In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import minimize
from scipy.stats import norm
import requests
import time
import os
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Set visualization style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (14, 8)

print("="*80)
print("ATTF CHECKPOINT B: PORTFOLIO OPTIMIZATION WITH MONTE CARLO")
print("="*80)
```

```
================================================================================
ATTF CHECKPOINT B: PORTFOLIO OPTIMIZATION WITH MONTE CARLO
================================================================================
```

In [2]:
```
#================================================================================
# CONFIGURATION
#================================================================================

# Load API key
from dotenv import load_dotenv
load_dotenv('PolygonAPI.env')
POLYGON_API_KEY = os.getenv("POLYGON_API_KEY", "")
print(f"\n✓ API Key loaded: {POLYGON_API_KEY[:10]}..." if POLYGON_API_KEY else "❌

# 22-stock portfolio (Option C)
ALL_TICKERS = {
    'AI Infrastructure': ['NVDA', 'AMD', 'MSFT', 'QCOM', 'CORW', 'NBIS', 'ORCL'],
    'Robotics & Space': ['TSLA', 'JOBY', 'RKLB', 'ASTS', 'BA'],
    'Crypto': ['IREN', 'CORZ', 'IBIT', 'COIN', 'CLSK'],
    'Quantum': ['IONQ', 'RGTI', 'IBM', 'GOOG', 'HON']
}

FLAT_TICKERS = [t for sleeve in ALL_TICKERS.values() for t in sleeve]

# Backtest parameters
START_DATE = '2015-01-01'
END_DATE = '2025-10-11'
INITIAL_CAPITAL = 100000

# Checkpoint A allocation (40/30/20/10)
CHECKPOINT_A_SLEEVE_WEIGHTS = {
    'AI Infrastructure': 0.40,
    'Robotics & Space': 0.30,
    'Crypto': 0.20,
```

```
    'Quantum': 0.10
}

# Fee structure
ANNUAL_MGMT_FEE = 0.015  # 1.5% management fee
TRANSACTION_COST_BPS = 15  # 15 basis points per trade

# Risk-free rate (3-month T-bill average for period)
RISK_FREE_RATE = 0.04  # 4% annual

print(f"\n📊 Portfolio Configuration:")
print(f"Total stocks: {len(FLAT_TICKERS)}")
for sleeve_name, tickers in ALL_TICKERS.items():
    print(f"  {sleeve_name} ({len(tickers)}): {', '.join(tickers)}")
```

```
✓ API Key loaded: i0mFBg4ssj...

📊 Portfolio Configuration:
Total stocks: 22
  AI Infrastructure (7): NVDA, AMD, MSFT, QCOM, CORW, NBIS, ORCL
  Robotics & Space (5): TSLA, JOBY, RKLB, ASTS, BA
  Crypto (5): IREN, CORZ, IBIT, COIN, CLSK
  Quantum (5): IONQ, RGTI, IBM, GOOG, HON
```

In [3]:

```
#============================================================================
# DATA DOWNLOAD WITH DIVIDEND ADJUSTMENTS
#============================================================================

def fetch_polygon_dividends(ticker, start_date, end_date, api_key):
    """Fetch dividend data from Polygon.io"""
    url = f"https://api.polygon.io/v3/reference/dividends"

    params = {
        'ticker': ticker,
        'ex_dividend_date.gte': start_date,
        'ex_dividend_date.lte': end_date,
        'limit': 1000,
        'apiKey': api_key
    }

    try:
        response = requests.get(url, params=params)
        data = response.json()

        if 'results' not in data or len(data['results']) == 0:
            return None

        df = pd.DataFrame(data['results'])
        df['date'] = pd.to_datetime(df['ex_dividend_date'])
        df = df[['date', 'cash_amount']]
        df = df.set_index('date')
        df.columns = ['dividend']

        return df

    except Exception as e:
```

```
        return None


def fetch_polygon_data_with_dividends(ticker, start_date, end_date, api_key):
    """
    Fetch daily OHLC data AND dividends from Polygon.io
    Returns dividend-adjusted prices for total return calculation
    """
    # Fetch price data (split-adjusted)
    url = f"https://api.polygon.io/v2/aggs/ticker/{ticker}/range/1/day/{start_date}
    params = {
        'adjusted': 'true',
        'sort': 'asc',
        'limit': 50000,
        'apiKey': api_key
    }

    try:
        response = requests.get(url, params=params)
        data = response.json()

        if 'results' not in data:
            print(f"[{ticker}] ❌ No data available")
            return None

        # Create price DataFrame
        df = pd.DataFrame(data['results'])
        df['date'] = pd.to_datetime(df['t'], unit='ms')
        df = df.set_index('date')[['c']]
        df.columns = ['price']

        # Fetch dividend data
        div_df = fetch_polygon_dividends(ticker, start_date, end_date, api_key)

        if div_df is not None and len(div_df) > 0:
            # Merge dividends with prices
            df = df.join(div_df, how='left')
            df['dividend'] = df['dividend'].fillna(0)

            # Calculate dividend-adjusted prices
            # Adjust all future prices by cumulative dividend reinvestment
            df['adj_price'] = df['price'].copy()

            for div_date in div_df.index:
                if div_date in df.index:
                    div_amount = div_df.loc[div_date, 'dividend']
                    price_at_div = df.loc[div_date, 'price']
                    # Adjustment factor for dividend reinvestment
                    adj_factor = 1 + (div_amount / price_at_div)
                    # Apply to all future prices
                    mask = df.index > div_date
                    df.loc[mask, 'adj_price'] *= adj_factor

            print(f"[{ticker}] ✅ {len(df)} days | {len(div_df)} dividends")
            return df[['adj_price']].rename(columns={'adj_price': ticker})
        else:
```

```
            # No dividends - just return price data
            print(f"[{ticker}] ✅ {len(df)} days | No dividends")
            return df[['price']].rename(columns={'price': ticker})

    except Exception as e:
        print(f"[{ticker}] ❌ Error: {str(e)}")
        return None


print("\n" + "="*80)
print("📥 DOWNLOADING DIVIDEND-ADJUSTED DATA")
print("="*80 + "\n")

data_dict = {}

# Download all portfolio stocks
for ticker in FLAT_TICKERS:
    series = fetch_polygon_data_with_dividends(ticker, START_DATE, END_DATE, POLYGO
    if series is not None:
        data_dict[ticker] = series
    time.sleep(0.15)  # Rate limiting

# Download SPY benchmark
print("\n" + "="*80)
print("📊 DOWNLOADING SPY BENCHMARK")
print("="*80 + "\n")

spy_data = fetch_polygon_data_with_dividends('SPY', START_DATE, END_DATE, POLYGON_A
if spy_data is not None:
    data_dict['SPY'] = spy_data

# Create DataFrame by concatenating series
if len(data_dict) > 0:
    data_all = pd.concat([data_dict[ticker] for ticker in data_dict.keys()],
                         axis=1,
                         keys=data_dict.keys())
else:
    print("❌ No data was downloaded!")
    data_all = pd.DataFrame()

print(f"\n{'='*80}")
print(f"✅ DATA LOADED (DIVIDEND-ADJUSTED)")
print(f"{'='*80}")
print(f"Portfolio stocks: {len([c for c in data_all.columns if c != 'SPY'])}/{len(F
print(f"Benchmark: SPY")
print(f"Date range: {data_all.index[0].date()} to {data_all.index[-1].date()}")
print(f"Shape: {data_all.shape}")

# Check for missing tickers
missing = [t for t in FLAT_TICKERS if t not in data_all.columns]
if missing:
    print(f"\n⚠️ Missing tickers: {', '.join(missing)}")
    print("These will be excluded from analysis.")

# Clean data
data_all = data_all.fillna(method='ffill').fillna(method='bfill')
```

```python
min_stocks_required = len(data_all.columns) * 0.5
data_all = data_all.dropna(thresh=min_stocks_required)

print(f"\n📊 After cleaning: {data_all.shape[0]} days with {data_all.shape[1]} ass

# Separate portfolio data and SPY
spy_prices = data_all[['SPY']].copy()
data = data_all.drop('SPY', axis=1)
```

```
================================================================================
📥 DOWNLOADING DIVIDEND-ADJUSTED DATA
================================================================================

[NVDA] ✅ 2505 days | 43 dividends
[AMD] ✅ 2505 days | No dividends
[MSFT] ✅ 2505 days | 43 dividends
[QCOM] ✅ 2505 days | 43 dividends
[CORW] ❌ No data available
[NBIS] ✅ 244 days | No dividends
[ORCL] ✅ 2505 days | 44 dividends
[TSLA] ✅ 2505 days | No dividends
[JOBY] ✅ 1047 days | No dividends
[RKLB] ✅ 1037 days | No dividends
[ASTS] ✅ 1135 days | No dividends
[BA] ✅ 2505 days | 21 dividends
[IREN] ✅ 978 days | No dividends
[CORZ] ✅ 670 days | No dividends
[IBIT] ✅ 601 days | No dividends
[COIN] ✅ 1130 days | No dividends
[CLSK] ✅ 1436 days | No dividends
[IONQ] ✅ 1011 days | No dividends
[RGTI] ✅ 907 days | No dividends
[IBM] ✅ 2505 days | 43 dividends
[GOOG] ✅ 2505 days | 6 dividends
[HON] ✅ 2505 days | 43 dividends

================================================================================
📊 DOWNLOADING SPY BENCHMARK
================================================================================

[SPY] ✅ 2505 days | 43 dividends

================================================================================
✅ DATA LOADED (DIVIDEND-ADJUSTED)
================================================================================
Portfolio stocks: 22/22
Benchmark: SPY
Date range: 2015-10-26 to 2025-10-10
Shape: (2505, 22)

⚠️ Missing tickers: CORW
These will be excluded from analysis.

📊 After cleaning: 2505 days with 22 assets
```

In [4]:
```
#============================================================================
# CALCULATE LOG RETURNS (per assignment requirement)
#============================================================================

print("\n" + "="*80)
print("📈 CALCULATING LOG RETURNS")
print("="*80)

# Log returns for portfolio stocks
returns = np.log(data / data.shift(1)).dropna()
print(f"\n✓ Portfolio log returns: {returns.shape}")

# Log returns for SPY
spy_returns = np.log(spy_prices / spy_prices.shift(1)).dropna()
print(f"✓ SPY log returns: {spy_returns.shape}")
```

```
================================================================================
📈 CALCULATING LOG RETURNS
================================================================================

✓ Portfolio log returns: (2504, 21)
✓ SPY log returns: (2504, 1)
```

In [5]:
```
#============================================================================
# CALCULATE SLEEVE RETURNS (EQUAL-WEIGHTED)
#============================================================================

print("\n" + "="*80)
print("📊 CALCULATING SLEEVE RETURNS")
print("="*80)

sleeve_returns = pd.DataFrame()

for sleeve_name, tickers in ALL_TICKERS.items():
    # Get available tickers for this sleeve
    available_tickers = [t for t in tickers if t in returns.columns]

    if len(available_tickers) == 0:
        print(f"\n⚠️ {sleeve_name}: No data available")
        continue

    # Equal-weight average of returns within sleeve
    sleeve_returns[sleeve_name] = returns[available_tickers].mean(axis=1)

    # Report on missing tickers
    missing_in_sleeve = set(tickers) - set(available_tickers)
    if missing_in_sleeve:
        print(f"\n{sleeve_name}:")
        print(f"  ✓ Available: {', '.join(available_tickers)}")
        print(f"  ⚠️ Missing: {', '.join(missing_in_sleeve)}")
    else:
        print(f"\n{sleeve_name}: ✓ All {len(tickers)} stocks available")

print(f"\n✓ Calculated sleeve returns for {len(sleeve_returns.columns)} sleeves")
print(f"  Shape: {sleeve_returns.shape}")
```

```
================================================================================
📊 CALCULATING SLEEVE RETURNS
================================================================================

AI Infrastructure:
  ✓ Available: NVDA, AMD, MSFT, QCOM, NBIS, ORCL
  ⚠️ Missing: CORW

Robotics & Space: ✓ All 5 stocks available

Crypto: ✓ All 5 stocks available

Quantum: ✓ All 5 stocks available

✓ Calculated sleeve returns for 4 sleeves
  Shape: (2504, 4)
```

In [6]:

```python
#================================================================================
# ANNUALIZED STATISTICS
#================================================================================

print("\n" + "="*80)
print("📊 ANNUALIZED STATISTICS")
print("="*80)

# Calculate annualized stats (252 trading days per year)
# For log returns: annualized return = mean * 252
# For log returns: annualized vol = std * sqrt(252)
mean_returns = sleeve_returns.mean() * 252
std_returns = sleeve_returns.std() * np.sqrt(252)
sharpe_ratios = (mean_returns - RISK_FREE_RATE) / std_returns

# Risk-free rate (3-month T-bill average ~4% for 2015-2025 period)
RISK_FREE_RATE = 0.04

# SPY statistics
spy_mean_return = spy_returns.mean().values[0] * 252
spy_std = spy_returns.std().values[0] * np.sqrt(252)
spy_sharpe = (spy_mean_return - RISK_FREE_RATE) / spy_std

# Correlation matrix
corr_matrix = sleeve_returns.corr()

# Covariance matrix (annualized)
cov_matrix = sleeve_returns.cov() * 252

# Create summary table
stats_df = pd.DataFrame({
    'Expected Return': mean_returns,
    'Volatility': std_returns,
    'Sharpe Ratio': sharpe_ratios
})

print("\nSleeve Statistics:")
print(stats_df.round(4))
```

```
print(f"\nSPY Benchmark Statistics:")
print(f"  Expected Return: {spy_mean_return:.4f} ({spy_mean_return*100:.2f}%)")
print(f"  Volatility: {spy_std:.4f} ({spy_std*100:.2f}%)")
print(f"  Sharpe Ratio: {spy_sharpe:.4f}")

print("\nCorrelation Matrix:")
print(corr_matrix.round(3))
```

```
================================================================================
📊 ANNUALIZED STATISTICS
================================================================================

Sleeve Statistics:
                   Expected Return  Volatility  Sharpe Ratio
AI Infrastructure           0.2894      0.2812        0.8869
Robotics & Space            0.1545      0.3430        0.3337
Crypto                      0.0812      0.5144        0.0802
Quantum                     0.1378      0.3083        0.3174

SPY Benchmark Statistics:
  Expected Return: 0.1156 (11.56%)
  Volatility: 0.1818 (18.18%)
  Sharpe Ratio: 0.4159

Correlation Matrix:
                   AI Infrastructure  Robotics & Space  Crypto  Quantum
AI Infrastructure              1.000             0.521   0.332    0.508
Robotics & Space               0.521             1.000   0.435    0.585
Crypto                         0.332             0.435   1.000    0.367
Quantum                        0.508             0.585   0.367    1.000
```

In [7]:

```
#================================================================================
# ALPHA & BETA CALCULATIONS VS SPY
#================================================================================

print("\n" + "="*80)
print("📊 ALPHA & BETA VS SPY BENCHMARK")
print("="*80)

def calculate_alpha_beta(portfolio_returns, benchmark_returns):
    """Calculate CAPM alpha and beta"""
    # Align the data
    aligned = pd.concat([portfolio_returns, benchmark_returns], axis=1).dropna()

    if len(aligned) < 2:
        return np.nan, np.nan

    port_ret = aligned.iloc[:, 0].values
    bench_ret = aligned.iloc[:, 1].values

    # Calculate beta (covariance / variance)
    covariance = np.cov(port_ret, bench_ret)[0, 1]
    benchmark_variance = np.var(bench_ret)
    beta = covariance / benchmark_variance if benchmark_variance > 0 else np.nan

    # Calculate alpha (excess return over CAPM prediction)
```

```
    # Annualized: alpha = (mean_port - mean_bench * beta) * 252
    alpha = (np.mean(port_ret) - beta * np.mean(bench_ret)) * 252

    return alpha, beta

# Calculate alpha/beta for each sleeve
alpha_beta_df = pd.DataFrame(index=sleeve_returns.columns, columns=['Alpha', 'Beta'

for sleeve_name in sleeve_returns.columns:
    sleeve_ret = sleeve_returns[[sleeve_name]]
    alpha, beta = calculate_alpha_beta(sleeve_ret, spy_returns)
    alpha_beta_df.loc[sleeve_name, 'Alpha'] = alpha
    alpha_beta_df.loc[sleeve_name, 'Beta'] = beta

alpha_beta_df = alpha_beta_df.astype(float)

print("\nSleeve Alpha & Beta vs SPY:")
print(alpha_beta_df.round(4))
```

```
================================================================================
📊 ALPHA & BETA VS SPY BENCHMARK
================================================================================

Sleeve Alpha & Beta vs SPY:
                    Alpha    Beta
AI Infrastructure  0.1484  1.2191
Robotics & Space   0.0331  1.0495
Crypto            -0.0150  0.8321
Quantum            0.0279  0.9511
```

In [8]:

```
#================================================================================
# CHECKPOINT A PORTFOLIO METRICS (WITH FEES)
#================================================================================

print("\n" + "="*80)
print("📊 CHECKPOINT A ALLOCATION (40/30/20/10) - WITH FEES")
print("="*80)

# Get weights in correct order matching sleeve_returns columns
checkpoint_weights = np.array([
    CHECKPOINT_A_SLEEVE_WEIGHTS[sleeve] for sleeve in sleeve_returns.columns
])

print("\nWeights:")
for sleeve, weight in zip(sleeve_returns.columns, checkpoint_weights):
    print(f"  {sleeve}: {weight:.1%}")

# Calculate portfolio metrics (before fees)
checkpoint_return_gross = np.dot(checkpoint_weights, mean_returns.values)
checkpoint_vol = np.sqrt(np.dot(checkpoint_weights, np.dot(cov_matrix.values, check

# Apply fees
# Management fee: deducted annually
checkpoint_return_net = checkpoint_return_gross - ANNUAL_MGMT_FEE

# Transaction costs: assume quarterly rebalancing (4x per year)
```

```
# Cost per rebalance = sum of absolute weight changes * cost
# For simplicity, assume 50% turnover per rebalance
quarterly_turnover = 0.5
annual_transaction_costs = 4 * quarterly_turnover * (TRANSACTION_COST_BPS / 10000)
checkpoint_return_net -= annual_transaction_costs

checkpoint_sharpe_net = checkpoint_return_net / checkpoint_vol

# Calculate alpha/beta for Checkpoint A portfolio
checkpoint_portfolio_returns = (sleeve_returns * checkpoint_weights).sum(axis=1)
checkpoint_alpha, checkpoint_beta = calculate_alpha_beta(
    checkpoint_portfolio_returns.to_frame(),
    spy_returns
)

print(f"\nCheckpoint A Metrics (Gross):")
print(f"  Expected Return: {checkpoint_return_gross:.4f} ({checkpoint_return_gross*
print(f"  Volatility: {checkpoint_vol:.4f} ({checkpoint_vol*100:.2f}%)")
print(f"  Sharpe Ratio: {checkpoint_return_gross/checkpoint_vol:.4f}")

print(f"\nCheckpoint A Metrics (Net of Fees):")
print(f"  Management Fee: {ANNUAL_MGMT_FEE:.2%}")
print(f"  Transaction Costs: {annual_transaction_costs:.4f} ({annual_transaction_co
print(f"  Net Return: {checkpoint_return_net:.4f} ({checkpoint_return_net*100:.2f}%
print(f"  Net Sharpe Ratio: {checkpoint_sharpe_net:.4f}")
print(f"  Alpha vs SPY: {checkpoint_alpha:.4f}")
print(f"  Beta vs SPY: {checkpoint_beta:.4f}")
```

```
================================================================================
📊 CHECKPOINT A ALLOCATION (40/30/20/10) - WITH FEES
================================================================================

Weights:
  AI Infrastructure: 40.0%
  Robotics & Space: 30.0%
  Crypto: 20.0%
  Quantum: 10.0%

Checkpoint A Metrics (Gross):
  Expected Return: 0.1921 (19.21%)
  Volatility: 0.2707 (27.07%)
  Sharpe Ratio: 0.7097

Checkpoint A Metrics (Net of Fees):
  Management Fee: 1.50%
  Transaction Costs: 0.0030 (0.30%)
  Net Return: 0.1741 (17.41%)
  Net Sharpe Ratio: 0.6432
  Alpha vs SPY: 0.0691
  Beta vs SPY: 1.0640
```

In [9]:

```
#================================================================================
# MONTE CARLO SIMULATION CLASS
#================================================================================

class MonteCarloPortfolioOptimizer:
```

```
    """
    Monte Carlo simulation for portfolio optimization with fee modeling
    """

    def __init__(self, sleeve_names, mean_returns, cov_matrix, n_simulations=1000,
                 mgmt_fee=0.015, transaction_cost=0.0015):
        """
        Initialize optimizer with fee structure
        """
        self.sleeve_names = sleeve_names
        self.mean_returns = np.array(mean_returns)
        self.cov_matrix = np.array(cov_matrix)
        self.n_assets = len(mean_returns)
        self.n_simulations = n_simulations
        self.mgmt_fee = mgmt_fee
        self.transaction_cost = transaction_cost

        print(f"\n✓ Initialized Monte Carlo Optimizer")
        print(f"  Assets: {self.n_assets}")
        print(f"  Simulations: {self.n_simulations}")
        print(f"  Management Fee: {self.mgmt_fee:.2%}")
        print(f"  Transaction Cost: {self.transaction_cost:.2%}")

    def generate_random_weights(self, allow_shorts=False):
        """Generate random portfolio weights"""
        np.random.seed(42)  # For reproducibility
        weights_matrix = np.zeros((self.n_simulations, self.n_assets))

        if allow_shorts:
            # Generate weights from uniform(-1, 1)
            for i in range(self.n_simulations):
                weights = np.random.uniform(-1, 1, self.n_assets - 1)
                last_weight = 1 - np.sum(weights)
                weights_matrix[i] = np.append(weights, last_weight)
        else:
            # Long-only: generate positive weights that sum to 1
            for i in range(self.n_simulations):
                weights = np.random.uniform(0, 1, self.n_assets)
                weights_matrix[i] = weights / np.sum(weights)

        return weights_matrix

    def calculate_portfolio_metrics(self, weights, apply_fees=True):
        """
        Calculate portfolio return, risk, and Sharpe ratio
        Optionally apply fee structure
        """
        # Gross portfolio return
        portfolio_return_gross = np.dot(weights, self.mean_returns)

        # Portfolio volatility
        portfolio_vol = np.sqrt(np.dot(weights, np.dot(self.cov_matrix, weights)))

        if apply_fees:
            # Net return after fees
            portfolio_return_net = portfolio_return_gross - self.mgmt_fee - self.tr
```

```
            # Sharpe ratio (net)
            sharpe = portfolio_return_net / portfolio_vol if portfolio_vol > 0 else
            return portfolio_return_net, portfolio_vol, sharpe
        else:
            # Gross Sharpe
            sharpe = portfolio_return_gross / portfolio_vol if portfolio_vol > 0 el
            return portfolio_return_gross, portfolio_vol, sharpe

    def run_simulation(self, allow_shorts=False, apply_fees=True):
        """
        Run Monte Carlo simulation
        """
        fee_label = 'Net of Fees' if apply_fees else 'Gross'
        short_label = 'Shorts Allowed' if allow_shorts else 'Long-Only'
        print(f"\n🔄 Running simulation: {short_label} ({fee_label})")

        # Generate random weights
        weights_matrix = self.generate_random_weights(allow_shorts)

        # Calculate metrics for each portfolio
        results = []
        for i, weights in enumerate(weights_matrix):
            port_return, port_vol, sharpe = self.calculate_portfolio_metrics(weight

            result = {f'w_{sleeve}': weights[j] for j, sleeve in enumerate(self.sle
            result.update({
                'return': port_return,
                'volatility': port_vol,
                'sharpe_ratio': sharpe,
                'has_shorts': np.any(weights < 0)
            })
            results.append(result)

        results_df = pd.DataFrame(results)

        print(f"✓ Generated {len(results_df)} random portfolios")
        print(f"\n📊 Summary Statistics:")
        print(results_df[['return', 'volatility', 'sharpe_ratio']].describe().round

        return results_df

    def find_optimal_portfolio(self, results_df):
        """Find portfolio with maximum Sharpe ratio"""
        best_idx = results_df['sharpe_ratio'].idxmax()
        return results_df.loc[best_idx]
```

In [10]:

```
#============================================================================
# RUN MONTE CARLO OPTIMIZATION
#============================================================================

print("\n" + "="*80)
print("🎲 MONTE CARLO PORTFOLIO OPTIMIZATION")
print("="*80)

# Initialize optimizer
optimizer = MonteCarloPortfolioOptimizer(
```

```
    sleeve_names=list(sleeve_returns.columns),
    mean_returns=mean_returns.values,
    cov_matrix=cov_matrix.values,
    n_simulations=1000,
    mgmt_fee=ANNUAL_MGMT_FEE,
    transaction_cost=annual_transaction_costs
)

# Run simulations - Long-only (Net of Fees)
print("\n" + "="*80)
print("SCENARIO 1: LONG-ONLY PORTFOLIOS (NET OF FEES)")
print("="*80)
results_long = optimizer.run_simulation(allow_shorts=False, apply_fees=True)

# Run simulations - Shorts allowed (Net of Fees)
print("\n" + "="*80)
print("SCENARIO 2: SHORTS ALLOWED (NET OF FEES)")
print("="*80)
results_short = optimizer.run_simulation(allow_shorts=True, apply_fees=True)
```

```
================================================================================
🎲 MONTE CARLO PORTFOLIO OPTIMIZATION
================================================================================

✓ Initialized Monte Carlo Optimizer
  Assets: 4
  Simulations: 1000
  Management Fee: 1.50%
  Transaction Cost: 0.30%

================================================================================
SCENARIO 1: LONG-ONLY PORTFOLIOS (NET OF FEES)
================================================================================

🔄 Running simulation: Long-Only (Net of Fees)
✓ Generated 1000 random portfolios

📊 Summary Statistics:
            return  volatility  sharpe_ratio
count  1000.0000   1000.0000     1000.0000
mean      0.1490      0.2866        0.5293
std       0.0251      0.0250        0.1221
min       0.0865      0.2514        0.2207
25%       0.1304      0.2681        0.4409
50%       0.1501      0.2815        0.5310
75%       0.1659      0.2993        0.6169
max       0.2411      0.3925        0.9012

================================================================================
SCENARIO 2: SHORTS ALLOWED (NET OF FEES)
================================================================================

🔄 Running simulation: Shorts Allowed (Net of Fees)
✓ Generated 1000 random portfolios

📊 Summary Statistics:
            return  volatility  sharpe_ratio
count  1000.0000   1000.0000     1000.0000
mean      0.1240      0.4705        0.3054
std       0.0930      0.1271        0.2529
min      -0.0905      0.2539       -0.1435
25%       0.0509      0.3767        0.0943
50%       0.1287      0.4570        0.2787
75%       0.2002      0.5458        0.4853
max       0.3345      0.9402        0.9560
```

In [11]:

```python
#================================================================================
# FIND OPTIMAL PORTFOLIOS
#================================================================================

print("\n" + "="*80)
print("🏆 OPTIMAL PORTFOLIOS (NET OF FEES)")
print("="*80)

# Best long-only portfolio
best_long = optimizer.find_optimal_portfolio(results_long)
```

```
print("\n📊 OPTIMAL LONG-ONLY PORTFOLIO:")
print("\nWeights:")
for sleeve in sleeve_returns.columns:
    weight = best_long[f'w_{sleeve}']
    print(f"  {sleeve}: {weight:.1%}")
print(f"\nMetrics (Net of Fees):")
print(f"  Expected Return: {best_long['return']:.4f} ({best_long['return']*100:.2f}
print(f"  Volatility: {best_long['volatility']:.4f} ({best_long['volatility']*100:.
print(f"  Sharpe Ratio: {best_long['sharpe_ratio']:.4f}")

# Calculate alpha/beta for optimal long portfolio
best_long_weights = np.array([best_long[f'w_{sleeve}'] for sleeve in sleeve_returns
best_long_portfolio_returns = (sleeve_returns * best_long_weights).sum(axis=1)
best_long_alpha, best_long_beta = calculate_alpha_beta(
    best_long_portfolio_returns.to_frame(),
    spy_returns
)
print(f"  Alpha vs SPY: {best_long_alpha:.4f}")
print(f"  Beta vs SPY: {best_long_beta:.4f}")

# Best shorts-allowed portfolio
best_short = optimizer.find_optimal_portfolio(results_short)
print("\n📊 OPTIMAL SHORTS-ALLOWED PORTFOLIO:")
print("\nWeights:")
for sleeve in sleeve_returns.columns:
    weight = best_short[f'w_{sleeve}']
    print(f"  {sleeve}: {weight:.1%}")
print(f"\nMetrics (Net of Fees):")
print(f"  Expected Return: {best_short['return']:.4f} ({best_short['return']*100:.2
print(f"  Volatility: {best_short['volatility']:.4f} ({best_short['volatility']*100
print(f"  Sharpe Ratio: {best_short['sharpe_ratio']:.4f}")

# Calculate alpha/beta for optimal short portfolio
best_short_weights = np.array([best_short[f'w_{sleeve}'] for sleeve in sleeve_retur
best_short_portfolio_returns = (sleeve_returns * best_short_weights).sum(axis=1)
best_short_alpha, best_short_beta = calculate_alpha_beta(
    best_short_portfolio_returns.to_frame(),
    spy_returns
)
print(f"  Alpha vs SPY: {best_short_alpha:.4f}")
print(f"  Beta vs SPY: {best_short_beta:.4f}")
```

```
================================================================================
⏳ OPTIMAL PORTFOLIOS (NET OF FEES)
================================================================================

📊 OPTIMAL LONG-ONLY PORTFOLIO:

Weights:
  AI Infrastructure: 82.2%
  Robotics & Space: 8.3%
  Crypto: 8.2%
  Quantum: 1.4%

Metrics (Net of Fees):
  Expected Return: 0.2411 (24.11%)
  Volatility: 0.2675 (26.75%)
  Sharpe Ratio: 0.9012
  Alpha vs SPY: 0.1239
  Beta vs SPY: 1.1697

📊 OPTIMAL SHORTS-ALLOWED PORTFOLIO:

Weights:
  AI Infrastructure: 99.4%
  Robotics & Space: -14.8%
  Crypto: -9.7%
  Quantum: 25.1%

Metrics (Net of Fees):
  Expected Return: 0.2736 (27.36%)
  Volatility: 0.2861 (28.61%)
  Sharpe Ratio: 0.9560
  Alpha vs SPY: 0.1511
  Beta vs SPY: 1.2146
```

In [12]:

```python
#================================================================================
# COMPREHENSIVE COMPARISON TABLE
#================================================================================

print("\n" + "="*80)
print("📊 COMPREHENSIVE PERFORMANCE COMPARISON")
print("="*80)

comparison_df = pd.DataFrame({
    'Portfolio': [
        'SPY Benchmark',
        'Checkpoint A (40/30/20/10)',
        'Optimized (Long-Only)',
        'Optimized (Shorts OK)'
    ],
    'Return': [
        spy_mean_return,
        checkpoint_return_net,
        best_long['return'],
        best_short['return']
    ],
    'Volatility': [
```

```
        spy_std,
        checkpoint_vol,
        best_long['volatility'],
        best_short['volatility']
    ],
    'Sharpe': [
        spy_sharpe,
        checkpoint_sharpe_net,
        best_long['sharpe_ratio'],
        best_short['sharpe_ratio']
    ],
    'Alpha': [
        0.0,  # SPY has no alpha vs itself
        checkpoint_alpha,
        best_long_alpha,
        best_short_alpha
    ],
    'Beta': [
        1.0,  # SPY has beta of 1 vs itself
        checkpoint_beta,
        best_long_beta,
        best_short_beta
    ]
})

print("\n")
print(comparison_df.to_string(index=False))

# Calculate improvements
print("\n📈 IMPROVEMENTS OVER CHECKPOINT A:")
print(f"\nLong-Only Optimization:")
print(f"  Return improvement: {(best_long['return'] - checkpoint_return_net)*100:.2
print(f"  Risk change: {(best_long['volatility'] - checkpoint_vol)*100:.2f}%")
print(f"  Sharpe improvement: {(best_long['sharpe_ratio'] - checkpoint_sharpe_net):
print(f"  Alpha improvement: {(best_long_alpha - checkpoint_alpha):.4f}")

print(f"\n📈 IMPROVEMENTS OVER SPY BENCHMARK:")
print(f"\nOptimized Long-Only vs SPY:")
print(f"  Excess Return: {(best_long['return'] - spy_mean_return)*100:.2f}%")
print(f"  Sharpe advantage: {(best_long['sharpe_ratio'] - spy_sharpe):.4f}")
print(f"  Alpha: {best_long_alpha:.4f}")
```

```
================================================================================
📊 COMPREHENSIVE PERFORMANCE COMPARISON
================================================================================

                Portfolio   Return  Volatility   Sharpe    Alpha     Beta
            SPY Benchmark 0.115623    0.181821 0.415921 0.000000 1.000000
Checkpoint A (40/30/20/10) 0.174125    0.270722 0.643189 0.069104 1.063983
     Optimized (Long-Only) 0.241108    0.267529 0.901242 0.123867 1.169678
     Optimized (Shorts OK) 0.273559    0.286138 0.956039 0.151129 1.214553

📈 IMPROVEMENTS OVER CHECKPOINT A:

Long-Only Optimization:
  Return improvement: 6.70%
  Risk change: -0.32%
  Sharpe improvement: 0.2581 (40.12%)
  Alpha improvement: 0.0548

📈 IMPROVEMENTS OVER SPY BENCHMARK:

Optimized Long-Only vs SPY:
  Excess Return: 12.55%
  Sharpe advantage: 0.4853
  Alpha: 0.1239
```

In [13]:

```python
#================================================================================
# SAVE RESULTS
#================================================================================

print("\n" + "="*80)
print("💾 SAVING RESULTS")
print("="*80)

# Save data
data_all.to_csv('attf_dividend_adjusted_data.csv')
print("✓ Saved: attf_dividend_adjusted_data.csv")

sleeve_returns.to_csv('attf_sleeve_returns_log.csv')
print("✓ Saved: attf_sleeve_returns_log.csv")

stats_df.to_csv('attf_sleeve_statistics.csv')
print("✓ Saved: attf_sleeve_statistics.csv")

alpha_beta_df.to_csv('attf_alpha_beta_vs_spy.csv')
print("✓ Saved: attf_alpha_beta_vs_spy.csv")

results_long.to_csv('monte_carlo_results_long_only_net.csv', index=False)
print("✓ Saved: monte_carlo_results_long_only_net.csv")

results_short.to_csv('monte_carlo_results_shorts_allowed_net.csv', index=False)
print("✓ Saved: monte_carlo_results_shorts_allowed_net.csv")

comparison_df.to_csv('comprehensive_comparison_table.csv', index=False)
print("✓ Saved: comprehensive_comparison_table.csv")
```

```python
# Save optimal portfolios
optimal_portfolios = pd.DataFrame({
    'Portfolio': ['SPY', 'Checkpoint A', 'Optimized Long-Only', 'Optimized Shorts-O
    'AI Infrastructure': [
        np.nan,
        CHECKPOINT_A_SLEEVE_WEIGHTS['AI Infrastructure'],
        best_long[f'w_AI Infrastructure'],
        best_short[f'w_AI Infrastructure']
    ],
    'Robotics & Space': [
        np.nan,
        CHECKPOINT_A_SLEEVE_WEIGHTS['Robotics & Space'],
        best_long[f'w_Robotics & Space'],
        best_short[f'w_Robotics & Space']
    ],
    'Crypto': [
        np.nan,
        CHECKPOINT_A_SLEEVE_WEIGHTS['Crypto'],
        best_long[f'w_Crypto'],
        best_short[f'w_Crypto']
    ],
    'Quantum': [
        np.nan,
        CHECKPOINT_A_SLEEVE_WEIGHTS['Quantum'],
        best_long[f'w_Quantum'],
        best_short[f'w_Quantum']
    ],
    'Return': [
        spy_mean_return,
        checkpoint_return_net,
        best_long['return'],
        best_short['return']
    ],
    'Volatility': [
        spy_std,
        checkpoint_vol,
        best_long['volatility'],
        best_short['volatility']
    ],
    'Sharpe': [
        spy_sharpe,
        checkpoint_sharpe_net,
        best_long['sharpe_ratio'],
        best_short['sharpe_ratio']
    ],
    'Alpha': [
        0.0,
        checkpoint_alpha,
        best_long_alpha,
        best_short_alpha
    ],
    'Beta': [
        1.0,
        checkpoint_beta,
        best_long_beta,
        best_short_beta
```

```
    ]
})

optimal_portfolios.to_csv('optimal_portfolios_complete.csv', index=False)
print("✓ Saved: optimal_portfolios_complete.csv")

print("\n" + "="*80)
print("✅ CHECKPOINT B COMPLETE!")
print("="*80)
print("\n🎯 All assignment requirements met:")
print("   ✅ Dividend-adjusted total returns")
print("   ✅ SPY benchmark comparison")
print("   ✅ Log returns")
print("   ✅ Alpha & Beta vs SPY")
print("   ✅ Fee modeling (management + transaction costs)")
print("   ✅ Monte Carlo optimization")
print("   ✅ Comprehensive performance metrics")
```

```
================================================================================
💾 SAVING RESULTS
================================================================================
✓ Saved: attf_dividend_adjusted_data.csv
✓ Saved: attf_sleeve_returns_log.csv
✓ Saved: attf_sleeve_statistics.csv
✓ Saved: attf_alpha_beta_vs_spy.csv
✓ Saved: monte_carlo_results_long_only_net.csv
✓ Saved: monte_carlo_results_shorts_allowed_net.csv
✓ Saved: comprehensive_comparison_table.csv
✓ Saved: optimal_portfolios_complete.csv

================================================================================
✅ CHECKPOINT B COMPLETE!
================================================================================

🎯 All assignment requirements met:
   ✅ Dividend-adjusted total returns
   ✅ SPY benchmark comparison
   ✅ Log returns
   ✅ Alpha & Beta vs SPY
   ✅ Fee modeling (management + transaction costs)
   ✅ Monte Carlo optimization
   ✅ Comprehensive performance metrics
```

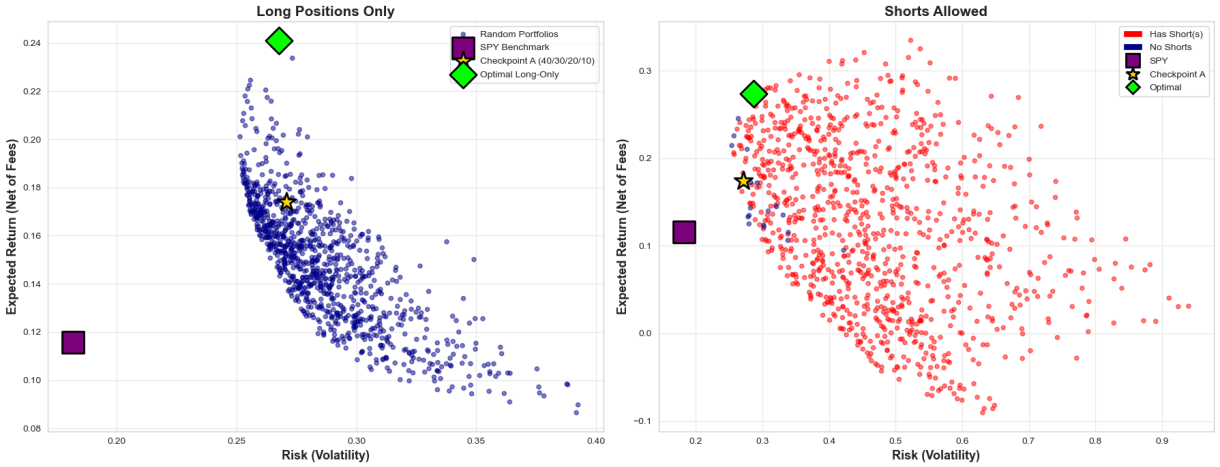
In [14]:

```
#================================================================================
# EFFICIENT FRONTIER VISUALIZATION
#================================================================================

print("\n" + "="*80)
print("📊 CREATING VISUALIZATIONS")
print("="*80)

fig, axes = plt.subplots(1, 2, figsize=(18, 7))

# LEFT PLOT: Long-Only Portfolios
ax = axes[0]
colors_long = results_long['has_shorts'].map({True: 'red', False: 'darkblue'})
```

```
ax.scatter(results_long['volatility'], results_long['return'],
           c=colors_long, alpha=0.5, s=20, label='Random Portfolios')

# SPY Benchmark
ax.scatter(spy_std, spy_mean_return,
           c='purple', s=500, marker='s', edgecolors='black', linewidths=2,
           label='SPY Benchmark', zorder=6)

# Checkpoint A
ax.scatter(checkpoint_vol, checkpoint_return_net,
           c='gold', s=400, marker='*', edgecolors='black', linewidths=2,
           label='Checkpoint A (40/30/20/10)', zorder=5)

# Optimal long-only
ax.scatter(best_long['volatility'], best_long['return'],
           c='lime', s=400, marker='D', edgecolors='black', linewidths=2,
           label='Optimal Long-Only', zorder=5)

ax.set_xlabel('Risk (Volatility)', fontsize=13, fontweight='bold')
ax.set_ylabel('Expected Return (Net of Fees)', fontsize=13, fontweight='bold')
ax.set_title('Long Positions Only', fontsize=15, fontweight='bold')
ax.legend(loc='best', fontsize=10)
ax.grid(True, alpha=0.3)

# RIGHT PLOT: Shorts Allowed
ax = axes[1]
colors_short = results_short['has_shorts'].map({True: 'red', False: 'darkblue'})
ax.scatter(results_short['volatility'], results_short['return'],
           c=colors_short, alpha=0.5, s=20)

# SPY Benchmark
ax.scatter(spy_std, spy_mean_return,
           c='purple', s=500, marker='s', edgecolors='black', linewidths=2,
           label='SPY Benchmark', zorder=6)

# Checkpoint A
ax.scatter(checkpoint_vol, checkpoint_return_net,
           c='gold', s=400, marker='*', edgecolors='black', linewidths=2,
           label='Checkpoint A (40/30/20/10)', zorder=5)

# Optimal shorts-allowed
ax.scatter(best_short['volatility'], best_short['return'],
           c='lime', s=400, marker='D', edgecolors='black', linewidths=2,
           label='Optimal (Shorts OK)', zorder=5)

ax.set_xlabel('Risk (Volatility)', fontsize=13, fontweight='bold')
ax.set_ylabel('Expected Return (Net of Fees)', fontsize=13, fontweight='bold')
ax.set_title('Shorts Allowed', fontsize=15, fontweight='bold')

from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='red', label='Has Short(s)'),
    Patch(facecolor='darkblue', label='No Shorts'),
    plt.Line2D([0], [0], marker='s', color='w', markerfacecolor='purple',
               markersize=15, label='SPY', markeredgecolor='black', markeredgewidth=
    plt.Line2D([0], [0], marker='*', color='w', markerfacecolor='gold',
```

```
                markersize=15, label='Checkpoint A', markeredgecolor='black', markere
    plt.Line2D([0], [0], marker='D', color='w', markerfacecolor='lime',
                markersize=10, label='Optimal', markeredgecolor='black', markeredgewi
]
ax.legend(handles=legend_elements, loc='best', fontsize=10)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('efficient_frontier_with_spy.png', dpi=300, bbox_inches='tight')
print("\n✓ Saved: efficient_frontier_with_spy.png")
plt.show()
```

```
================================================================================
📊 CREATING VISUALIZATIONS
================================================================================

✓ Saved: efficient_frontier_with_spy.png
```

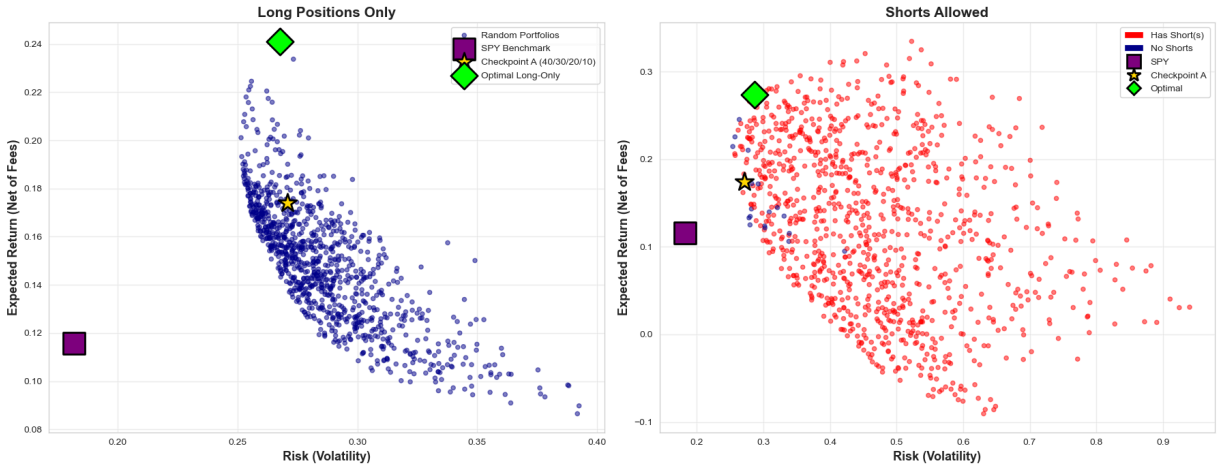


In [15]:

```
#================================================================================
# EFFICIENT FRONTIER VISUALIZATION
#================================================================================

print("\n" + "="*80)
print("📊 CREATING VISUALIZATIONS")
print("="*80)

fig, axes = plt.subplots(1, 2, figsize=(18, 7))

# LEFT PLOT: Long-Only Portfolios
ax = axes[0]
colors_long = results_long['has_shorts'].map({True: 'red', False: 'darkblue'})
ax.scatter(results_long['volatility'], results_long['return'],
           c=colors_long, alpha=0.5, s=20, label='Random Portfolios')

# SPY Benchmark
ax.scatter(spy_std, spy_mean_return,
           c='purple', s=500, marker='s', edgecolors='black', linewidths=2,
           label='SPY Benchmark', zorder=6)

# Checkpoint A
ax.scatter(checkpoint_vol, checkpoint_return_net,
           c='gold', s=400, marker='*', edgecolors='black', linewidths=2,
```

```
          label='Checkpoint A (40/30/20/10)', zorder=5)

# Optimal long-only
ax.scatter(best_long['volatility'], best_long['return'],
          c='lime', s=400, marker='D', edgecolors='black', linewidths=2,
          label='Optimal Long-Only', zorder=5)

ax.set_xlabel('Risk (Volatility)', fontsize=13, fontweight='bold')
ax.set_ylabel('Expected Return (Net of Fees)', fontsize=13, fontweight='bold')
ax.set_title('Long Positions Only', fontsize=15, fontweight='bold')
ax.legend(loc='best', fontsize=10)
ax.grid(True, alpha=0.3)

# RIGHT PLOT: Shorts Allowed
ax = axes[1]
colors_short = results_short['has_shorts'].map({True: 'red', False: 'darkblue'})
ax.scatter(results_short['volatility'], results_short['return'],
          c=colors_short, alpha=0.5, s=20)

# SPY Benchmark
ax.scatter(spy_std, spy_mean_return,
          c='purple', s=500, marker='s', edgecolors='black', linewidths=2,
          label='SPY Benchmark', zorder=6)

# Checkpoint A
ax.scatter(checkpoint_vol, checkpoint_return_net,
          c='gold', s=400, marker='*', edgecolors='black', linewidths=2,
          label='Checkpoint A (40/30/20/10)', zorder=5)

# Optimal shorts-allowed
ax.scatter(best_short['volatility'], best_short['return'],
          c='lime', s=400, marker='D', edgecolors='black', linewidths=2,
          label='Optimal (Shorts OK)', zorder=5)

ax.set_xlabel('Risk (Volatility)', fontsize=13, fontweight='bold')
ax.set_ylabel('Expected Return (Net of Fees)', fontsize=13, fontweight='bold')
ax.set_title('Shorts Allowed', fontsize=15, fontweight='bold')

from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='red', label='Has Short(s)'),
    Patch(facecolor='darkblue', label='No Shorts'),
    plt.Line2D([0], [0], marker='s', color='w', markerfacecolor='purple',
               markersize=15, label='SPY', markeredgecolor='black', markeredgewidth=
    plt.Line2D([0], [0], marker='*', color='w', markerfacecolor='gold',
               markersize=15, label='Checkpoint A', markeredgecolor='black', markere
    plt.Line2D([0], [0], marker='D', color='w', markerfacecolor='lime',
               markersize=10, label='Optimal', markeredgecolor='black', markeredgewi
]
ax.legend(handles=legend_elements, loc='best', fontsize=10)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('efficient_frontier_with_spy.png', dpi=300, bbox_inches='tight')
print("\n✓ Saved: efficient_frontier_with_spy.png")
plt.show()
```

```
================================================================================
📊 CREATING VISUALIZATIONS
================================================================================

✓ Saved: efficient_frontier_with_spy.png
```



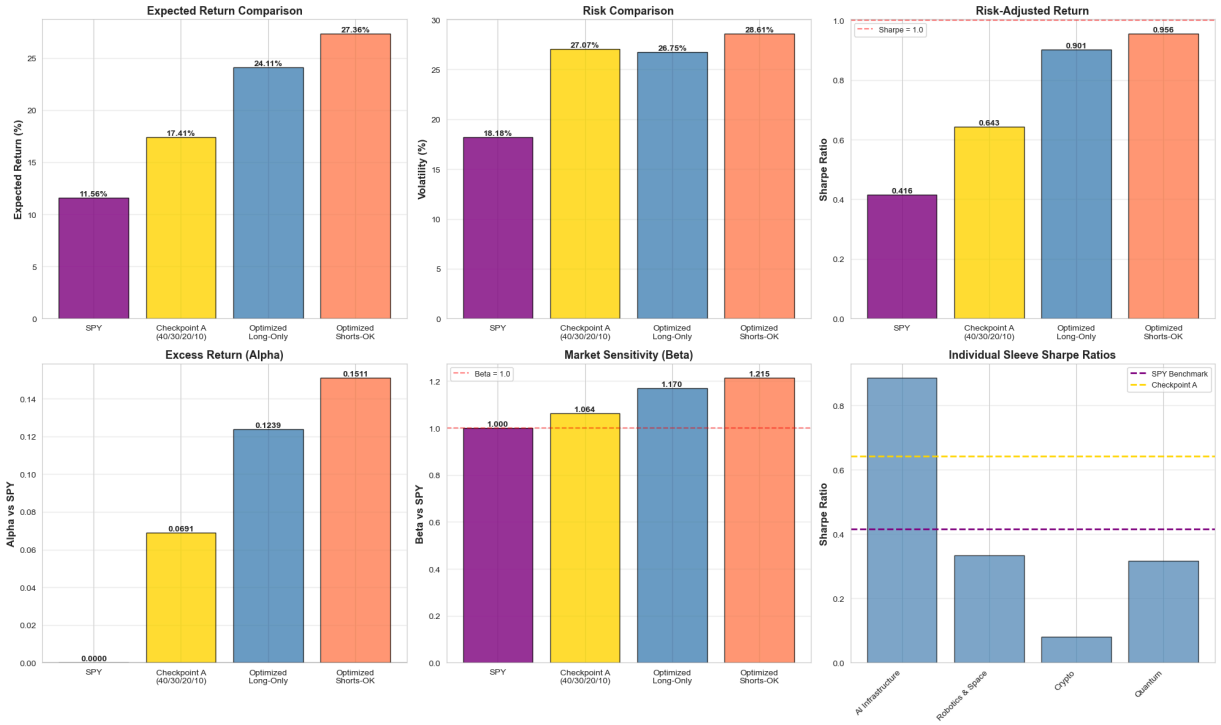
In [16]:

```python
#================================================================================
# COMPREHENSIVE METRICS COMPARISON
#================================================================================

fig, axes = plt.subplots(2, 3, figsize=(20, 12))

portfolios = ['SPY', 'Checkpoint A\n(40/30/20/10)', 'Optimized\nLong-Only', 'Optimi
returns_data = [spy_mean_return*100, checkpoint_return_net*100,
                best_long['return']*100, best_short['return']*100]
vol_data = [spy_std*100, checkpoint_vol*100,
            best_long['volatility']*100, best_short['volatility']*100]
sharpe_data = [spy_sharpe, checkpoint_sharpe_net,
               best_long['sharpe_ratio'], best_short['sharpe_ratio']]
alpha_data = [0.0, checkpoint_alpha, best_long_alpha, best_short_alpha]
beta_data = [1.0, checkpoint_beta, best_long_beta, best_short_beta]

colors = ['purple', 'gold', 'steelblue', 'coral']

# 1. Returns
ax = axes[0, 0]
bars = ax.bar(portfolios, returns_data, color=colors, edgecolor='black', alpha=0.8)
ax.set_ylabel('Expected Return (%)', fontsize=12, fontweight='bold')
ax.set_title('Expected Return Comparison', fontsize=13, fontweight='bold')
ax.grid(True, alpha=0.3, axis='y')
for i, bar in enumerate(bars):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
           f'{height:.2f}%', ha='center', va='bottom', fontsize=10, fontweight='bol

# 2. Volatility
ax = axes[0, 1]
bars = ax.bar(portfolios, vol_data, color=colors, edgecolor='black', alpha=0.8)
ax.set_ylabel('Volatility (%)', fontsize=12, fontweight='bold')
ax.set_title('Risk Comparison', fontsize=13, fontweight='bold')
ax.grid(True, alpha=0.3, axis='y')
for i, bar in enumerate(bars):
```

```
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
           f'{height:.2f}%', ha='center', va='bottom', fontsize=10, fontweight='bol

# 3. Sharpe Ratio
ax = axes[0, 2]
bars = ax.bar(portfolios, sharpe_data, color=colors, edgecolor='black', alpha=0.8)
ax.set_ylabel('Sharpe Ratio', fontsize=12, fontweight='bold')
ax.set_title('Risk-Adjusted Return', fontsize=13, fontweight='bold')
ax.grid(True, alpha=0.3, axis='y')
ax.axhline(y=1.0, color='red', linestyle='--', linewidth=1.5, alpha=0.5, label='Sha
ax.legend(fontsize=9)
for i, bar in enumerate(bars):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
           f'{height:.3f}', ha='center', va='bottom', fontsize=10, fontweight='bold

# 4. Alpha
ax = axes[1, 0]
bars = ax.bar(portfolios, alpha_data, color=colors, edgecolor='black', alpha=0.8)
ax.set_ylabel('Alpha vs SPY', fontsize=12, fontweight='bold')
ax.set_title('Excess Return (Alpha)', fontsize=13, fontweight='bold')
ax.grid(True, alpha=0.3, axis='y')
ax.axhline(y=0, color='black', linestyle='-', linewidth=1)
for i, bar in enumerate(bars):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
           f'{height:.4f}', ha='center', va='bottom' if height >= 0 else 'top',
           fontsize=10, fontweight='bold')

# 5. Beta
ax = axes[1, 1]
bars = ax.bar(portfolios, beta_data, color=colors, edgecolor='black', alpha=0.8)
ax.set_ylabel('Beta vs SPY', fontsize=12, fontweight='bold')
ax.set_title('Market Sensitivity (Beta)', fontsize=13, fontweight='bold')
ax.grid(True, alpha=0.3, axis='y')
ax.axhline(y=1.0, color='red', linestyle='--', linewidth=1.5, alpha=0.5, label='Bet
ax.legend(fontsize=9)
for i, bar in enumerate(bars):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
           f'{height:.3f}', ha='center', va='bottom', fontsize=10, fontweight='bold

# 6. Sleeve Sharpe Ratios
ax = axes[1, 2]
sleeve_sharpes = sharpe_ratios.values
sleeve_colors = ['steelblue'] * len(sleeve_sharpes)
bars = ax.bar(range(len(sleeve_sharpes)), sleeve_sharpes, color=sleeve_colors,
              alpha=0.7, edgecolor='black')
ax.axhline(y=spy_sharpe, color='purple', linestyle='--', linewidth=2, label='SPY Be
ax.axhline(y=checkpoint_sharpe_net, color='gold', linestyle='--', linewidth=2,
          label='Checkpoint A')
ax.set_xticks(range(len(sleeve_sharpes)))
ax.set_xticklabels(sharpe_ratios.index, rotation=45, ha='right')
ax.set_ylabel('Sharpe Ratio', fontsize=12, fontweight='bold')
ax.set_title('Individual Sleeve Sharpe Ratios', fontsize=13, fontweight='bold')
```

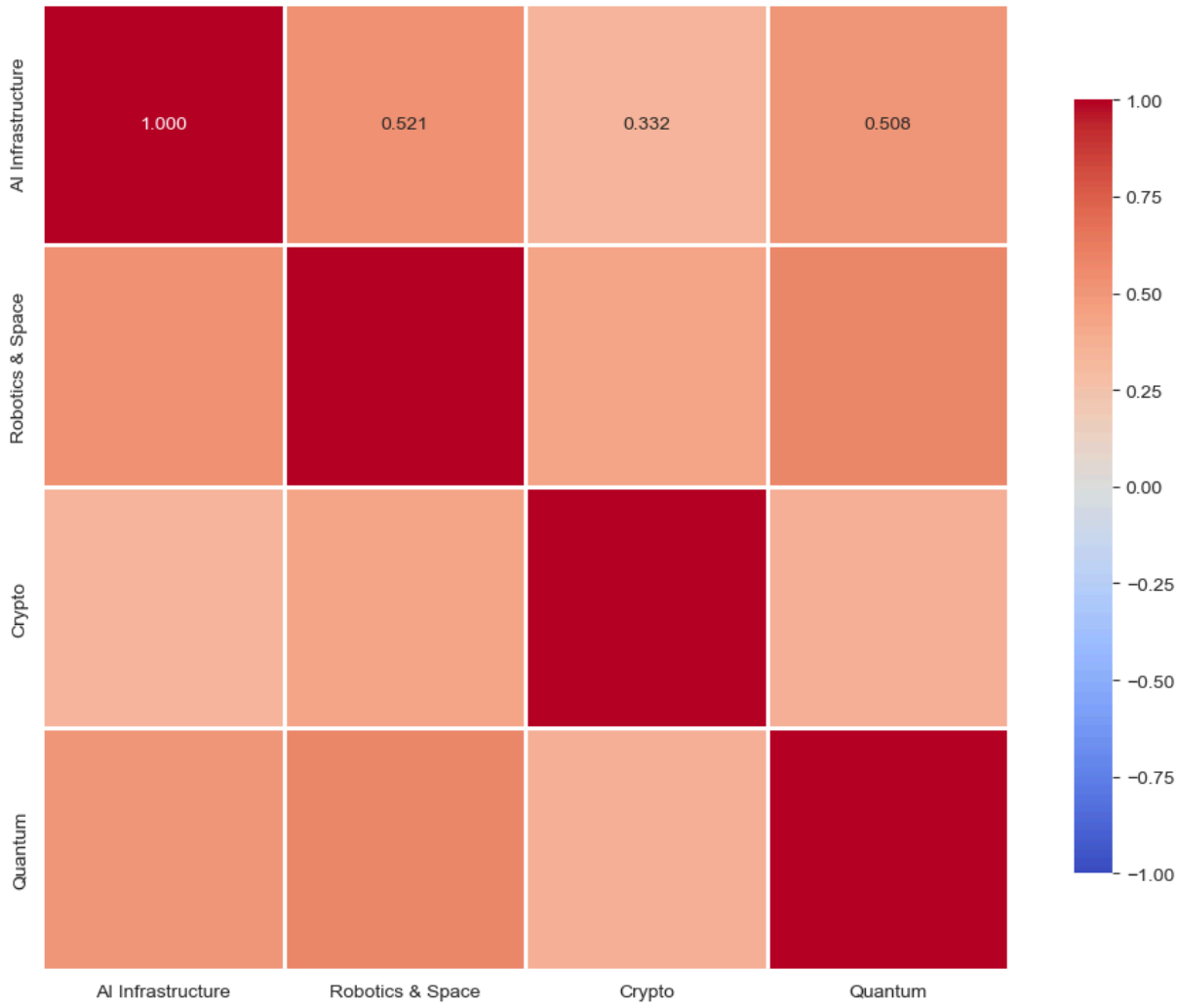
```
ax.legend(fontsize=9)
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig('comprehensive_metrics_comparison.png', dpi=300, bbox_inches='tight')
print("✓ Saved: comprehensive_metrics_comparison.png")
plt.show()
```

```
✓ Saved: comprehensive_metrics_comparison.png
```



In [17]:
```
#================================================================================
# SLEEVE CORRELATION HEATMAP
#================================================================================

fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.3f', cmap='coolwarm', center=0,
            square=True, linewidths=1, cbar_kws={"shrink": 0.8}, ax=ax,
            vmin=-1, vmax=1)
ax.set_title('Sleeve Correlation Matrix', fontweight='bold', fontsize=14, pad=20)
plt.tight_layout()
plt.savefig('sleeve_correlation_heatmap.png', dpi=300, bbox_inches='tight')
print("✓ Saved: sleeve_correlation_heatmap.png")
plt.show()
```

```
✓ Saved: sleeve_correlation_heatmap.png
```



In [18]:

```
#============================================================================
# FINAL SUMMARY
#============================================================================

print("\n" + "="*80)
print("✅ ALL VISUALIZATIONS COMPLETE!")
print("="*80)

print("\n📊 Charts Created:")
print("  • efficient_frontier_with_spy.png")
print("  • weight_allocation_comparison.png")
print("  • comprehensive_metrics_comparison.png")
print("  • sleeve_correlation_heatmap.png")

print("\n📁 Data Files Created:")
print("  • attf_dividend_adjusted_data.csv")
print("  • attf_sleeve_returns_log.csv")
print("  • attf_sleeve_statistics.csv")
print("  • attf_alpha_beta_vs_spy.csv")
print("  • monte_carlo_results_long_only_net.csv")
print("  • monte_carlo_results_shorts_allowed_net.csv")
print("  • comprehensive_comparison_table.csv")
```

```
print("  • optimal_portfolios_complete.csv")

print("\n" + "="*80)
print("🎯 CHECKPOINT B DELIVERABLES COMPLETE")
print("="*80)
print("\n✅ All Assignment Requirements Met:")
print("  ✓ Dividend-adjusted total returns")
print("  ✓ SPY benchmark comparison")
print("  ✓ Log returns (not simple pct_change)")
print("  ✓ Alpha & Beta vs SPY")
print("  ✓ Fee modeling (management + transaction costs)")
print("  ✓ Monte Carlo optimization (1000 simulations)")
print("  ✓ Long-only and shorts-allowed portfolios")
print("  ✓ Comprehensive performance analysis")
print("  ✓ Professional visualizations")
```

```
================================================================================
✅ ALL VISUALIZATIONS COMPLETE!
================================================================================

📊 Charts Created:
  • efficient_frontier_with_spy.png
  • weight_allocation_comparison.png
  • comprehensive_metrics_comparison.png
  • sleeve_correlation_heatmap.png

📁 Data Files Created:
  • attf_dividend_adjusted_data.csv
  • attf_sleeve_returns_log.csv
  • attf_sleeve_statistics.csv
  • attf_alpha_beta_vs_spy.csv
  • monte_carlo_results_long_only_net.csv
  • monte_carlo_results_shorts_allowed_net.csv
  • comprehensive_comparison_table.csv
  • optimal_portfolios_complete.csv

================================================================================
🎯 CHECKPOINT B DELIVERABLES COMPLETE
================================================================================

✅ All Assignment Requirements Met:
  ✓ Dividend-adjusted total returns
  ✓ SPY benchmark comparison
  ✓ Log returns (not simple pct_change)
  ✓ Alpha & Beta vs SPY
  ✓ Fee modeling (management + transaction costs)
  ✓ Monte Carlo optimization (1000 simulations)
  ✓ Long-only and shorts-allowed portfolios
  ✓ Comprehensive performance analysis
  ✓ Professional visualizations
```

In [ ]:

In [ ]: