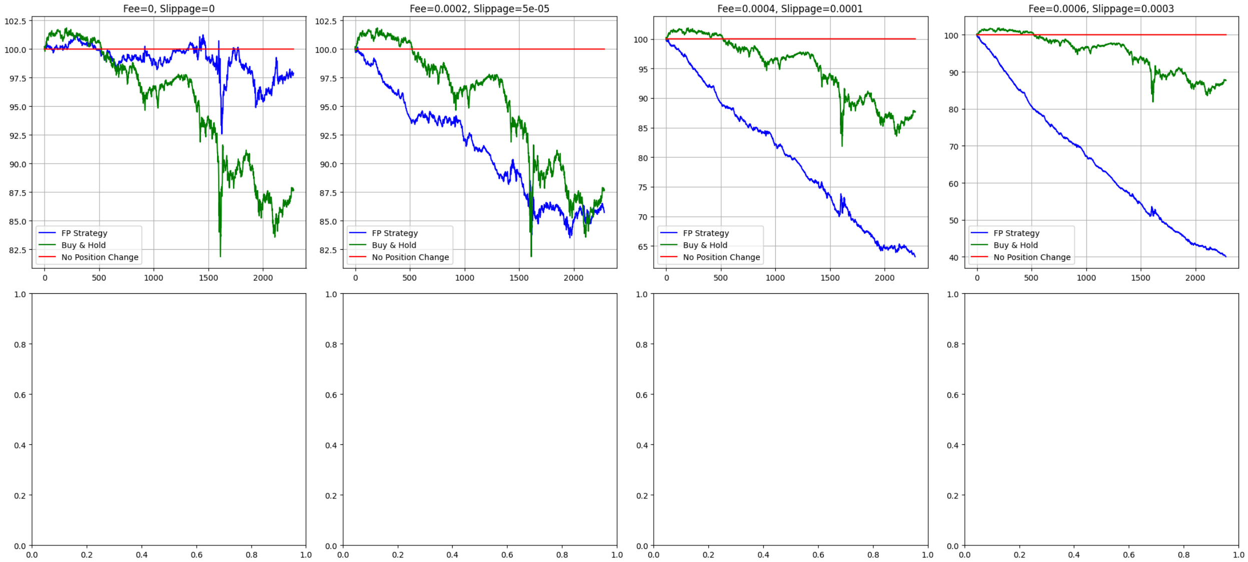
In [ ]:

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

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

```
# Load and preprocess data
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_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)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = df['dobi'].abs()

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

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

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

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

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

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

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

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

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+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)
```

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

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

    if not test_positions:
        continue

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

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

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

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

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = np.abs(first_position) * (fee + slip) if first_positi
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))
    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

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

plt.tight_layout()
plt.show()

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

```
/tmp/ipython-input-6-3446055485.py:25: 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)
/tmp/ipython-input-6-3446055485.py:66: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_slope +
a6 * spread + a7 * depth_var + a8 * abs_dobi + a9 * np.sign(x[t-1]))
/tmp/ipython-input-6-3446055485.py:68: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-6-3446055485.py:40: RuntimeWarning: overflow encountered in
divide
  positions = np.tanh(signal / threshold)
/tmp/ipython-input-6-3446055485.py:67: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
```



```
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             97.90          -2.10           87.65
-12.35    100.0              0.0
0.0002   0.00005             85.71         -14.29           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
```