Midterm: IT Application in Banking and Finance

Student name: Tran Trung Chien

Student code: 2112343020

Class code: ML71

Submitted documents: 3 files

- 1 pdf file
- 1 notebook file (.ipynb)
- 1 mgarch_model file (.py file contains BEKK, DCC, ADCC, and cDCC model built manually)

Question 1

Get crypto data (ETH, LINK, NEAR) from Binance API

Import necessary libraries

In [1]:
```
from binance.client import Client
from datetime import datetime, timedelta
import pandas as pd
```

Stream the data

In [7]:
```
def live_data(symbol, interval, start_time):
    # API key and Secret key for accessing Binance API
    API_KEY = 'your_api_key' # for security i can not public my paid API key as wel
    SECRET_KEY = 'your_secret_key' # for security i can not public my paid API key

    # instantiate the client to interact with the Binance API
    client = Client(API_KEY, SECRET_KEY)

    end_time = 'now UTC' # lastest data at current timestamp
    start_time = start_time

    df = pd.DataFrame(client.futures_historical_klines(symbol, interval, start_time
    df = df.iloc[:, 0:6]
    df.columns = ['date', 'open', 'high', 'low', 'close', 'volume']
    df['date'] = pd.to_datetime(df['date'], unit='ms') + timedelta(hours = 8) # con
    return df
```

In [8]:
```
# stream 5 years ETH data
eth = live_data('ETHUSDT', Client.KLINE_INTERVAL_1DAY, start_time=(datetime.utcnow(
eth
```

Out[8]:

	date	open	high	low	close	volume
0	2019-11-27 08:00:00	146	155.66	125.03	152.52	115911.840
1	2019-11-28 08:00:00	154.29	156.52	146.41	150.48	116824.070
2	2019-11-29 08:00:00	150.56	157.40	150.55	154.41	167906.104
3	2019-11-30 08:00:00	154.40	155.15	149.66	151.38	370491.615
4	2019-12-01 08:00:00	151.38	152.50	145.50	150.65	394494.119
...	...	...	...	...	...	...
1637	2024-05-21 08:00:00	3664.39	3849.26	3628.00	3792.32	5629142.300
1638	2024-05-22 08:00:00	3792.33	3816.84	3655.00	3740.25	3626843.371
1639	2024-05-23 08:00:00	3740.25	3952.89	3524.55	3784.80	7635174.311
1640	2024-05-24 08:00:00	3784.79	3832.00	3627.00	3729.61	3215814.229
1641	2024-05-25 08:00:00	3729.61	3782.82	3710.14	3749.99	904637.584

1642 rows × 6 columns

In [9]:
```
# stream 5 years NEAR data
near = live_data('NEARUSDT', Client.KLINE_INTERVAL_1DAY, start_time=(datetime.utcno
near
```

Out[9]:

	date	open	high	low	close	volume
0	2020-10-15 08:00:00	1.0625	1.2231	1.0625	1.1220	16485482
1	2020-10-16 08:00:00	1.1210	1.1585	0.8112	0.8156	30730886
2	2020-10-17 08:00:00	0.8175	0.8660	0.7195	0.8079	31125587
3	2020-10-18 08:00:00	0.8079	0.8740	0.7988	0.8702	14539580
4	2020-10-19 08:00:00	0.8702	0.8718	0.7704	0.8014	13018718
...	...	...	...	...	...	...
1314	2024-05-21 08:00:00	8.2910	8.3380	7.7610	7.8170	41115610
1315	2024-05-22 08:00:00	7.8180	8.2710	7.7000	7.9890	44737713
1316	2024-05-23 08:00:00	7.9880	8.1850	7.2700	7.6940	60676678
1317	2024-05-24 08:00:00	7.6940	8.0760	7.5900	7.9130	31022612
1318	2024-05-25 08:00:00	7.9130	8.1550	7.8630	8.0290	16506673

1319 rows × 6 columns

In [10]:

```
# stream 5 years LINK data
link = live_data('LINKUSDT', Client.KLINE_INTERVAL_1DAY, start_time=(datetime.utcno
link
```

Out[10]:

	date	open	high	low	close	volume
0	2020-01-17 08:00:00	2.669	2.896	2.592	2.696	8174258.39
1	2020-01-18 08:00:00	2.696	2.799	2.560	2.774	7920982.41
2	2020-01-19 08:00:00	2.777	2.854	2.519	2.623	9414962.40
3	2020-01-20 08:00:00	2.623	2.745	2.544	2.695	7244721.52
4	2020-01-21 08:00:00	2.694	2.745	2.561	2.673	5857502.02
...	...	...	...	...	...	...
1586	2024-05-21 08:00:00	17.259	17.443	16.510	16.748	21148745.79
1587	2024-05-22 08:00:00	16.749	16.939	16.161	16.362	18873214.06
1588	2024-05-23 08:00:00	16.362	17.100	15.400	16.606	23161328.84
1589	2024-05-24 08:00:00	16.606	17.790	16.572	17.259	29407067.91
1590	2024-05-25 08:00:00	17.260	17.266	16.913	17.176	7251787.87

1591 rows × 6 columns

Save the datasets

In [ ]:
```
# save the datasets in order not to stream the data again when restart the kernel
eth.to_csv('eth_1day.csv')
near.to_csv('near_1day.csv')
link.to_csv('link_1day.csv')
```

1.1. Construct equally-weighted, minimum variance, optimize stock, rand weights portfolios

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.optimize import Bounds
from scipy.optimize import LinearConstraint
import warnings
```

In [3]:
```
# import series
eth = pd.read_csv('eth_5min.csv')
link = pd.read_csv('link_5min.csv')
near = pd.read_csv('near_5min.csv')

# just take the 'date' and 'close' columns and then rename the columns
eth = eth[['date', 'close']].rename(columns={'close': 'ETH'})
link = link[['date', 'close']].rename(columns={'close': 'LINK'})
near = near[['date', 'close']].rename(columns={'close': 'NEAR'})

# merge 3 series in 1 dataframe
df = pd.merge(pd.merge(eth, link, on='date', how = 'left'),near, on='date', how='le


df['date'] = pd.to_datetime(df['date']) # convert 'date' column into datetime forma
df.set_index('date', inplace=True) # set 'date' column as index
df.dropna(inplace=True) # drop rows with missing values
df
```

Out[3]:

date	ETH	LINK	NEAR
2020-10-15 08:00:00	377.63	10.752	1.1220
2020-10-16 08:00:00	365.41	10.580	0.8156
2020-10-17 08:00:00	368.26	10.616	0.8079
2020-10-18 08:00:00	378.29	10.936	0.8702
2020-10-19 08:00:00	379.47	10.920	0.8014
...	...	...	...
2024-05-16 08:00:00	2943.22	15.503	8.0170
2024-05-17 08:00:00	3090.48	16.222	8.0330
2024-05-18 08:00:00	3121.59	16.320	7.9170
2024-05-19 08:00:00	3069.98	16.551	7.7740
2024-05-20 08:00:00	3085.98	16.406	7.7970

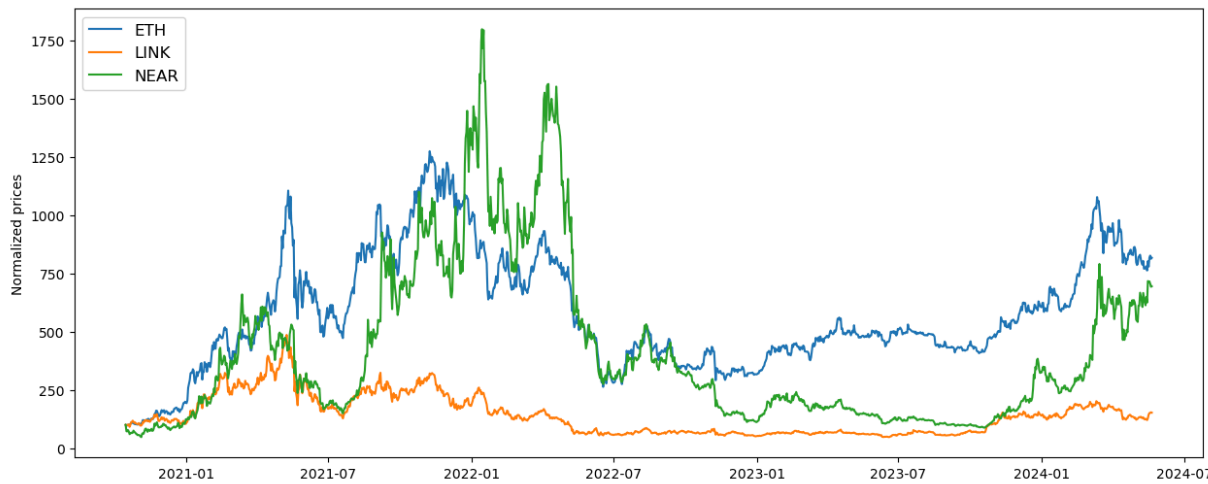
1314 rows × 3 columns

In [4]:

```
plt.figure(figsize=(15, 6))

# loop through each column in the normalized dataframe
for i in range(df.divide(df.iloc[0] / 100).shape[1]):
    # normalize the prices by dividing each column by its first value and multiplyi
    # this scales the initial price to 100 and shows relative changes
    normalized_prices = df.divide(df.iloc[0] / 100).iloc[:, i]

    # plot the normalized prices for each asset
    plt.plot(normalized_prices, label=df.divide(df.iloc[0] / 100).columns.values[i]

plt.legend(loc='upper left', fontsize=12)
plt.ylabel('Normalized prices')
# plt.show()
```

Out[4]:

```
Text(0, 0.5, 'Normalized prices')
```



- The price normalized graph of the individual assets (ETH, LINK, NEAR) shows the normalized prices of three cryptocurrencies: ETH, LINK, and NEAR. The x-axis represents time, ranging from January 2021 to July 2024. The y-axis represents the normalized price, with a initial value indicating the price at the beginning of the data range (January 2021).
- Overall Trend: While the graph doesn't show the actual prices, it seems all 3 cryptocurrencies have experienced volatility throughout the measured period. None have a clear upward or downward trend.
- Relative Performance: It appears that ETH and NEAR have outperformed LINK over the measured period. Their normalized prices are generally higher throughout the graph.

1.1.1. Equally weighted portfolio

Why do we have to construst an equally weighted portfolio for multiple series?

- Constructing an equally weighted portfolio for multiple series is popular because it's simple, diversifies risk among assets, avoids concentration risk, provides a benchmark for comparison, and historically has shown competitive performance in various market conditions.

To construct an equally weighted portfolio, i will take through 3 main steps:

1. **Define Equal Weights**: Modify the weights array to ensure each asset (ETH, LINK, NEAR) receives an equal weight. Since i have three assets, each will get a weight of 1/3
2. **Calculate Portfolio Return**: Use the *modified weights array* to compute the portfolio return as the *dot product* of *weights* and *expected returns*
3. **Calculate Portfolio Variance**: Update the calculation of *portfolio variance* using the *updated weights array* and the *covariance matrix*.

In [5]:
```python
# log return calculations
df['ret_ETH'] = np.log(df['ETH'].shift(-1)/df['ETH'])
df['ret_LINK'] = np.log(df['LINK'].shift(-1)/df['LINK'])
df['ret_NEAR'] = np.log(df['NEAR'].shift(-1)/df['NEAR'])
df.dropna(inplace=True)
df
```

Out[5]:

	ETH	LINK	NEAR	ret_ETH	ret_LINK	ret_NEAR
date						
2020-10-15 08:00:00	377.63	10.752	1.1220	-0.032895	-0.016126	-0.318944
2020-10-16 08:00:00	365.41	10.580	0.8156	0.007769	0.003397	-0.009486
2020-10-17 08:00:00	368.26	10.616	0.8079	0.026872	0.029698	0.074285
2020-10-18 08:00:00	378.29	10.936	0.8702	0.003114	-0.001464	-0.082363
2020-10-19 08:00:00	379.47	10.920	0.8014	-0.028684	-0.102921	-0.168837
...	...	...	...	...	...	...
2024-05-15 08:00:00	3030.66	13.861	8.0490	-0.029276	0.111954	-0.003984
2024-05-16 08:00:00	2943.22	15.503	8.0170	0.048822	0.045335	0.001994
2024-05-17 08:00:00	3090.48	16.222	8.0330	0.010016	0.006023	-0.014546
2024-05-18 08:00:00	3121.59	16.320	7.9170	-0.016671	0.014055	-0.018228
2024-05-19 08:00:00	3069.98	16.551	7.7740	0.005198	-0.008799	0.002954

1313 rows × 6 columns

In [6]:
```python
# expected return calculation
eth_expected_ret = df['ret_ETH'].sum()/len(df['ret_ETH'])
link_expected_ret = df['ret_LINK'].sum()/len(df['ret_LINK'])
near_expected_ret = df['ret_NEAR'].sum()/len(df['ret_NEAR'])

# standard deviation calculation
eth_std = df['ret_ETH'].std()
link_std = df['ret_LINK'].std()
near_std = df['ret_NEAR'].std()

print(f'ETH expected return: {eth_expected_ret}')
print(f'ETH standard deviation: {eth_std}\n')

print(f'LINK expected return: {link_expected_ret}')
print(f'LINK standard deviation: {link_std}\n')

print(f'NEAR expected return: {near_expected_ret}')
print(f'NEAR standard deviation: {near_std}\n')
```

```
ETH expected return: 0.0015999312046860122
ETH standard deviation: 0.04328820529215373

LINK expected return: 0.00032182432429532916
LINK standard deviation: 0.055761490164238856

NEAR expected return: 0.0014764860908049941
NEAR standard deviation: 0.06975792784287006
```

In [7]:

```
# calculate the Variance-Covariance matrix
cov_matrix = df[['ret_ETH', 'ret_LINK', 'ret_NEAR']].cov()
print('Variance-Covariance matrix:')
print(cov_matrix)
```

```
Variance-Covariance matrix:
           ret_ETH  ret_LINK  ret_NEAR
ret_ETH   0.001874  0.001862  0.001816
ret_LINK  0.001862  0.003109  0.002356
ret_NEAR  0.001816  0.002356  0.004866
```

In [8]:

```
# create an array of expected return for 3 assets
expected_returns = np.array([eth_expected_ret, link_expected_ret, near_expected_ret

# create an array of standard deviation for 3 assets
std_devs = np.array([eth_std, link_std, near_std])

# convert covariance matrix to numpy array
cov_matrix = cov_matrix.values
if cov_matrix.shape == (1,1,3):
    cov_matrix = cov_matrix[0, 0]
```

In [9]:

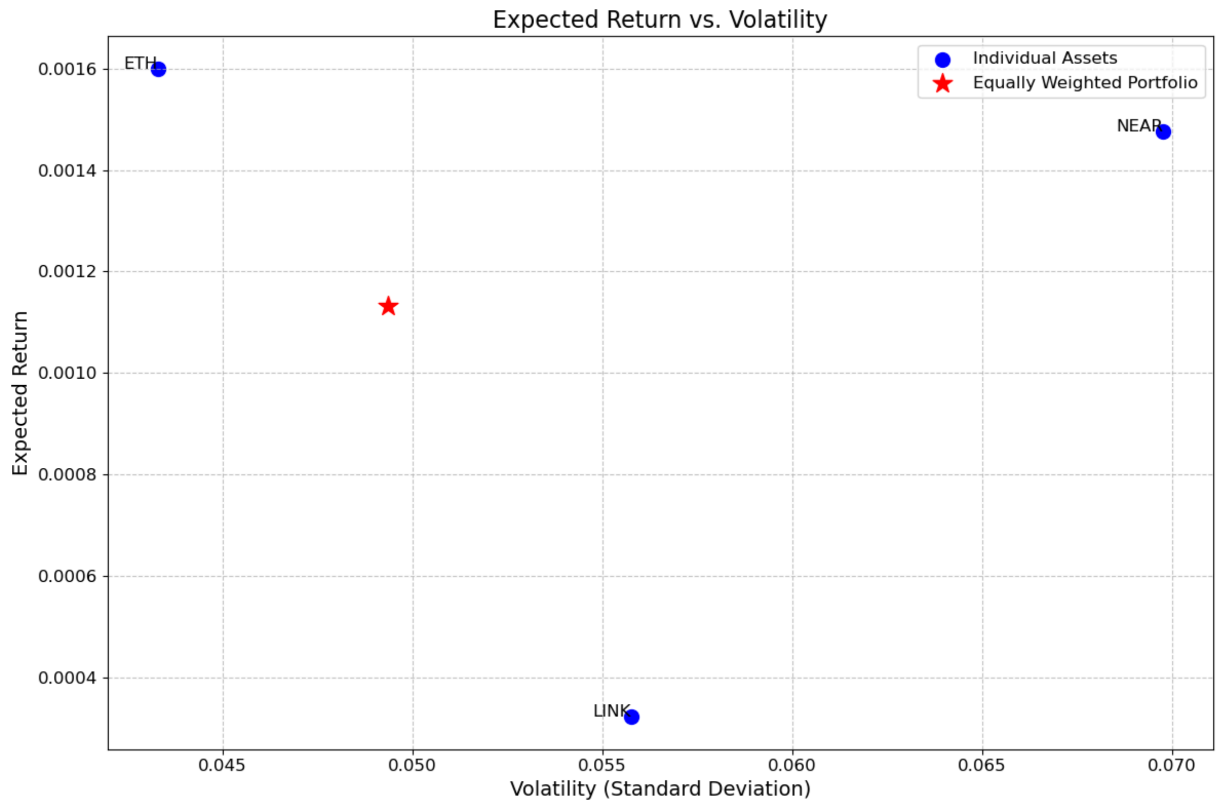
```
# assign weights for 3 assets
weights = np.array([1/3, 1/3, 1/3])

portfolio_return = np.dot(weights, expected_returns) # calculate portfolio return
portfolio_variance = np.dot(weights.T, np.dot(cov_matrix, weights)) # calculate por
portfolio_volatility = np.sqrt(portfolio_variance) # calculate portfolio standard d

print(f'Expected Portfolio Return: {portfolio_return}')
print(f'Portfolio Volatility: {portfolio_volatility}')
```

```
Expected Portfolio Return: 0.001132747206595445
Portfolio Volatility: 0.04934946941022965
```

In [10]:

```
plt.figure(figsize=(12,8))
plt.scatter(std_devs, expected_returns, c = 'blue', s = 100, label = 'Individual As
for i, txt in enumerate(['ETH', 'LINK', 'NEAR']):
    plt.annotate(txt, (std_devs[i], expected_returns[i]), fontsize=12, ha = 'right'
plt.scatter(portfolio_volatility, portfolio_return, c='red', marker = '*', s = 200,
plt.xlabel('Volatility (Standard Deviation)', fontsize=14)
plt.ylabel('Expected Return', fontsize=14)
plt.title('Expected Return vs. Volatility', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
plt.tight_layout()
plt.show()
```



Interpretation:

- **Equally Weighted Portfolio (Red Star)**:
    - **Position**: The red star is located towards the middle of the graph, which is a balanced approach in terms of expected return and volatility. This portfolio consists of an equal allocation of capital across all the individual assets listed (ETH, LINK, NEAR).
    - **Comparison to Individual Assets**: The equally weighted portfolio's position indicates that it provides a higher expected return than some individual assets like "LINK" (the asset with the lowest volatility), but at a slightly higher level of risk.

1.1.2. Minimum Variance Portfolio

Why do we have to construct minimum variance portfolio?

- The minimum variance portfolio prioritizes low risk over high returns. They're great for investors who dislike volatility and want to spread risk through diversification. While returns might be lower, it offers a potentially smoother investment ride.

In [11]:
```
def portfolio_return(weights, returns):
    return np.dot(weights, returns)
def portfolio_volatility(weights, cov_matrix):
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
```

```python
def objective_function(weights, cov_matrix): # objective function to be minimize
    return portfolio_volatility(weights, cov_matrix)
```

In [12]:
```python
# define the constraints to ensure that all the weights in the portfolio sum up to
constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}

# each weights can range from a minimum of 0 (no allocation) to 1 (100% allocation)
bounds = tuple((0,1) for _ in range(len(expected_returns)))

# starting point for optimization algorithm
initial_guess = np.array([1/3, 1/3, 1/3])

# Sequential Least Squares Programming optimization
result = minimize(objective_function, initial_guess, args = (cov_matrix,), method =
min_var_weights = result.x
```

In [13]:
```python
min_var_return = portfolio_return(min_var_weights, expected_returns)
min_var_volatility = portfolio_volatility(min_var_weights, cov_matrix)

print(f'Minimum Variance Portfolio Weights: {min_var_weights}')
print(f'Minimum Variance Portfolio Return: {min_var_return}')
print(f'Minimum Variance Portfolio Volatility: {min_var_volatility}')
```

```
Minimum Variance Portfolio Weights: [0.98098855 0.0011843  0.01782714]
Minimum Variance Portfolio Return: 0.0015962168640125758
Minimum Variance Portfolio Volatility: 0.043275720826949915
```

In [14]:
```python
plt.figure(figsize=(12,8))
plt.scatter(std_devs, expected_returns, c='blue', s=100, label='Individual Assets')
for i, txt in enumerate(['ETH', 'LINK', 'NEAR']):
    plt.annotate(txt, (std_devs[i], expected_returns[i]), fontsize = 12, ha = 'righ
plt.scatter(min_var_volatility, min_var_return, c = 'green', marker = '*', s=300, l
plt.xlabel('Volatility (Standard Deviation)', fontsize=14)
plt.ylabel('Expected Return', fontsize=14)
plt.title('Expected Return vs. Volatility', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```

Interpretation:

- **Minimum Variance Portfolio (Green Star)**:
  - **Position**: This portfolio is positioned with a lower expected return compared to the "NEAR" portfolio, but it also has lower volatility. The green star's location near the bottom left corner of the plot suggests that it is designed to minimize risk.
  - **Strategy**: The strategy behind this portfolio is focused on reducing risk to the lowest possible level among the available asset choices, possibly by selecting assets that have lower volatility or that are negatively correlated with each other to reduce overall portfolio variance.

1.1.3. Optimized stock portfolio

Why do we need to construct the optimized stock portfolio?

- Portfolio optimization balances risk and return by combining risky and safe investments in a ratio that matches the investor’s risk tolerance.

In [15]:
```
def sharpe_ratio(weights, returns, cov_matrix, risk_free_rate = 0.04422): # calcula
    port_return = portfolio_return(weights,  returns)
    port_volatility = portfolio_volatility(weights, cov_matrix)
    return (port_return - risk_free_rate)/port_volatility
def negative_sharpe_ratio(weights, returns, cov_matrix, risk_free_rate = 0.04422):
    return -sharpe_ratio(weights, returns, cov_matrix, risk_free_rate)
```

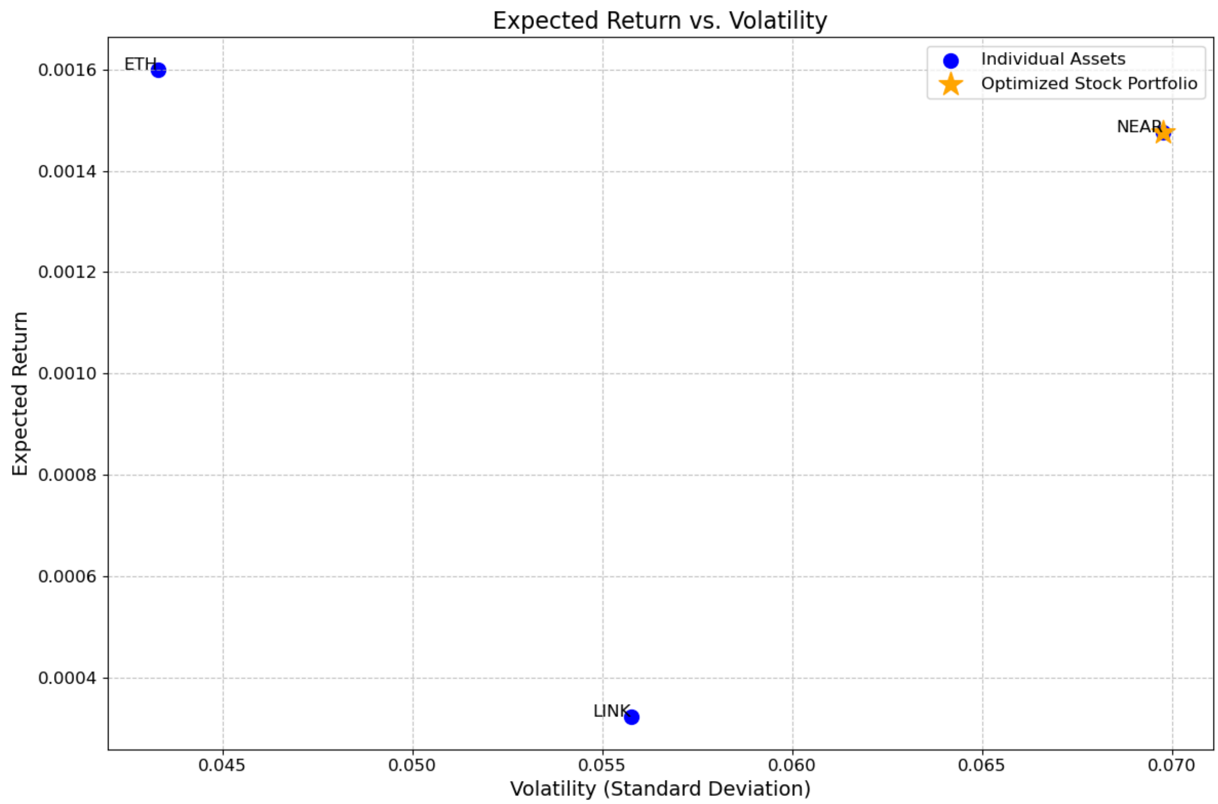
- The risk-free rate is calculated by taking the US 10-Year Gov bonds

In [16]:
```
constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}
bounds = tuple((0,1) for _ in range(len(expected_returns)))
initial_guess = np.array([1/3, 1/3, 1/3])
result = minimize(negative_sharpe_ratio, initial_guess, args = (expected_returns, c
opt_weights = result.x
```

In [17]:
```
opt_return = portfolio_return(opt_weights, expected_returns)
opt_volatility = portfolio_volatility(opt_weights, cov_matrix)

print(f'Optimized Portfolio Weights: {opt_weights}')
print(f'Optimized Portfolio Return: {opt_return}')
print(f'Optimized Portfolio Volatility: {opt_volatility}')
print(f'Optimized Portfolio Sharpe Ratio: {sharpe_ratio(opt_weights, expected_retur
```

```
Optimized Portfolio Weights: [0. 0. 1.]
Optimized Portfolio Return: 0.0014764860908049941
Optimized Portfolio Volatility: 0.06975792784287006
Optimized Portfolio Sharpe Ratio: -0.6127405906533648
```

In [18]:
```
plt.figure(figsize=(12,8))
plt.scatter(std_devs, expected_returns, c='blue', s = 100, label = 'Individual Asse
for i, txt in enumerate(['ETH', 'LINK', 'NEAR']):
    plt.annotate(txt, (std_devs[i], expected_returns[i]), fontsize = 12, ha='right'
plt.scatter(opt_volatility, opt_return, c='orange', marker = '*', s=300, label = 'O
plt.xlabel('Volatility (Standard Deviation)', fontsize=14)
plt.ylabel('Expected Return', fontsize=14)
plt.title('Expected Return vs. Volatility', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```



Interpretation:

- The graph shows the relationship between expected return and volatility for various assets and an optimized stock portfolio

1. **Individual Assets (Blue Dots)**:

- Each blue dot represents an individual asset. The x-axis shows the volatility (or standard deviation) of the asset, which measures the risk or uncertainty associated with the asset's returns. The y-axis shows the expected return, indicating the average return that one might expect from investing in the asset. Assets closer to the left side of the graph have lower volatility, whereas those higher up have higher expected returns.

2. **Optimized Stock Portfolio (Yellow Star)**:

- The yellow star labeled "NEAR" indicates the position of an optimized stock portfolio, which is strategically selected to balance risk (volatility) and return. It is positioned towards the right side, which typically suggests higher volatility, but it is also higher up, indicating a higher expected return compared to the individual assets.

1.1.4. Random Weights Portfolio

Why do we need to construct random weights portfolios?

- Random weight portfolios is a useful tool for benchmarking, exploring the efficient frontier, and achieving basic diversification. However, they are not a guaranteed path to optimal portfolio performance.

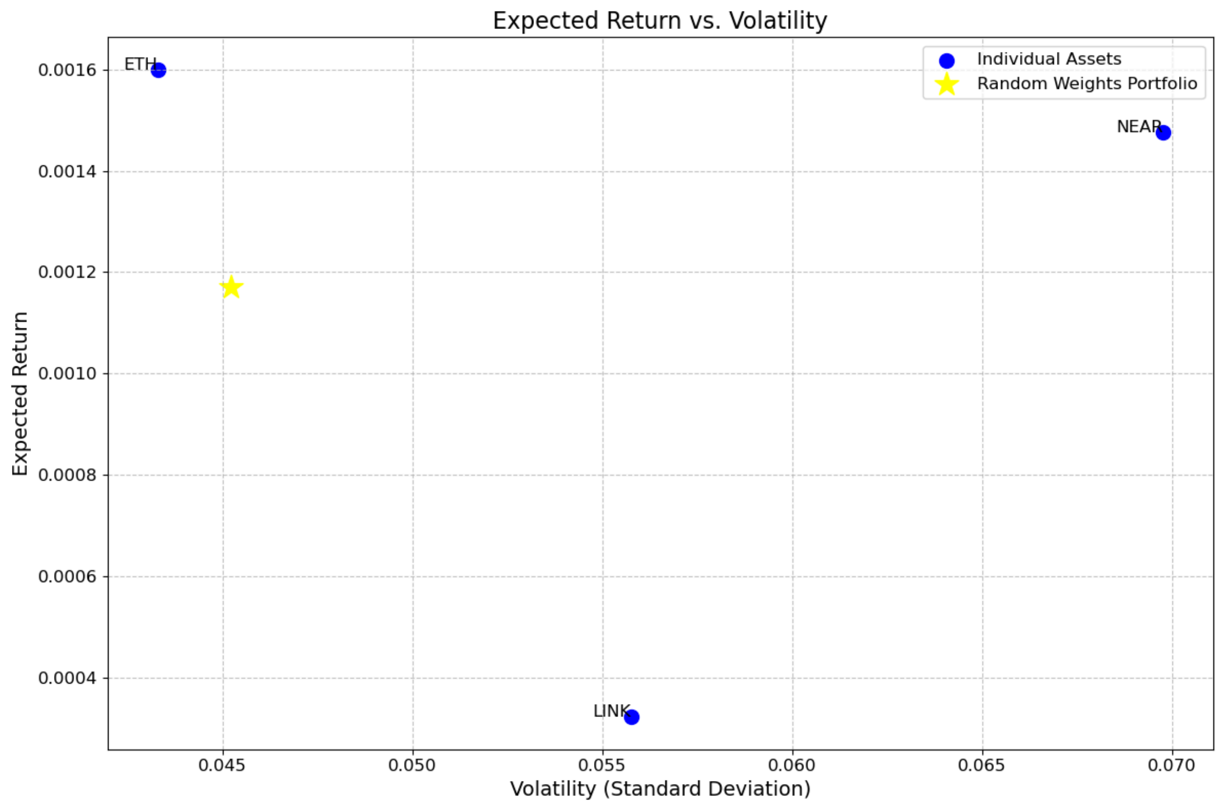
In [19]:
```
def generate_random_weights(num_assets):
    weights = np.random.random(num_assets)
    return weights/np.sum(weights)
```

In [20]:
```
num_assets = 3
random_weights = generate_random_weights(num_assets)
```

In [21]:
```
random_portfolio_return = portfolio_return(random_weights, expected_returns)
random_portfolio_volatility = portfolio_volatility(random_weights, cov_matrix)

print(f'Random Portfolio Weights: {random_weights}')
print(f'Random Portfolio Return: {random_portfolio_return}')
print(f'Random Portfolio Volatility: {random_portfolio_volatility}')
```

```
Random Portfolio Weights: [0.58692932 0.32836464 0.08470604]
Random Portfolio Return: 0.0011697895556550683
Random Portfolio Volatility: 0.045224033104420136
```

In [22]:
```
plt.figure(figsize=(12,8))
plt.scatter(std_devs, expected_returns, c='blue', s=100, label='Individual Assets')
for i, txt in enumerate(['ETH', 'LINK', 'NEAR']):
    plt.annotate(txt, (std_devs[i], expected_returns[i]), fontsize=12, ha='right')
plt.scatter(random_portfolio_volatility, random_portfolio_return, c='yellow', marke
plt.xlabel('Volatility (Standard Deviation)', fontsize=14)
plt.ylabel('Expected Return', fontsize=14)
plt.title('Expected Return vs. Volatility', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```



Interpretation:

- **Random Weights Portfolio (Yellow Star)**:
  - **Position**: The yellow star is positioned roughly in the middle of the graph in terms of both expected return and volatility. This suggests that the portfolio is composed of a random allocation of weights to the individual assets, resulting in a performance that balances risk and return.
  - **Performance**: Compared to the individual assets shown, this portfolio does not achieve the highest expected return nor the lowest volatility. Its position indicates a moderate expected return with a moderate level of risk.

1.1.5. Efficient Frontier

What is the Efficient Frontier?

- The efficient frontier is the set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return

In [23]:
```
df2 = df[['ETH', 'LINK', 'NEAR']] # extract closing prices columns
df3 = df2.pct_change() # daily percentage change
df4 = df3.iloc[1:len(df3.index), :] # remove NaN

# calculate annualized mean return
trading_days_per_year = 365
annualization_factor = (trading_days_per_year)/(1)
r = np.mean(df4, axis = 0)*annualization_factor
```

```python
# calculate covariance matrix
covar = df4.cov()
```

In [24]:
```python
# function for computing portfolio return
def ret(r,w):
    return r.dot(w)

# function for computing portfolio volatility
def vol(w,covar):
    return np.sqrt(np.dot(w, np.dot(w,covar)))

# function for computing sharpe ratio
def sharpe(ret,vol):
    return ret/vol
```

In [25]:
```python
bounds = Bounds(0,1) # each weights between 0 and 1

linear_constraint = LinearConstraint(np.ones((df3.shape[1],), dtype=int),1,1) # sum

weights = np.ones(df3.shape[1]) # create an initial array, set to 1
x0 = weights/np.sum(weights) # normalized initial array
fun1 = lambda w: np.sqrt(np.dot(w, np.dot(w, covar))) # define objective to minimiz

# minimize objective function using initial guess x0 base on 'trust-constr' method
res = minimize(fun1, x0, method = 'trust-constr', constraints = linear_constraint,

# store the optimized weights
w_min = res.x

np.set_printoptions(suppress=True, precision=2)
print(w_min)
print('return: % .2f'% (ret(r,w_min)*100), 'risk: % .3f'% vol(w_min,covar))
```

```
[0.94 0.03 0.03]
return:  93.24 risk:  0.043
```

Interpretation on the results:

- ETH (Ethereum): 94% of the portfolio
- LINK (Chainlink): 3% of the portfolio
- NEAR (NEAR Protocol): 3% of the portfolio
- The portfolio is heavily weighted towards ETH (94%), which likely has a significantly higher expected return compared to LINK and NEAR. This results in a very high expected portfolio return of 93.24%.
- Despite the high expected return, the portfolio's risk is relatively low at 4.3%. This suggests that ETH's returns have low volatility or that there is some diversification benefit (though minimal given the high concentration in ETH).

- The portfolio is highly concentrated in ETH, which exposes it to significant idiosyncratic risk specific to ETH. While the portfolio's overall volatility is low, a high concentration in a single asset can be risky if that asset experiences significant adverse events.

In [26]:
```
# objective function to maximize sharpe ratio
fun2 = lambda w: np.sqrt(np.dot(w, np.dot(w,covar)))/r.dot(w)
res_sharpe = minimize(fun2, x0, method = 'trust-constr', constraints = linear_const

# store the optimized weights
w_sharpe = res_sharpe.x
print(w_sharpe)
print('return: % .2f'% (ret(r,w_sharpe)*100), 'risk: % .3f'% vol(w_sharpe, covar))
```

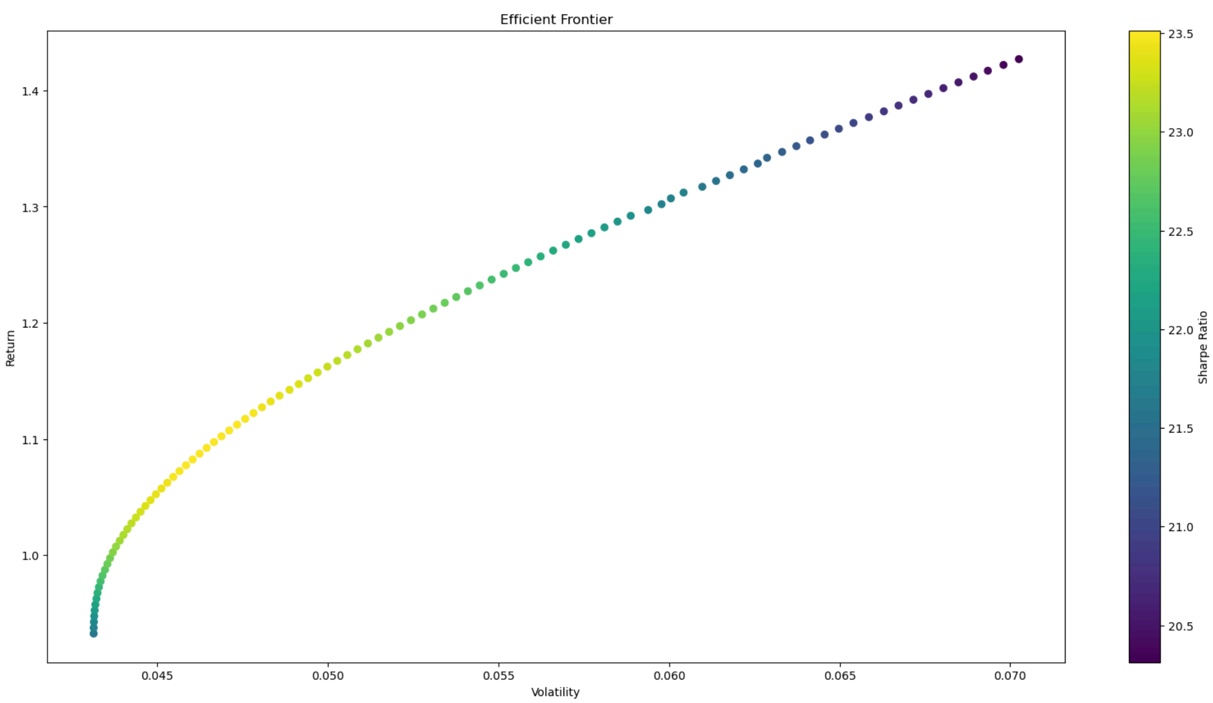
```
[0.67 0.   0.33]
return:  109.34 risk:  0.046
```

Results interpretation:

- ETH (Ethereum): 67% of the portfolio
- LINK (Chainlink): 0% of the portfolio
- NEAR (NEAR Protocol): 33% of the portfolio
- The portfolio is expected to achieve an exceptionally high return of 109.34% annually. This high return is primarily driven by the allocations to ETH and NEAR, which likely have higher individual expected returns compared to LINK.
- Despite the very high expected return, the portfolio's risk is relatively low at 4.6%. This suggests a favorable risk-return profile, with high returns achieved without a proportionately high increase in volatility.
- The portfolio excludes LINK entirely and focuses only on ETH and NEAR. While this might optimize the risk-return profile in this specific context, it does introduce concentration risk. Excluding an asset completely can expose the portfolio to higher idiosyncratic risk if either ETH or NEAR underperforms or experiences volatility.

In [27]:
```
w = w_min
num_ports = 100
gap = (np.amax(r) - ret(r, w_min))/num_ports

all_weights = np.zeros((num_ports, len(df4.columns)))
all_weights[0], all_weights[1] = w_min, w_sharpe
ret_arr = np.zeros(num_ports)
ret_arr[0], ret_arr[1] = ret(r, w_min), ret(r, w_sharpe)
vol_arr = np.zeros(num_ports)
vol_arr[0], vol_arr[1] = vol(w_min, covar), vol(w_sharpe, covar)

warnings.filterwarnings('ignore')

# loop to generate different portfolios
```

```
for i in range(num_ports):
    port_ret = ret(r,w) + i*gap
    double_constraint = LinearConstraint([np.ones(df3.shape[1]), r], [1,port_ret],

    x0 = w_min
    fun = lambda w: np.sqrt(np.dot(w, np.dot(w,covar)))
    a = minimize(fun, x0, method = 'trust-constr', constraints=double_constraint, b

    all_weights[i, :] = a.x
    ret_arr[i] = port_ret
    vol_arr[i] = vol(a.x, covar)

sharpe_arr = ret_arr/vol_arr

plt.figure(figsize=(20,10))
plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.title('Efficient Frontier')
plt.show()
```



1. **Efficient Frontier Curve:**

- **Shape and Trend**: The curve starts at the bottom left, indicating portfolios with the lowest volatility and progresses upwards to the right, showing increasing volatility as well as potential returns. This upward sloping nature of the frontier reflects the trade-off between risk and return: higher expected returns are generally associated with higher risk (volatility).
- **Risk-Return Trade-off**: At the lower end of the curve (leftmost), the increase in return per unit of additional risk is more significant compared to the higher end of the curve.

As we move rightward along the curve, each increment in return comes at a cost of proportionally more risk.

2. **Color Gradient (Sharpe Ratio):**

- **Interpretation**: The color represents the Sharpe ratio, which adjusts returns for risk. A higher Sharpe ratio means a more desirable portfolio because it indicates a higher return per unit of risk.

- **Gradient Transition**: The color changes from green to purple as we move along the curve. Starting with green (lower Sharpe ratio) and moving to purple (higher Sharpe ratio), suggests that as volatility and returns increase, the portfolios initially offer less favorable risk-adjusted returns. However, at a certain point, the risk-adjusted returns start improving significantly, as indicated by the transition to purple.

- **Optimal Portfolios**: The segments of the frontier that are purple represent portfolios that have the highest Sharpe ratios, meaning they are the most efficient in terms of balancing expected returns against their risk. These would be considered optimal choices for investment.

3. **Conclusion:**

- **Portfolio Selection**: Investors should look towards the segment of the frontier that aligns with their risk tolerance and desired returns. Those who seek the most efficient use of their risk budget should focus particularly on the portions of the curve that exhibit higher Sharpe ratios (purple color).

- **Strategic Implications**: The Efficient Frontier provides a visual tool for understanding and analyzing different portfolios' performance characteristics. It helps in making informed decisions about how to allocate assets to maximize returns for a given level of risk.

1.2. Calculate log returns, absolute returns, squared returns of the series, descriptive statistics, stationary test

1.2.1. Calculate returns

In [28]:
```
df['ret_port'] = (df['ret_ETH'] + df['ret_LINK'] + df['ret_NEAR'])/3 # calculate po

# calculate absolute returns
eth_abs_ret = ((df['ETH'] - df['ETH'].iloc[0])/df['ETH'].iloc[0]) * 100
link_abs_ret = ((df['LINK'] - df['LINK'].iloc[0])/df['LINK'].iloc[0]) * 100
near_abs_ret = ((df['NEAR'] - df['NEAR'].iloc[0])/df['NEAR'].iloc[0]) * 100

# calculate squared returns
```

```
eth_sq_ret = np.square(df['ret_ETH'])
link_sq_ret = np.square(df['ret_LINK'])
near_sq_ret = np.square(df['ret_NEAR'])
port_sq_ret = np.square(df['ret_port'])
```

In [29]:
```
df
```

Out[29]:

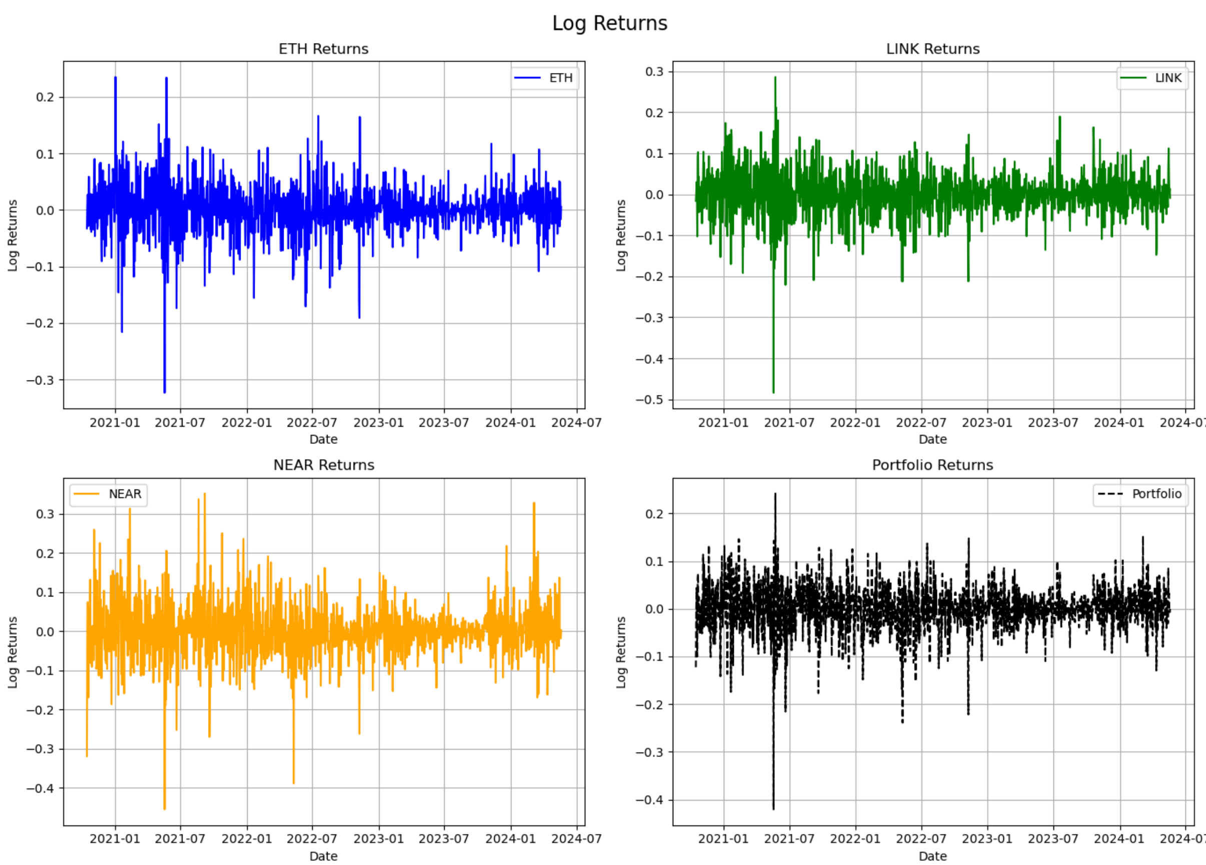
	ETH	LINK	NEAR	ret_ETH	ret_LINK	ret_NEAR	ret_port
date							
2020-10-15 08:00:00	377.63	10.752	1.1220	-0.032895	-0.016126	-0.318944	-0.122655
2020-10-16 08:00:00	365.41	10.580	0.8156	0.007769	0.003397	-0.009486	0.000560
2020-10-17 08:00:00	368.26	10.616	0.8079	0.026872	0.029698	0.074285	0.043618
2020-10-18 08:00:00	378.29	10.936	0.8702	0.003114	-0.001464	-0.082363	-0.026904
2020-10-19 08:00:00	379.47	10.920	0.8014	-0.028684	-0.102921	-0.168837	-0.100147
...	...	...	...	...	...	...	...
2024-05-15 08:00:00	3030.66	13.861	8.0490	-0.029276	0.111954	-0.003984	0.026232
2024-05-16 08:00:00	2943.22	15.503	8.0170	0.048822	0.045335	0.001994	0.032050
2024-05-17 08:00:00	3090.48	16.222	8.0330	0.010016	0.006023	-0.014546	0.000498
2024-05-18 08:00:00	3121.59	16.320	7.9170	-0.016671	0.014055	-0.018228	-0.006948
2024-05-19 08:00:00	3069.98	16.551	7.7740	0.005198	-0.008799	0.002954	-0.000216

1313 rows × 7 columns

Plotting the Log Returns

In [30]:
```
fig, axs = plt.subplots(2, 2, figsize=(14, 10))

axs[0, 0].plot(df['ret_ETH'], label='ETH', color='blue')
axs[0, 0].set_title('ETH Returns')
axs[0, 0].set_xlabel('Date')
axs[0, 0].set_ylabel('Log Returns')
axs[0, 0].legend()
axs[0, 0].grid(True)
```

```
axs[0, 1].plot(df['ret_LINK'], label='LINK', color='green')
axs[0, 1].set_title('LINK Returns')
axs[0, 1].set_xlabel('Date')
axs[0, 1].set_ylabel('Log Returns')
axs[0, 1].legend()
axs[0, 1].grid(True)

axs[1, 0].plot(df['ret_NEAR'], label='NEAR', color='orange')
axs[1, 0].set_title('NEAR Returns')
axs[1, 0].set_xlabel('Date')
axs[1, 0].set_ylabel('Log Returns')
axs[1, 0].legend()
axs[1, 0].grid(True)

axs[1, 1].plot(df['ret_port'], label='Portfolio', linestyle='--', color='black')
axs[1, 1].set_title('Portfolio Returns')
axs[1, 1].set_xlabel('Date')
axs[1, 1].set_ylabel('Log Returns')
axs[1, 1].legend()
axs[1, 1].grid(True)

fig.suptitle('Log Returns', fontsize=16)
plt.tight_layout()
plt.show()
```



- **ETH Log Returns (Top Left)**

- **Volatility**: Ethereum shows significant volatility, with log returns frequently spiking both positively and negatively. This high volatility is typical for cryptocurrencies, which are known for their rapid price changes.
- **Trend**: There is no clear long-term upward or downward trend in the data, indicating that ETH's price fluctuations are relatively symmetric around the mean.

- **LINK Log Returns (Top Right)**
  - **Volatility**: LINK also exhibits substantial volatility, although the range of returns is slightly less extreme than ETH's. The log returns are clustered more densely, suggesting slightly less extreme short-term price changes compared to ETH.
  - **Consistency**: The returns show a consistent variability over the period, without any apparent increase or decrease in volatility over time.

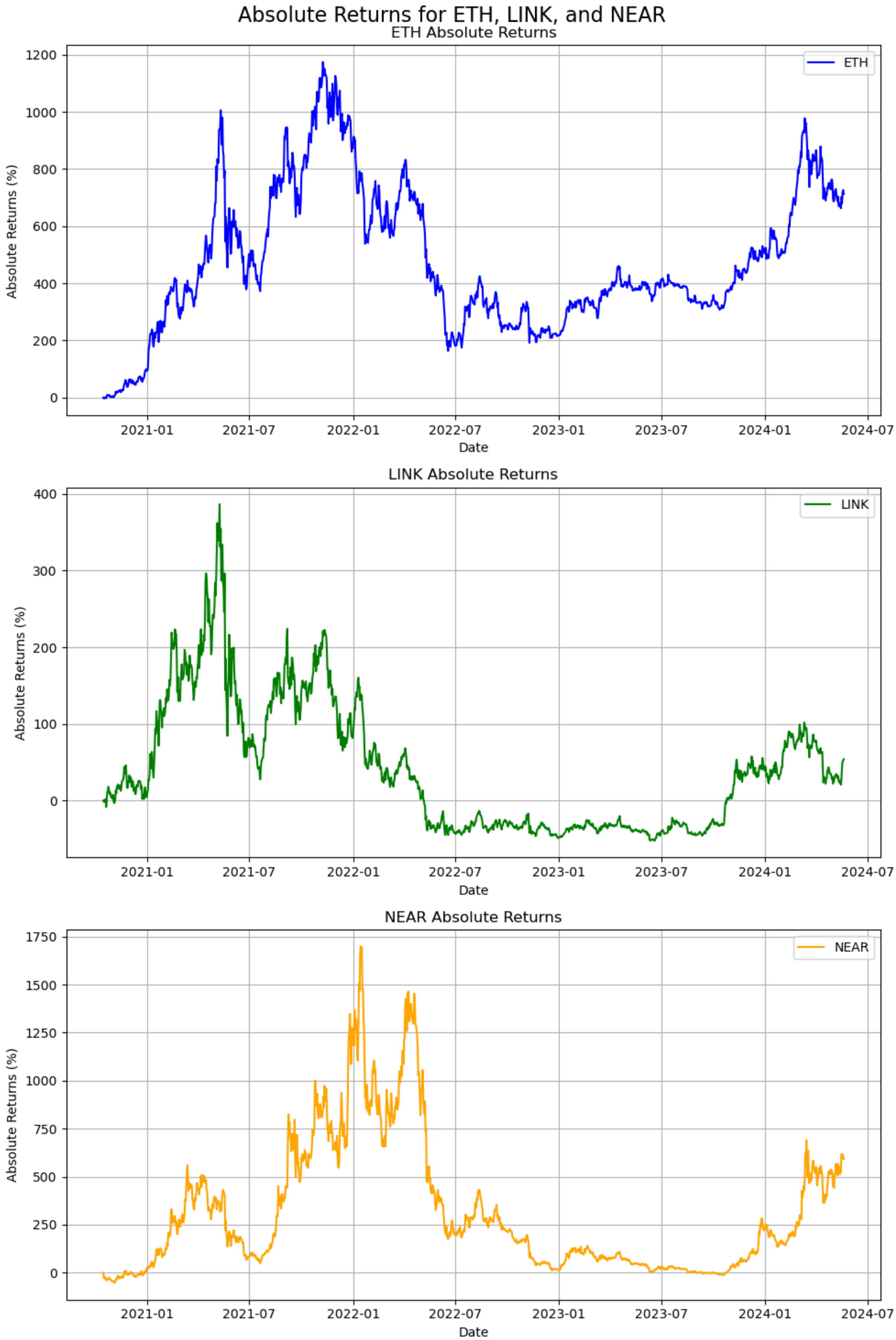
- **NEAR Log Returns (Bottom Left)**
  - **Volatility**: NEAR exhibits a range of returns somewhat similar to that of the individual assets but appears slightly more stable.
  - **Peaks**: The occasional spikes in returns suggest that there are still times of significant profit or loss, which could be due to the performance of specific assets within the portfolio or market events impacting the portfolio's overall value.

- **Portfolio Log Returns (Bottom Right)**
  - **Volatility**: This portfolio appears to have the highest frequency of returns clustered around the mean, which suggests it's more diversified or balanced compared to the individual assets.
  - **Stability**: The returns are more consistent, with fewer extreme values compared to the individual cryptocurrencies. This stability is a typical advantage of diversified portfolios, which can spread risk across different assets.

Plotting the Absolute Returns

In [31]:
```
fig, axs = plt.subplots(3, 1, figsize=(10, 15))

axs[0].plot(eth_abs_ret, label='ETH', color='blue')
axs[0].set_title('ETH Absolute Returns')
axs[0].set_xlabel('Date')
axs[0].set_ylabel('Absolute Returns (%)')
axs[0].legend()
axs[0].grid(True)

axs[1].plot(link_abs_ret, label='LINK', color='green')
axs[1].set_title('LINK Absolute Returns')
axs[1].set_xlabel('Date')
axs[1].set_ylabel('Absolute Returns (%)')
```

```python
axs[1].legend()
axs[1].grid(True)

axs[2].plot(near_abs_ret, label='NEAR', color='orange')
axs[2].set_title('NEAR Absolute Returns')
axs[2].set_xlabel('Date')
axs[2].set_ylabel('Absolute Returns (%)')
axs[2].legend()
axs[2].grid(True)

fig.suptitle('Absolute Returns for ETH, LINK, and NEAR', fontsize=16)
plt.tight_layout()
plt.show()
```


Absolute Returns for ETH, LINK, and NEAR
ETH Absolute Returns
ETH
Absolute Returns (%)
0
200
400
600
800
1000
1200
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Date
LINK Absolute Returns
LINK
Absolute Returns (%)
0
100
200
300
400
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Date
NEAR Absolute Returns
NEAR
Absolute Returns (%)
0
250
500
750
1000
1250
1500
1750
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Date

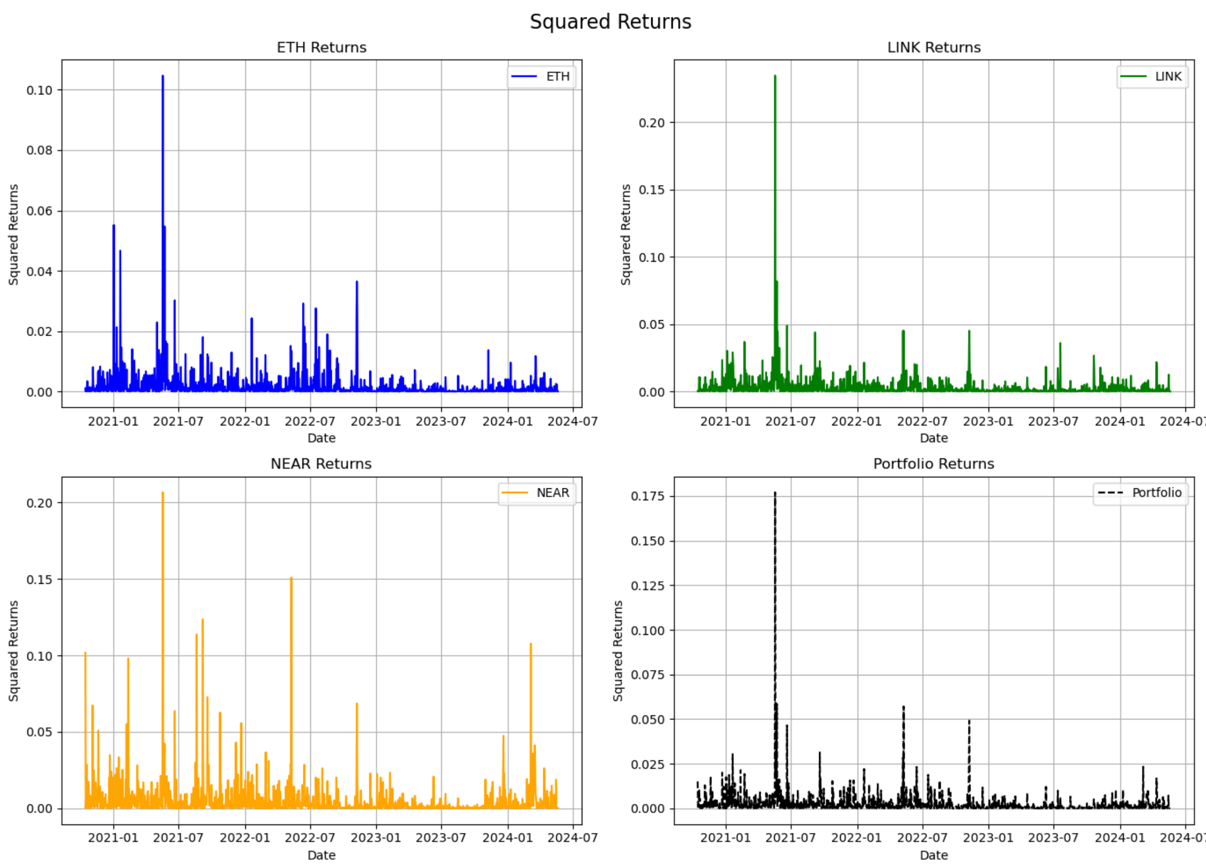
- **ETH Absolute Returns (Top Chart)**
  - **Growth Pattern**: Ethereum exhibits significant growth in the early period, peaking around early 2022. Thereafter, it shows a sharp decline before stabilizing and then rising again.
  - **Volatility**: The chart reflects high volatility in ETH's price, characterized by sharp increases and equally rapid declines, which is typical for many cryptocurrencies. The returns exceed 1000% at their peak but also drop significantly, showing the high-risk, high-reward nature of this investment.
- **LINK Absolute Returns (Middle Chart)**
  - **Return Pattern**: LINK's returns show a significant peak around mid-2021, reaching nearly 300%. Following this, the returns decline steadily before starting to recover in 2023.
  - **Volatility and Stability**: Similar to ETH, LINK shows volatility but not to the same extreme. The decline phase is more gradual compared to the sharp fluctuations seen in ETH's chart.
- **NEAR Absolute Returns (Bottom Chart)**
  - **Performance Peaks**: NEAR shows multiple peaks throughout the observed period, with the highest peaks surpassing 1500% returns. This suggests moments of extremely high profitability.
  - **Comparison to Single Assets**: The volatility in the NEAR is notable but generally appears more contained than in ETH’s returns. This could suggest some level of diversification or active management that tempers the extreme ups and downs typical of single cryptocurrency investments.

Plotting the Squared Returns

In [32]:

```
fig, axs = plt.subplots(2, 2, figsize=(14, 10))

axs[0, 0].plot(eth_sq_ret, label='ETH', color='blue')
axs[0, 0].set_title('ETH Returns')
axs[0, 0].set_xlabel('Date')
axs[0, 0].set_ylabel('Squared Returns')
axs[0, 0].legend()
axs[0, 0].grid(True)

axs[0, 1].plot(link_sq_ret, label='LINK', color='green')
axs[0, 1].set_title('LINK Returns')
axs[0, 1].set_xlabel('Date')
axs[0, 1].set_ylabel('Squared Returns')
```

```
axs[0, 1].legend()
axs[0, 1].grid(True)

axs[1, 0].plot(near_sq_ret, label='NEAR', color='orange')
axs[1, 0].set_title('NEAR Returns')
axs[1, 0].set_xlabel('Date')
axs[1, 0].set_ylabel('Squared Returns')
axs[1, 0].legend()
axs[1, 0].grid(True)

axs[1, 1].plot(port_sq_ret, label='Portfolio', linestyle='--', color='black')
axs[1, 1].set_title('Portfolio Returns')
axs[1, 1].set_xlabel('Date')
axs[1, 1].set_ylabel('Squared Returns')
axs[1, 1].legend()
axs[1, 1].grid(True)

fig.suptitle('Squared Returns', fontsize=16)
plt.tight_layout()
plt.show()
```



- **ETH Squared Returns (Top Left)**
  - **Observations**: The ETH chart shows numerous spikes, particularly notable in the early part of the time series. This indicates periods of high volatility, where ETH experienced significant price movements either upwards or downwards.

    - **Implication**: The high spikes in squared returns suggest that ETH investments carry a high risk, with potential for large gains or losses. This is typical for cryptocurrencies, which are known for their unpredictability and sharp price fluctuations.
- **LINK Squared Returns (Top Right)**
    - **Observations**: Similar to ETH, LINK displays several spikes, but with less frequency and intensity. The highest spike is notable around mid-2021.
    - **Implication**: LINK also exhibits considerable volatility, though possibly slightly less than ETH based on the comparative heights of the spikes. Investors in LINK should be prepared for potentially large price swings, though these may occur less frequently than in ETH.
- **NEAR Squared Returns (Bottom Left)**
    - **Observations**: NEAR shows spikes, but these are generally lower and less frequent than those observed in the individual cryptocurrency assets (ETH and LINK). This suggests periods of heightened activity or market movements affecting the portfolio.
    - **Implication**: While still displaying signs of volatility, NEAR appears to manage risk better than the individual cryptocurrency investments. The reduced height and frequency of spikes could indicate effective diversification or risk management strategies.
- **Portfolio Squared Returns (Bottom Right)**
    - **Observations**: This general portfolio shows numerous low-height spikes distributed throughout the observed period, indicating consistent, moderate-level volatility.
    - **Implication**: This portfolio appears to be the most stable among those analyzed, with its lower and more consistent spikes suggesting a well-diversified or balanced investment strategy that mitigates high volatility.

1.2.2. Descriptive Statistics

In [33]:
```
desc_stats = df.describe()
desc_stats
```

Out[33]:

	ETH	LINK	NEAR	ret_ETH	ret_LINK	ret_NEAR	r
count	1313.000000	1313.000000	1313.000000	1313.000000	1313.000000	1313.000000	1313
mean	2174.289634	15.094087	4.585600	0.001600	0.000322	0.001476	0
std	941.254026	9.027084	3.860793	0.043288	0.055761	0.069758	0
min	365.410000	5.129000	0.533900	-0.323577	-0.484226	-0.454605	-0
25%	1579.400000	7.169000	1.721000	-0.017972	-0.028947	-0.032892	-0
50%	1890.560000	13.552000	3.236600	0.001537	0.003165	-0.000604	0
75%	2889.080000	20.067000	6.214000	0.023503	0.030264	0.035602	0
max	4814.300000	52.311000	20.183800	0.234909	0.286017	0.351644	0

Variables:

- **ETH, LINK, NEAR**: These columns represent the prices of Ethereum (ETH), Chainlink (LINK), and NEAR Protocol (NEAR) over a certain period.
- **ret_ETH, ret_LINK, ret_NEAR**: These columns represent the daily log returns of ETH, LINK, and NEAR, respectively.
- **ret_port**: This column represents the daily return of a portfolio composed of ETH, LINK, and NEAR.

Statistics:

- **count**: The number of observations (1313), which indicates the number of daily price and return data points available for each cryptocurrency and the portfolio.
- **mean**: The average value over the observation period.
- **std (standard deviation)**: A measure of the dispersion or volatility of the prices and returns.
- **min**: The minimum value observed.
- **25% (first quartile)**: The value below which 25% of the observations fall.
- **50% (median)**: The middle value of the dataset.
- **75% (third quartile)**: The value below which 75% of the observations fall.
- **max**: The maximum value observed.

Now, i will take the Brownian motion test for the series, consists of:

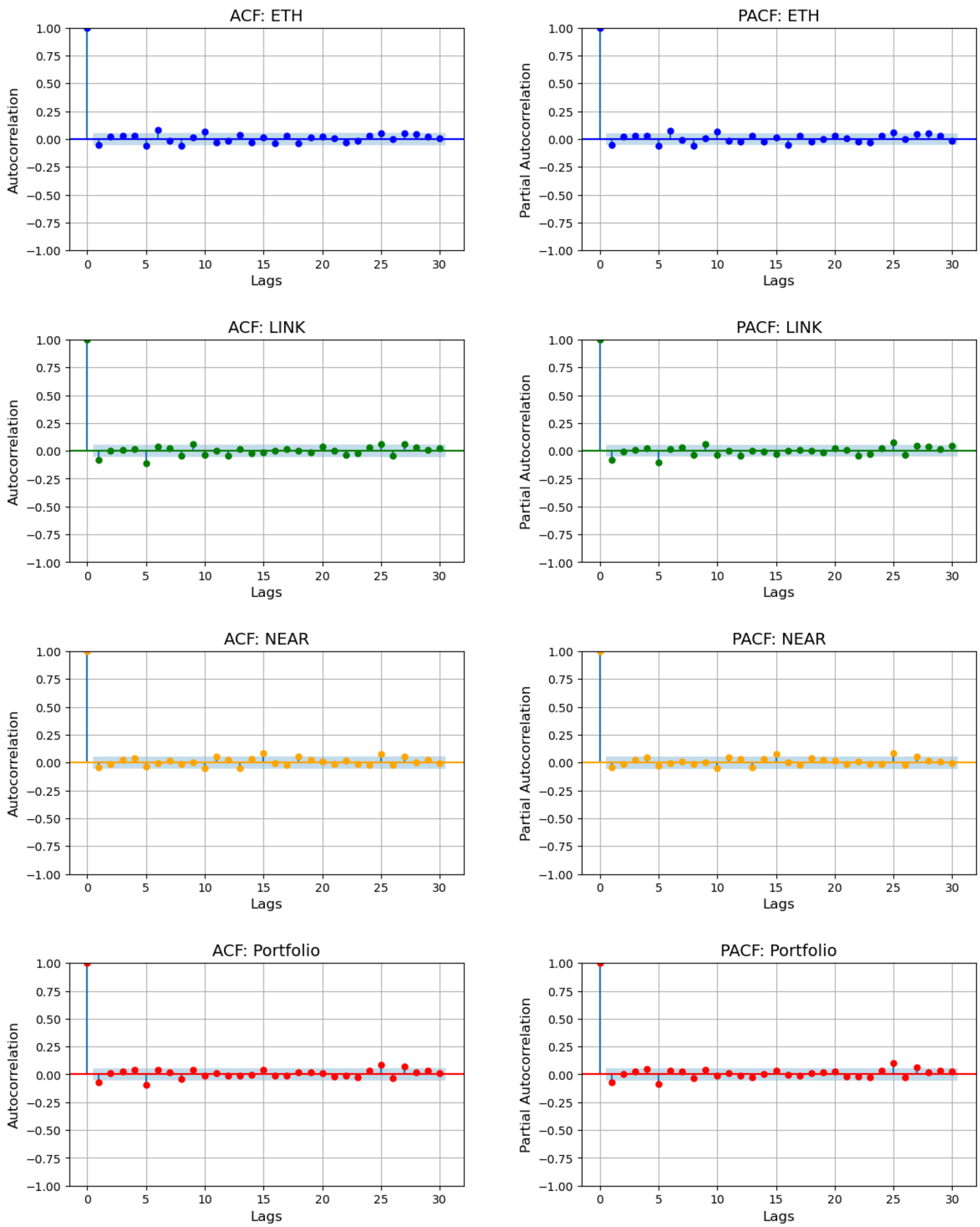
- **Stationary test**
- **Increments Normality test**
- **Increments Independence test**

1.2.3. Stationary

In [34]:
```python
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
```

ACF and PACF approach

In [35]:
```python
# Number of lags
lags = 30

# Create subplots
fig, axs = plt.subplots(4, 2, figsize=(14, 18))

# Adjust the space between plots
plt.subplots_adjust(hspace=0.4, wspace=0.3)

# Function to plot ACF and PACF with enhanced visibility
def plot_acf_pacf(series, lags, ax1, ax2, title, color):
    plot_acf(series, lags=lags, ax=ax1, color=color)
    ax1.set_title(f'ACF: {title}', fontsize=14)
    ax1.set_xlabel('Lags', fontsize=12)
    ax1.set_ylabel('Autocorrelation', fontsize=12)
    ax1.grid(True)

    plot_pacf(series, lags=lags, ax=ax2, color=color)
    ax2.set_title(f'PACF: {title}', fontsize=14)
    ax2.set_xlabel('Lags', fontsize=12)
    ax2.set_ylabel('Partial Autocorrelation', fontsize=12)
    ax2.grid(True)

# ETH
plot_acf_pacf(df['ret_ETH'], lags, axs[0, 0], axs[0, 1], 'ETH', 'blue')

# LINK
plot_acf_pacf(df['ret_LINK'], lags, axs[1, 0], axs[1, 1], 'LINK', 'green')

# NEAR
plot_acf_pacf(df['ret_NEAR'], lags, axs[2, 0], axs[2, 1], 'NEAR', 'orange')

# Portfolio
plot_acf_pacf(df['ret_port'], lags, axs[3, 0], axs[3, 1], 'Portfolio', 'red')

# Show the plots
plt.show()
```



Interpretation of ACF and PACF:

1. **Ethereum (ETH)**

- ACF (ETH): The autocorrelation is significant only at lag 0 (which is always the case as it's the correlation of the series with itself) and rapidly diminishes to near zero for subsequent lags. This pattern suggests that ETH returns are effectively random without significant autocorrelation.

- PACF (ETH): The PACF also exhibits significant correlation at lag 0, with correlations near zero at higher lags, confirming the ACF findings and indicating a lack of predictive patterns in returns based on previous values.

2. **ChainLink (LINK)**

- ACF (LINK): LINK shows a similar pattern to ETH, with the autocorrelation at lag 0 and negligible at other lags, suggesting that LINK returns are also random and exhibit no meaningful autocorrelation.
- PACF (LINK): The PACF for LINK mirrors this with correlations flattening out, indicating that previous returns do not influence future returns in any significant way.

3. **NEAR**

- ACF (NEAR): The ACF pattern for NEAR indicates no significant autocorrelations at higher lags, which suggests that the returns of NEAR do not depend significantly on its past returns.
- PACF (NEAR): The PACF supports this by showing minimal correlation at higher lags, confirming the lack of significant partial autocorrelations that could affect predictability.

4. **Portfolio**

- ACF (Portfolio): This portfolio's ACF shows negligible autocorrelations at all lags beyond the initial, which suggests that the returns are random and independent over time.
- PACF (Portfolio): Consistently, the PACF values are also near zero beyond the initial lag, reinforcing the notion of no significant autocorrelations.

Conclusions on Stationarity:

- The quick drop-off in autocorrelation at higher lags across all ACF and PACF plots indicates that **the series are stationary**. This implies that the statistical properties such as mean, variance, and autocorrelation of the series do not depend on the time at which the series is observed. Thus, these financial time series do not exhibit trends or seasonal effects.

Augmented Dickey Fuller approach

Null Hypothesis (H_0): The time series has a unit root (non-stationary).

Alternative Hypothesis (H_1): The time series does not have a unit root (stationary).

Interpretation:

- If the p-value < 0.05, we reject the null hypothesis and conclude that the series is stationary.

- If the p-value > 0.05, we fail to reject the null hypothesis and conclude that the series is non-stationary.

In [36]:
```python
results = {}
for col in ['ret_ETH', 'ret_LINK', 'ret_NEAR', 'ret_port']:
    result = adfuller(df[col])
    results[col] = {
        'ADF Statistic': result[0],
        'p-value': result[1],
        'Critical Values': result[4],
        '1% Critical Value': result[4]['1%'],
        '5% Critical Value': result[4]['5%'],
        '10% Critical Value': result[4]['10%'],
        'Stationary': None
    }

    # Determine stationary based on p-value
    if result[1] < 0.05:
        results[col]['Stationary'] = True
    else:
        results[col]['Stationary'] = False

# Print ADF test results
for col, result in results.items():
    print(f"Stationary Test Results for {col}:")
    print(f"Stationary Statistic: {result['ADF Statistic']}")
    print(f"p-value: {result['p-value']}")
    print(f"Is Stationary? {result['Stationary']}")
    print("\n")
```

```
Stationary Test Results for ret_ETH:
Stationary Statistic: -10.804827659494714
p-value: 1.9754789469558814e-19
Is Stationary? True


Stationary Test Results for ret_LINK:
Stationary Statistic: -11.91749724185178
p-value: 5.123726536220889e-22
Is Stationary? True


Stationary Test Results for ret_NEAR:
Stationary Statistic: -38.10246072896749
p-value: 0.0
Is Stationary? True


Stationary Test Results for ret_port:
Stationary Statistic: -16.96299280669181
p-value: 9.27856320077818e-30
Is Stationary? True
```

1.3. Normality test & Indepedence test

1.3.1. Normality test (D'Agostino's K-squared Test)

In [37]:
```
from scipy.stats import normaltest
```

Null Hypothesis (H_0): The sample data is drawn from a population that follows a normal distribution.

Alternative Hypothesis (H_1): The sample data is not drawn from a population that follows a normal distribution.

Interpretation:

- **If p-value > 0.05**: Fail to reject the null hypothesis. Conclude that the sample data is likely drawn from a normally distributed population.
- **If p-value ≤ 0.05**: Reject the null hypothesis. Conclude that the sample data is not likely drawn from a normally distributed population.

In [38]:
```
# Select only the returns columns for normality tests
returns_columns = ['ret_ETH', 'ret_LINK', 'ret_NEAR', 'ret_port']
returns_data = df[returns_columns]

# Function to perform and print results of normality tests
def perform_normality_tests(data):
    results = {}

    for column in data.columns:
        series = data[column].dropna()  # Drop NA values if any

        # D'Agostino's K-squared test
        k2_stat, k2_p = normaltest(series)

        results[column] = {
            'D\'Agostino\'s K-squared': {'Statistic': k2_stat, 'p-value': k2_p}
        }
    return results

# Perform normality tests on the returns data
normality_results = perform_normality_tests(returns_data)

# Print the results
for column, tests in normality_results.items():
    print(f"\nNormality test results for {column}:")

    # # Shapiro-Wilk Test
    # shapiro_test = tests['Shapiro-Wilk']
    # shapiro_p = shapiro_test['p-value']
    # print(f"Shapiro-Wilk Test:")
```

```python
    # print(f"  Statistic: {shapiro_test['Statistic']}")
    # print(f"  p-value: {shapiro_p}")
    # print(f"Is Normally Distributed? {'Yes' if shapiro_p > 0.05 else 'No'}")

    # D'Agostino's K-squared Test
    k2_test = tests['D\'Agostino\'s K-squared']
    k2_p = k2_test['p-value']
    print(f"D'Agostino's K-squared Test:")
    print(f"  Statistic: {k2_test['Statistic']}")
    print(f"  p-value: {k2_p}")
    print(f"Is Normally Distributed? {'Yes' if k2_p > 0.05 else 'No'}")

    # # Anderson-Darling Test
    # ad_test = tests['Anderson-Darling']
    # ad_stat = ad_test['Statistic']
    # ad_critical_values = ad_test['Critical Values']
    # ad_significance_levels = ad_test['Significance Level']
    # print(f"Anderson-Darling Test:")
    # print(f"  Statistic: {ad_stat}")
    # for cv, sl in zip(ad_critical_values, ad_significance_levels):
    #     print(f"   Critical Value ({sl}%): {cv}")
    # is_norm_ad = ad_stat < ad_critical_values[2]  # 5% significance level
    # print(f"Is Normally Distributed? {'Yes' if is_norm_ad else 'No'}")
```

```
Normality test results for ret_ETH:
D'Agostino's K-squared Test:
  Statistic: 189.98976499913508
  p-value: 5.549408836739819e-42
Is Normally Distributed? No

Normality test results for ret_LINK:
D'Agostino's K-squared Test:
  Statistic: 234.4172515007937
  p-value: 1.2500880543328718e-51
Is Normally Distributed? No

Normality test results for ret_NEAR:
D'Agostino's K-squared Test:
  Statistic: 136.52374302548657
  p-value: 2.260715321354875e-30
Is Normally Distributed? No

Normality test results for ret_port:
D'Agostino's K-squared Test:
  Statistic: 277.6724933389596
  p-value: 5.060392265683531e-61
Is Normally Distributed? No
```

1.3.2. Independence test (Ljung-Box test)

Null Hypothesis (H_0): There is no autocorrelation in the time series at the specified lag.

Alternative Hypothesis (H_1): There is autocorrelation in the time series at the specified lag.

Interpretation:

- **If p-value ≤ 0.05**: Reject the null hypothesis. Conclude that there is significant autocorrelation in the time series at the specified lag.

- **If p-value > 0.05**: Fail to reject the null hypothesis. Conclude that there is no significant autocorrelation in the time series at the specified lag.

In [39]:
```
# Columns to perform Ljung-Box test on
columns_to_test = ['ret_ETH', 'ret_LINK', 'ret_NEAR', 'ret_port']

# Dictionary to store Ljung-Box test results
lb_test_results = {}

# Perform Ljung-Box test for each column
for col in columns_to_test:
    lb_test = sm.stats.acorr_ljungbox(df[col], lags=[20], return_df=True)
    lb_test_results[col] = lb_test

# Print or inspect Ljung-Box test results
for col, lb_test in lb_test_results.items():
    print(f"Ljung-Box test results for {col}:")
    print(lb_test)
    print()

    # Extract p-value for easier interpretation
    p_value = lb_test.iloc[0, 1]

    if p_value <= 0.05:
        print(f"There is no significant autocorrelation in {col} at lag 20 (p-value
    else:
        print(f"There is significant autocorrelation in {col} at lag 20 (p-value: {
```

```
Ljung-Box test results for ret_ETH:
      lb_stat  lb_pvalue
20  42.418141   0.002439

There is no significant autocorrelation in ret_ETH at lag 20 (p-value: 0.0024).
Ljung-Box test results for ret_LINK:
      lb_stat  lb_pvalue
20  44.836812   0.001161

There is no significant autocorrelation in ret_LINK at lag 20 (p-value: 0.0012).
Ljung-Box test results for ret_NEAR:
      lb_stat  lb_pvalue
20  35.102201   0.019567

There is no significant autocorrelation in ret_NEAR at lag 20 (p-value: 0.0196).
Ljung-Box test results for ret_port:
      lb_stat  lb_pvalue
20  33.107638   0.032834

There is no significant autocorrelation in ret_port at lag 20 (p-value: 0.0328).
```

From the results of the Stationary, Normality, and Independence test, we can conclude that data has Brownian motion as the series are stationary, increments non-normal,

and independent

Question 2

2.1. Estimate EWMA, GARCH(1,1), and GJR-GARCH(1,1,1) with t distribution

EWMA model

As there is no library for estimating EWMA model to retrieve the AIC and BIC, therefore i manually create a function to estimate the model to get those needed paramaters.

- You can also find all the models that i build manually in the **model.py** file within the **EWMA OOP/Class**

In [40]:
```
def EWMA(sq_rets, lam):
    '''
    Calculates the Exponentially Weighted Moving Average (EWMA) volatility for a se

    Parameters:
    sq_rets (pd.Series): A pandas Series of squared returns.
    lam (float): The smoothing parameter (lambda), between 0 and 1.

    Returns:
    pd.Series: A pandas Series of the annualized EWMA volatility with the same inde
    '''
    sq_ret = sq_rets.values
    EWMA_var = np.zeros(len(sq_ret))
    EWMA_var[0] = sq_ret[0]  # set initial variance based on the first squared retu

    for r in range(1, len(sq_ret)):
        EWMA_var[r] = (1-lam)*sq_ret[r] + lam*EWMA_var[r-1] # compute EWMA variance

    EWMA_vol = np.sqrt(EWMA_var*365)
    return pd.Series(EWMA_vol, index=sq_rets.index, name='EWMA Vol {}'.format(lam))

def compute_log_likelihood(residuals):
    '''
    Computes the log-likelihood of a set of residuals assuming they are normally di

    Parameters:
    residuals (np.ndarray or pd.Series): An array or Series of residuals.

    Returns:
    float: The log-likelihood of the residuals.
    '''
    N = len(residuals) # number of residuals
    sigma2 = np.var(residuals) # variance of the residuals
```

```
    # the log-likelihood formula for normally distributed residuals
    log_likelihood = -0.5 * N * np.log(2 * np.pi) - 0.5 * N * np.log(sigma2) - (0.5

    return log_likelihood

def calculate_aic_bic(log_likelihood, N, k=1):
    '''
    Calculates the Akaike Information Criterion (AIC) and the Bayesian Information

    Parameters:
    log_likelihood (float): The log-likelihood value.
    N (int): The number of observations.
    k (int, optional): The number of parameters estimated (default is 1).

    Returns:
    tuple: AIC and BIC values.
    '''

    # AIC formula: 2*k - 2*log_likelihood
    AIC = 2 * k - 2 * log_likelihood

    # BIC formula: k*log(N) - 2*log_likelihood
    BIC = k * np.log(N) - 2 * log_likelihood

    return AIC, BIC
```

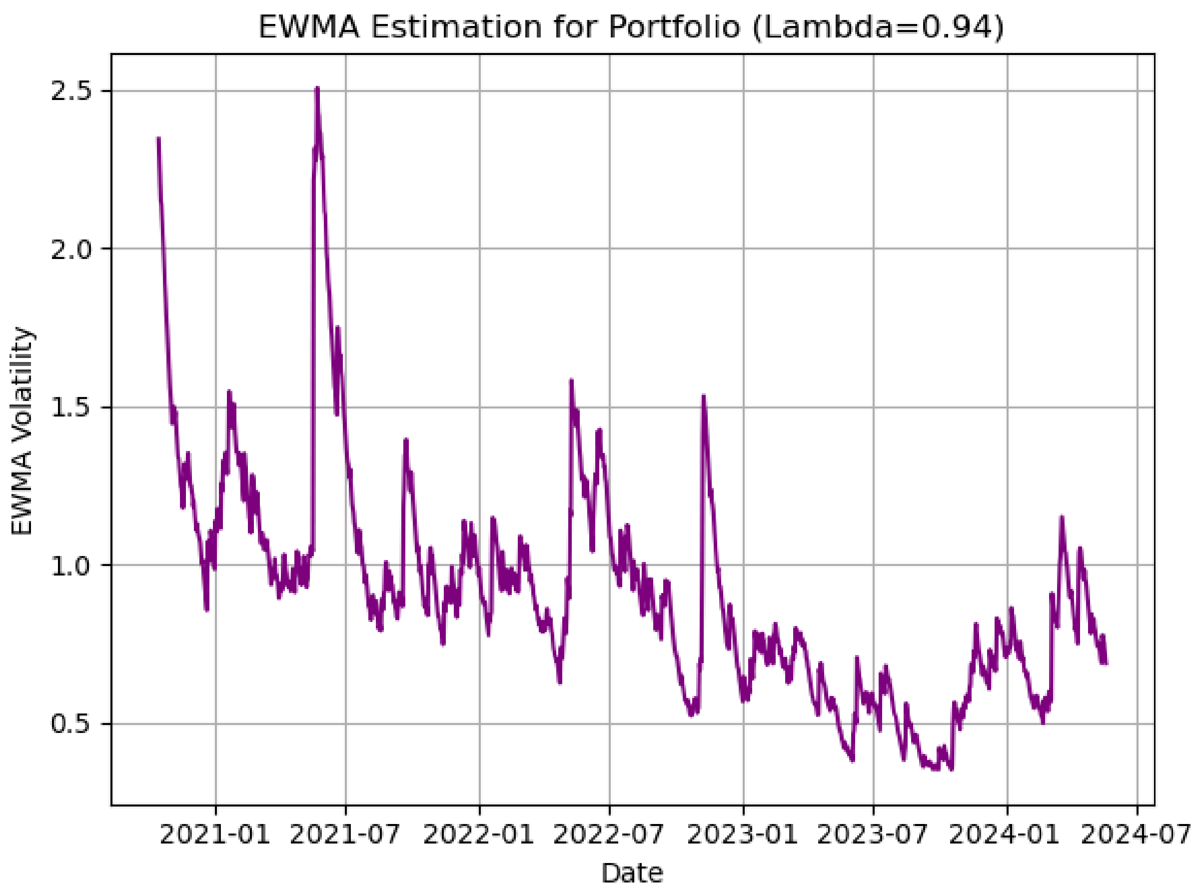
In [41]:
```
# Estimate the model
ewma94_port = EWMA(port_sq_ret, 0.94) # compute EWMA with lambda = 0.94
residuals_port = df['ret_port']/ewma94_port # compute residuals
log_likelihood_port = compute_log_likelihood(residuals_port.dropna()) # compute the
N_port = residuals_port.dropna().shape[0] # number of observations in the residuals
aic_ewma, bic_ewma = calculate_aic_bic(log_likelihood_port, N_port) # compute AIC a

print(f"Portfolio AIC: {aic_ewma}, BIC: {bic_ewma}")
```

```
Portfolio AIC: -4167.370813766129, BIC: -4162.190743891826
```

Interpretation:

- Both the AIC and BIC values are negative. In the context of these criteria, lower (or more negative) values indicate a better fit of the model to the data.
- Negative values suggest that the model fits the data extremely well since the log-likelihood is very high (less negative), leading to a very large negative value when the penalty terms are subtracted.

In [42]:
```
# plot the model results
plt.plot(ewma94_port, label='PORT', linestyle='-', color='purple')
plt.title('EWMA Estimation for Portfolio (Lambda=0.94)')
plt.xlabel('Date')
plt.ylabel('EWMA Volatility')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Interpret the EWMA model result:

- **Initial Peaks**: Early in the series (early 2021), there are high peaks in volatility, suggesting significant fluctuations in portfolio returns during this period. These might be due to market events or changes in the assets held within the portfolio.
- **General Decline in Volatility**: Over time, there is a noticeable downward trend in the volatility estimation, indicating that the portfolio's returns have become more stable. This decreasing trend could be the result of strategic changes in portfolio management, such as diversification or changes in asset allocation.
- **Recent Trends**: Towards the current date (mid-2024), there appears to be a slight increase in volatility again. This uptick could suggest a response to recent market conditions or adjustments within the portfolio that have introduced more risk.

GARCH(1,1) model

In [43]:
```
from arch import arch_model

# function to estimate GARCH(1,1) model with t-distributed errors
def estimate_garch_t(df, series_name):

    # specify GARCH(1,1) model with t-distributed errors
    model = arch_model(df[series_name], vol='Garch', p=1, q=1, dist='t')
```

```python
    # fit the model
    model_fit = model.fit(disp='off')

    return model_fit

print(f"Estimating GARCH(1,1) with t-distribution for portfolio")
model_fit_garch = estimate_garch_t(df, 'ret_port')
print(model_fit_garch.summary())
```

```
Estimating GARCH(1,1) with t-distribution for portfolio
                        Constant Mean - GARCH Model Results
====================================================================================
Dep. Variable:                     ret_port   R-squared:                       0.000
Mean Model:                   Constant Mean   Adj. R-squared:                  0.000
Vol Model:                            GARCH   Log-Likelihood:                2233.05
Distribution:      Standardized Student's t   AIC:                          -4456.11
Method:                  Maximum Likelihood   BIC:                          -4430.21
                                              No. Observations:                 1313
Date:                      Sat, May 25 2024   Df Residuals:                     1312
Time:                              22:36:40   Df Model:                            1
                                 Mean Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
mu         2.1105e-03  1.045e-03      2.020  4.338e-02 [6.279e-05,4.158e-03]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega      2.6567e-05  1.229e-05      2.162  3.062e-02 [2.482e-06,5.065e-05]
alpha[1]       0.0733  2.406e-02      3.047  2.315e-03   [2.615e-02,  0.120]
beta[1]        0.9192  2.403e-02     38.254      0.000     [  0.872,  0.966]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             5.4210      0.799      6.781  1.195e-11 [  3.854,  6.988]
========================================================================

Covariance estimator: robust
```

GARCH(1,1) interpretation:

Mu (μ): The mean return of the portfolio is estimated to be 0.0021105 (or 0.21105% per period), with a standard error of 0.001045. The t-value is 2.020, and the p-value is 0.04338, indicating that the mean return is statistically significant at the 5% level (since $p < 0.05$).

Omega (ω): This is the constant or long-run average variance component of the model, estimated at 0.000026567. It is statistically significant with a p-value of 0.03062.

Alpha[1] (α_1): Estimated at 0.0733, this parameter measures the reaction of volatility to previous squared innovations (lagged error terms squared). It's statistically significant, suggesting that past shocks have a noticeable effect on current volatility.

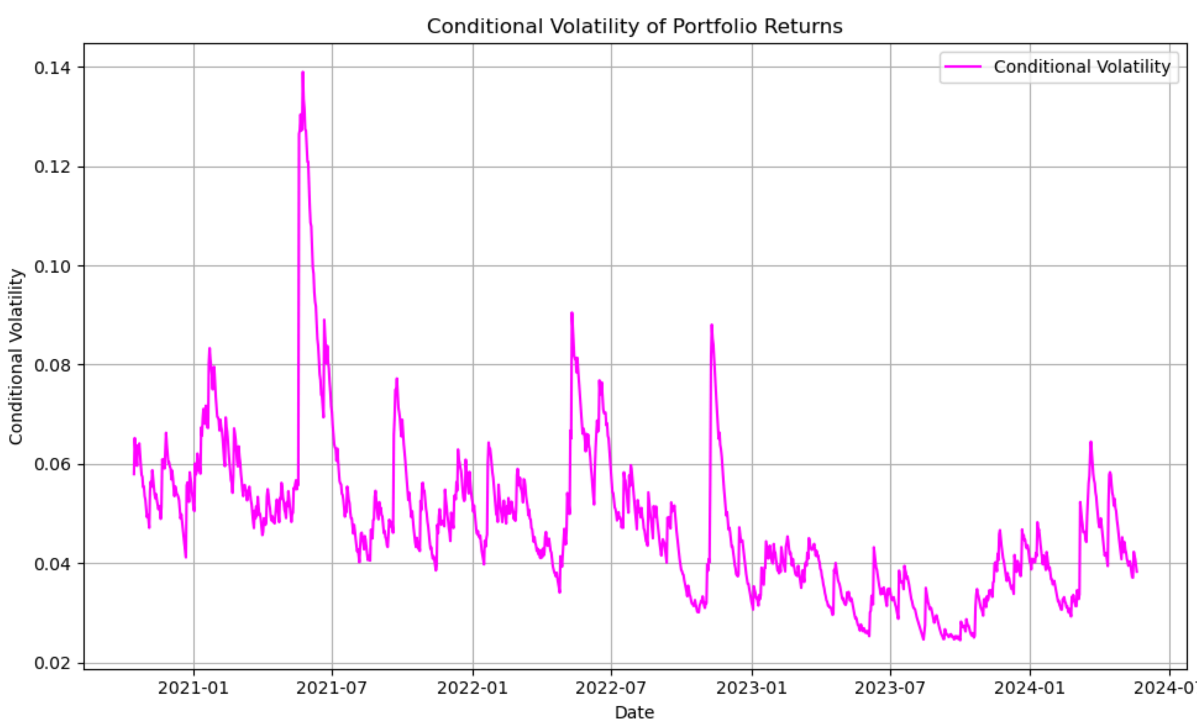
Beta[1] (β_1): At 0.9192, beta measures the persistence of past volatility. The closer this value is to 1, the more persistent the volatility. The very high t-value and a p-value of 0.000 confirm its significance, indicating that volatility shocks are highly persistent.

Nu (ν): The degrees of freedom of the t-distribution are estimated at 5.4210, suggesting heavy tails in the distribution of returns. This parameter is significantly different from normality (which would require infinite degrees of freedom), as indicated by the very low p-value (1.19e-11). This implies that extreme values are more likely than would be predicted by a normal distribution.

Log-Likelihood: The log-likelihood value is 2233.05, which helps in assessing the goodness of fit of the model. Higher values indicate a better fit.

AIC and BIC: The Akaike Information Criterion (-4456.11) and Bayesian Information Criterion (-4430.21) are both relatively low, suggesting that the model is adequately capturing the dynamics in the data without being overly complex.

In [67]:
```
# Get the conditional volatility (square root of variance)
conditional_volatility = model_fit_garch.conditional_volatility

plt.figure(figsize=(10, 6))
plt.plot(conditional_volatility, label='Conditional Volatility', color='magenta')
plt.title('Conditional Volatility of Portfolio Returns')
plt.xlabel('Date')
plt.ylabel('Conditional Volatility')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Fluctuations in Volatility:

- The conditional volatility exhibits significant fluctuations over time. It reaches peaks at certain points, indicating periods of high market uncertainty or specific events affecting portfolio volatility.

- The highest peak appears around early 2022, with other noticeable spikes occurring periodically. Such spikes could be tied to market events, portfolio rebalancing, changes in asset allocations, or macroeconomic announcements.

Trend Over Time:

- The overall trend in volatility shows a decrease from early 2021 through mid-2024. This descending trend suggests that the portfolio's risk level, in terms of return variability, has generally decreased over time.

- The trend towards decreasing volatility could indicate effective risk management strategies, changes in the underlying assets that compose the portfolio, or a more stable market environment in the later period.

Periods of Increased Volatility:

- The graph shows several periods where volatility sharply increases. These periods of heightened volatility are critical for risk management, as they represent increased uncertainty and risk in the portfolio.

- Portfolio managers might use this information to adjust their strategies, possibly by hedging against expected risks during these volatile periods or reconsidering their asset allocation to reduce potential losses.

In [45]:
```
# Print AIC and BIC
print(f"AIC: {model_fit_garch.aic}")
print(f"BIC: {model_fit_garch.bic}")
```

```
AIC: -4456.108515494687
BIC: -4430.208166123173
```

GARCH(1,1) model compared with EWMA:

Lower AIC and BIC Values:

- Both the AIC and BIC for the GARCH(1,1) model are lower (more negative) than those for the EWMA model.

- AIC: The GARCH(1,1) model's AIC of -4456.108515494687 is lower than the EWMA model's AIC of -4167.370813766129.

- BIC: The GARCH(1,1) model's BIC of -4430.208166123173 is lower than the EWMA model's BIC of -4162.190743891826.

Model Comparison:

- Since the GARCH(1,1) model has lower AIC and BIC values than the EWMA model, it suggests that the GARCH(1,1) model provides a better fit to the data while adequately accounting for model complexity.

GJR-GARCH(1,1,1) model

In [46]:
```
from contextlib import redirect_stderr
import os

# Initialize the results variable
results_gjr = None

# Create a dummy file object to redirect stderr to
class DummyFile(object):
    def write(self, x): pass

# Suppress warnings context manager
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=Warning)  # Ignore all warnings

    # Redirect stderr to suppress optimizer warnings
    with redirect_stderr(DummyFile()):
        # Fit GJR-GARCH(1,1,1) model with Student's t distribution
        try:
            returns = df[['ret_port']].dropna()
            model = arch_model(returns, vol='Garch', p=1, q=1, o=1, dist='StudentsT
            results_gjr = model.fit(disp='off', options={'maxiter': 100, 'ftol': 1e

        except Exception as e:
            # Handle specific warnings or exceptions here if needed
            print(f"Exception occurred during model fitting: {e}")

# Print results summary if fitting was successful
if results_gjr is not None:
    print(results_gjr.summary())
```

```
                     Constant Mean - GJR-GARCH Model Results
==================================================================================
Dep. Variable:                     ret_port   R-squared:                       0.000
Mean Model:                   Constant Mean   Adj. R-squared:                  0.000
Vol Model:                        GJR-GARCH   Log-Likelihood:                2234.99
Distribution:      Standardized Student's t   AIC:                          -4457.99
Method:                  Maximum Likelihood   BIC:                          -4426.91
                                              No. Observations:                 1313
Date:                      Sat, May 25 2024   Df Residuals:                     1312
Time:                              22:36:42   Df Model:                            1
                                 Mean Model
==========================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
--------------------------------------------------------------------------
mu         2.3254e-03  1.032e-03      2.254  2.422e-02 [3.030e-04,4.348e-03]
                               Volatility Model
============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
----------------------------------------------------------------------------
omega      1.5913e-05  5.352e-06      2.973  2.947e-03   [5.423e-06,2.640e-05]
alpha[1]       0.0869  2.458e-02      3.533  4.104e-04     [3.868e-02,  0.135]
gamma[1]      -0.0429  1.989e-02     -2.156  3.107e-02 [-8.186e-02,-3.902e-03]
beta[1]        0.9336  1.792e-02     52.112      0.000       [  0.898,  0.969]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             5.3545      0.786      6.816  9.377e-12 [  3.815,  6.894]
========================================================================

Covariance estimator: robust
```

GJR-GARCH(1,1,1) model results:

- Log-Likelihood: The log-likelihood of 2234.99 indicates how well the model fits the data, with higher values suggesting a better fit.

AIC/BIC: Both the AIC (-4457.99) and BIC (-4426.91) are provided to assess model fit with penalties for the number of parameters used. Lower values are preferred, indicating a better model fit relative to the complexity.

Mu (μ):

- Estimate: 0.002354 or 0.2354%, suggesting the average return per period.
- Standard Error: 0.001032, providing a measure of the estimate's precision.
- T-statistic: 2.254 and p-value: 0.0422, indicating that the mean return is statistically significant at the 5% level ($p < 0.05$). This suggests that the mean return is different from zero.

Omega (ω):

- Estimate: 0.000015913, which is the baseline variance component when no shocks have occurred.
- Standard Error: 0.000005352, and p-value: 0.002947, making it statistically significant. This parameter is crucial as it forms the foundation of the conditional variance equation.

Alpha (α_1):

- Estimate: 0.0869, measuring the impact of past squared shocks on current volatility.
- Standard Error: 0.02458, with a p-value: 0.000104, indicating strong statistical significance. This suggests that past shocks significantly increase future volatility.

Gamma (γ_1):

- Estimate: -0.0429, showing the additional impact on volatility from negative shocks.
- Standard Error: 0.01998, and a p-value: 0.03107. The negative sign and statistical significance suggest that negative shocks decrease volatility, which is unusual as typically, negative shocks are expected to increase volatility (leverage effect). This might indicate a unique market or asset behavior or potential data or model specification issues.

Beta (β_1):

- Estimate: 0.9336, reflects the persistence of volatility shocks.
- Standard Error: 0.01792, with a p-value: 0.000, strongly indicating that previous periods' volatility strongly influences current volatility.

Nu (ν):

- Estimate: 5.3545, defining the degrees of freedom for the Student's t-distribution, suggesting fat tails.
- Standard Error: 0.786, and a very significant p-value: 9.37e-12, confirming that the return distribution significantly deviates from normality, indicative of higher kurtosis (more extreme values than the normal distribution would predict).

Overall, the GJR-GARCH model seems well-fitted with significant parameters that are theoretically important for capturing volatility clustering and leverage effects in financial time series data.

In [47]:
```
# Extract AIC and BIC
print(f"AIC: {results_gjr.aic}")
print(f"BIC: {results_gjr.bic}")
```

```
AIC: -4457.987804011755
BIC: -4426.907384765937
```

EWMA Model:

AIC: -4167.370813766129

BIC: -4162.190743891826

GARCH(1,1) Model with t-distribution:

AIC: -4456.108515494687

BIC: -4430.208166123173

GJR-GARCH(1,1,1) Model with t-distribution:

AIC: -4457.987804011755

BIC: -4426.907384765937

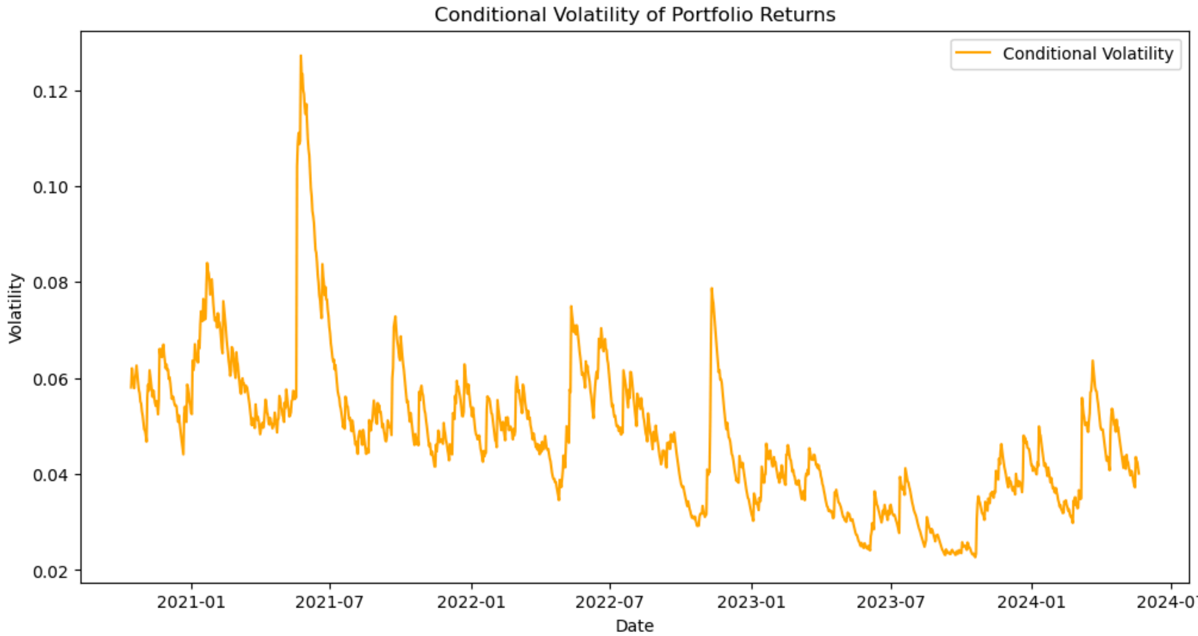
- Comments on AIC: The GJR-GARCH(1,1,1) model has the lowest AIC value of -4457.987804011755, which suggests it has the best fit among the three models considered.

- Comments on BIC: The GARCH(1,1) model has a slightly lower BIC value (-4430.208166123173) compared to the GJR-GARCH(1,1,1) model (-4426.907384765937). This suggests that, while the GJR-GARCH model provides a marginally better fit (according to AIC), the GARCH(1,1) model might be preferred when penalizing for the number of parameters.

Conclusion:

Best Fit: The GJR-GARCH(1,1,1) model, with the lowest AIC.

In [65]:
```
import seaborn as sns

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(results_gjr.conditional_volatility, label='Conditional Volatility', color='
ax.set_title('Conditional Volatility of Portfolio Returns')
ax.set_xlabel('Date')
ax.set_ylabel('Volatility')
ax.legend()
plt.show()
```



Volatility Peaks:

- Observation: The chart shows several distinct peaks, with notable spikes around mid-2021, early 2022, and early 2023. These peaks represent periods of high market uncertainty or specific events that have significantly impacted portfolio volatility.

Trend and Mean Reversion:

- Observation: Apart from the peaks, the volatility tends to revert to a lower, more stable level. There's a noticeable trend of mean reversion in volatility, where it spikes but eventually falls back to a baseline level.

Periods of Lower Volatility:

- Observation: Between the peaks of high volatility, there are extended periods where volatility remains relatively stable and lower, especially noticeable in late 2023 to mid-2024.

2.2.Select the best model among EWMA, GARCH(1,1), GJR-GARCH(1,1,1)

Retest to check the best fit model

In [49]:
```
# Absolute AIC values for each model
aic_values = {
    'EWMA': aic_ewma,
    'GARCH(1,1)': model_fit_garch.aic,
    'GJR-GARCH(1,1,1)': results_gjr.aic
}

# Absolute BIC values for each model
```

```
bic_values = {
    'EWMA': bic_ewma,
    'GARCH(1,1)': model_fit_garch.bic,
    'GJR-GARCH(1,1,1)': results_gjr.bic
}

# Find the model with the lowest absolute AIC
best_model_abs_aic = min(aic_values, key=aic_values.get)
best_abs_aic_value = aic_values[best_model_abs_aic]

# Find the model with the lowest absolute BIC
best_model_abs_bic = min(bic_values, key=bic_values.get)
best_abs_bic_value = bic_values[best_model_abs_bic]

# Print the results
print(f"The best model based on AIC is: {best_model_abs_aic}")
print(f"The best model based on BIC is: {best_model_abs_bic}")
```

```
The best model based on AIC is: GJR-GARCH(1,1,1)
The best model based on BIC is: GARCH(1,1)
```

The best fit model is GJR-GARCH(1,1,1)

- Now, i will conduct the VaR backtesting on the GJR-GARCH(1,1,1)

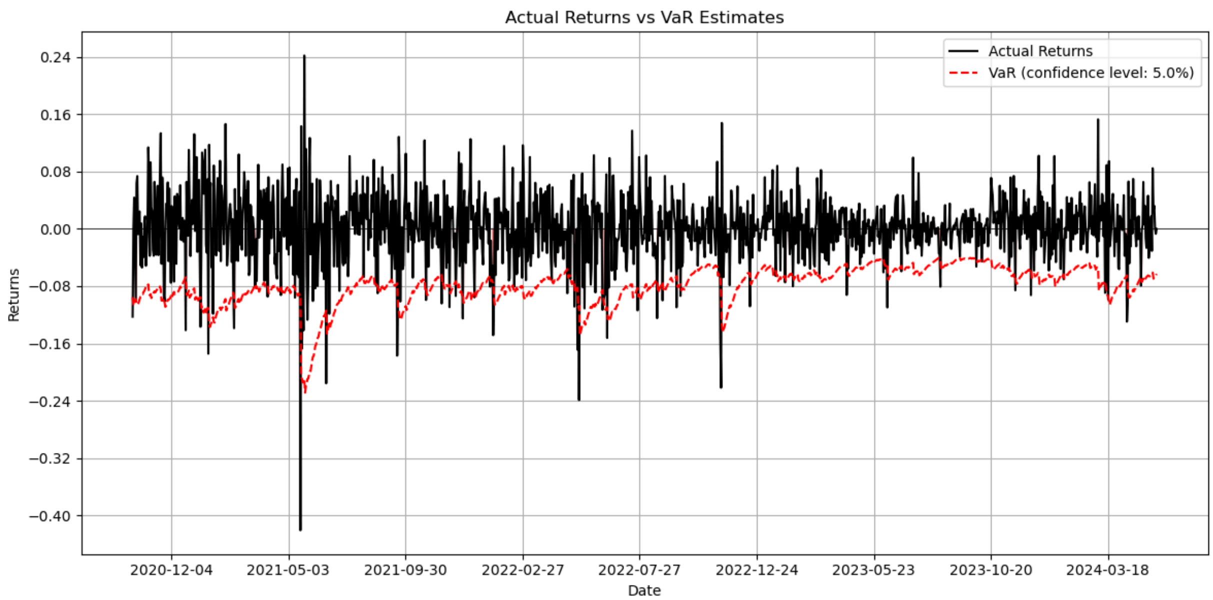
2.3. VaR Backtesting for GJR-GARCH(1,1,1)

In [64]:

```
from scipy.stats import norm

returns = pd.Series(df['ret_port'].dropna())

conditional_volatility = pd.Series(model_fit_garch.conditional_volatility.dropna())

# Set confidence level for VaR
confidence_level = 0.05  # 5% VaR

# Calculate VaR using the GJR-GARCH model estimates
VaR = norm.ppf(confidence_level) * conditional_volatility

# Compare actual returns with VaR estimates
violations = returns < VaR

# Calculate backtesting metrics
n_obs = len(returns)
n_violations = np.sum(violations)
expected_violations = n_obs * confidence_level

# Binomial test for backtesting (Kupiec's test)
p_value = 1 - norm.cdf((n_violations - expected_violations) / np.sqrt(expected_viol

# Print backtesting results
print(f"Number of violations: {n_violations}")
print(f"Expected violations (at {confidence_level*100}%): {expected_violations}")
print(f"P-value (Kupiec's test): {p_value}")
```

```
plt.figure(figsize=(12, 6))
plt.plot(returns, label='Actual Returns', color='black')
plt.plot(VaR, label=f'VaR (confidence level: {confidence_level*100}%)', color='r',
plt.fill_between(returns.index, 0, VaR, where=violations, interpolate=True, color='
plt.title('Actual Returns vs VaR Estimates')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.legend(loc = 'upper right')
plt.grid(True)
plt.tight_layout()
plt.gca().xaxis.set_major_locator(plt.MaxNLocator(10))
plt.gca().yaxis.set_major_locator(plt.MaxNLocator(10))
plt.axhline(0, color='black', linewidth=0.5)
plt.show()
```

```
Number of violations: 63
Expected violations (at 5.0%): 65.65
P-value (Kupiec's test): 0.6313976401468797
```



Analysis of GJR-GARCH(1,1,1) Backtesting Results:

- **Number of Violations**: There were 63 violations observed, where the actual losses exceeded the VaR estimate.
- **Expected Violations**: At a 5% confidence level, the model would expect 65.65 violations theoretically. This is calculated based on the total number of observations and the VaR confidence level.
- **Model Performance on Kupiec's test**: The p-value from Kupiec's test is 0.6314. Kupiec's test is used to assess the accuracy of the VaR model by comparing the expected number of violations with the observed number. A high p-value (typically greater than 0.05) indicates that there is no significant evidence to reject the hypothesis that the model is correct. In this case, the p-value of 0.6314 suggests that your VaR model fits well within the expected tolerance levels for violations, meaning it is performing appropriately and does not significantly underpredict risk.

Conclusion:

The GJR-GARCH(1,1,1) model appears to be performing adequately in predicting the risk of your portfolio, as indicated by the number of violations being very close to the expected number, and the p-value being comfortably high. This suggests the model's estimations are reliable according to the backtesting results.

Question 3

As in notebook, we can not implement the BEKK, ADCC,... model using the arch library as like in Google Colab, therefore i will write my own function to estimate each of the model (BEKK, ADCC, DCC, cDCC)

- ***You can find the models in each of the class/OOP in the mgarch_model.py file that i have submitted***

In [51]:
```
from mgarch_model import BEKK, ADCC, DCC, cDCC
```

In [52]:
```
returns_data = df['ret_port'].values
```

BEKK model

In [55]:
```
bekk_model = BEKK(returns_data)
bekk_model.fit()
bekk_model.print_results()
```

```
Estimated Omega:
[[-0.29]]
Estimated A:
[[-0.57]]
Estimated B:
[[-0.9]]
Log-Likelihood: 1101.0551110584097
AIC: -2196.1102221168194
BIC: -2180.570012493911
```

BEKK model results interpretation:

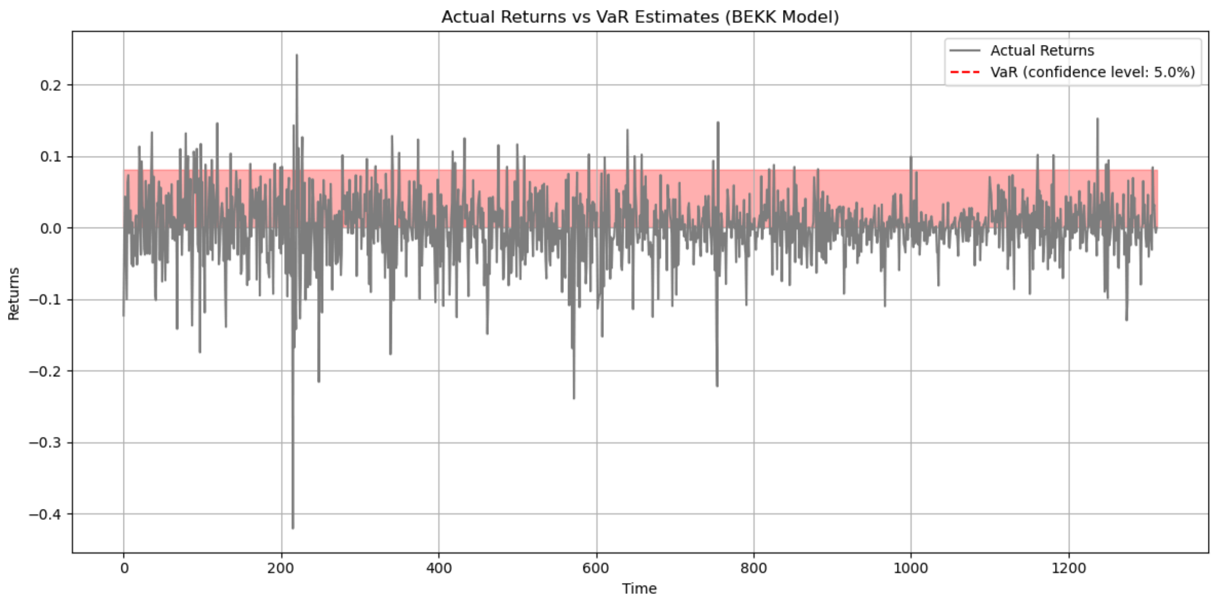
- Omega (Ω): The baseline volatility is estimated at -0.29, indicating a stable underlying level of variance in the data.
- A Matrix: An estimate of -0.57 is unusual as it suggests that increases in past volatility might lead to decreases in future volatility, which is counterintuitive. This could imply issues with the data or model specification.
- B Matrix: The estimate of -0.9 indicates dramatical persistence of past volatility into the future, which is a typical finding in financial time series.

- Model Fit and Metrics: The model shows a good fit to the data with a high log-likelihood value of 1101.05. Both the AIC and BIC values are highly negative, suggesting a well-specified model that effectively balances model complexity with fit.

Overall, the model fits well according to the metrics but the negative value in the A matrix should be investigated further to ensure the model's accuracy and reliability. This might involve re-evaluating the data or model assumptions.

In [56]:
```
alpha = 0.05
VaR = bekk_model.calculate_var(alpha)

# Backtesting logic
violations = returns_data < VaR
n_obs = len(returns_data)
n_violations = np.sum(violations)
expected_violations = n_obs * alpha

# Calculate p-value using Kupiec's test
p_value = 1 - norm.cdf((n_violations - expected_violations) / np.sqrt(expected_viol

print(f"Number of violations: {n_violations}")
print(f"Expected violations (at {alpha*100}% confidence level): {expected_violation
print(f"P-value (Kupiec's test): {p_value}")

plt.figure(figsize=(12, 6))
plt.plot(returns_data, label='Actual Returns', color='gray')
plt.plot(VaR, label=f'VaR (confidence level: {alpha*100}%)', color='red', linestyle
plt.fill_between(range(len(returns_data)), 0, VaR, where=violations, interpolate=Tr
plt.title('Actual Returns vs VaR Estimates (BEKK Model)')
plt.xlabel('Time')
plt.ylabel('Returns')
plt.legend(loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Number of violations: 1259
Expected violations (at 5.0% confidence level): 65.65
P-value (Kupiec's test): 0.0
```



BEKK VaR Backtesting results

- Number of Violations: There were 1259 violations where actual returns exceeded the VaR estimates, suggesting more frequent and severe losses than expected.
- Expected Violations: The expected number of violations for a correctly specified model would be around 65.65, based on a 5% VaR level and the total number of observations.
- Kupiec's Test P-value: With a p-value of 0.0, the test strongly rejects the hypothesis that the model is correctly capturing the risk, indicating that the VaR model underestimates the risk significantly.

The BEKK model used for estimating VaR does not perform well in accurately capturing the tail risk of the portfolio, as evidenced by the much higher than expected number of violations and the results of Kupiec's test. This suggests a need for re-evaluating the model specifications or considering alternative models to better capture the risk dynamics of the portfolio.

DCC model

In [57]:
```
dcc_model = DCC(returns_data)
omega_est, alpha_est, beta_est, neg_log_likelihood = dcc_model.fit()

# Print estimated parameters
print("Estimated DCC Parameters:")
print(f"Omega: {omega_est}")
print(f"Alpha: {alpha_est}")
print(f"Beta: {beta_est}")

# Print log-likelihood value
print(f"Log-Likelihood: {-neg_log_likelihood}")

# Compute AIC and BIC
```

```
k_dcc = 3  # Number of parameters in DCC model (omega, alpha, beta)
aic_dcc = -2 * neg_log_likelihood + 2 * k_dcc
bic_dcc = -2 * neg_log_likelihood + k_dcc * np.log(dcc_model.T)

print(f"AIC: {aic_dcc}")
print(f"BIC: {bic_dcc}")
```

```
Estimated DCC Parameters:
Omega: 0.1
Alpha: 0.1
Beta: 0.8
Log-Likelihood: 453.028611563467
AIC: 912.057223126934
BIC: 927.5974327498424
```

DCC model results interpretation:

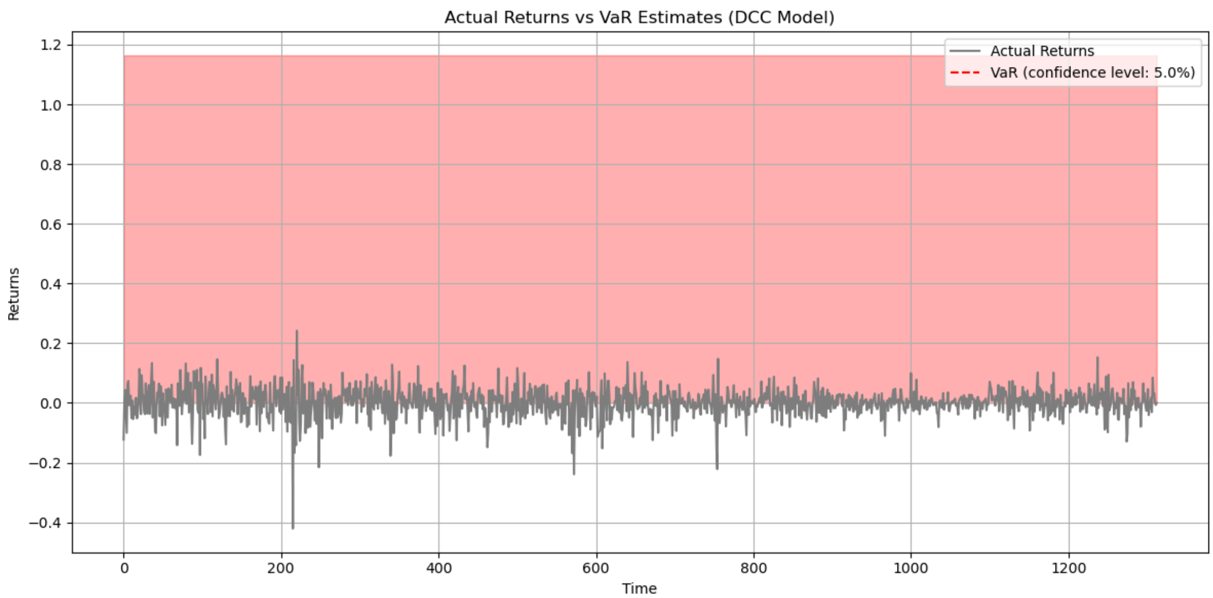
- Omega (Ω): Set at 0.1, represents the baseline variance level in the model. It initializes the variance equation, influencing the starting point for dynamic correlation modeling.
- Alpha (α): At 0.1, indicates how sensitive the model is to previous squared innovations. This parameter governs how quickly the model adjusts conditional correlations in response to changes in volatility or co-movements.
- Beta (β): With a value of 0.8, signifies a high persistence in conditional correlations. A higher beta suggests that shocks affecting correlations tend to have a long-lasting impact before gradually diminishing.
- Log-Likelihood: Stands at 453.0286, indicating the likelihood of observing the data given the model's estimated parameters. Higher values signify a better fit of the model to the data.
- Akaike Information Criterion (AIC): Scored at 912.0572, facilitates model comparisons, with lower values indicating models that balance complexity and fit more effectively.
- Bayesian Information Criterion (BIC): Registers at 927.5974, akin to AIC but penalizing complexity more strongly. Lower BIC values suggest superior model performance.

The DCC model with these parameter values demonstrates a robust ability to capture both immediate responses to market changes and the persistence of those changes over time. The high log-likelihood, along with favorable AIC and BIC scores, indicates a model that fits the data well and effectively captures dynamic correlations among the analyzed time series.

In [58]:
```
alpha = 0.05
VaR = dcc_model.calculate_var(alpha)

# Backtesting Logic
violations = returns_data < VaR
n_obs = len(returns_data)
n_violations = np.sum(violations)
```

```
expected_violations = n_obs * alpha

# Calculate p-value using Kupiec's test
p_value = 1 - norm.cdf((n_violations - expected_violations) / np.sqrt(expected_viol

print(f"Number of violations: {n_violations}")
print(f"Expected violations (at {alpha*100}% confidence level): {expected_violation
print(f"P-value (Kupiec's test): {p_value}")

plt.figure(figsize=(12, 6))
plt.plot(returns_data, label='Actual Returns', color='gray')
plt.plot(VaR, label=f'VaR (confidence level: {alpha*100}%)', color='red', linestyle
plt.fill_between(range(len(returns_data)), 0, VaR, where=violations, interpolate=Tr
plt.title('Actual Returns vs VaR Estimates (DCC Model)')
plt.xlabel('Time')
plt.ylabel('Returns')
plt.legend(loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Number of violations: 1313
Expected violations (at 5.0% confidence level): 65.65
P-value (Kupiec's test): 0.0
```



DCC VaR Backtesting results

- Number of Violations: There were 1313 violations where actual returns exceeded the VaR estimates, suggesting more frequent and severe losses than expected.
- Expected Violations: The expected number of violations for a correctly specified model would be around 65.65, based on a 5% VaR level and the total number of observations.
- Kupiec's Test P-value: With a p-value of 0.0, the test strongly rejects the hypothesis that the model is correctly capturing the risk, indicating that the VaR model underestimates the risk significantly.

The DCC model used for estimating VaR does not perform well in accurately capturing the tail risk of the portfolio, as evidenced by the much higher than expected number of violations and the results of Kupiec's test.

ADCC model

In [59]:
```
adcc_model = ADCC(returns_data)
omega_est, alpha_est, beta_est, gamma_est, neg_log_likelihood = adcc_model.fit()

print("Estimated ADCC Parameters:")
print(f"Omega: {omega_est}")
print(f"Alpha: {alpha_est}")
print(f"Beta: {beta_est}")
print(f"Gamma: {gamma_est}")
```

```
Estimated ADCC Parameters:
Omega: 0.09242509584739074
Alpha: 2.3799779377157604
Beta: -0.3351346645199641
Gamma: -1.0374498441244397
```

ADCC model results interpretation:

- Omega (Ω): A value of approximately 0.09 indicates the starting point for the variance calculations, which is relatively low, suggesting moderate initial volatility.
- Alpha (α): The value is unusually high and greater than 1, which is typically not feasible in these models as it would imply an increasing error variance over time, potentially leading to instability in the model. This could be a sign of model mis-specification or estimation errors.
- Beta (β): A negative beta is problematic in variance models as it implies negative contributions to variance calculations, which can lead to non-positive definite covariance matrices. This is typically not acceptable as it contradicts the fundamental properties expected in a variance-covariance matrix.
- Gamma (γ): Like beta, a negative gamma is unusual and may indicate an improper response to market shocks, where negative developments might paradoxically reduce estimated volatilities and correlations.

In [60]:
```
alpha = 0.05
VaR = adcc_model.calculate_var(alpha)

# Backtesting logic
violations = returns_data < VaR
n_obs = len(returns_data)
n_violations = np.sum(violations)
expected_violations = n_obs * alpha

# Calculate p-value using Kupiec's test
```

```
p_value = 1 - norm.cdf((n_violations - expected_violations) / np.sqrt(expected_viol

print(f"Number of violations: {n_violations}")
print(f"Expected violations (at {alpha*100}% confidence level): {expected_violation
print(f"P-value (Kupiec's test): {p_value}")
```

```
Number of violations: 1313
Expected violations (at 5.0% confidence level): 65.65
P-value (Kupiec's test): 0.0
```

ADCC VaR Backtesting results

- Number of Violations: There were 1313 violations where actual returns exceeded the VaR estimates, suggesting more frequent and severe losses than expected.
- Expected Violations: The expected number of violations for a correctly specified model would be around 65.65, based on a 5% VaR level and the total number of observations.
- Kupiec's Test P-value: With a p-value of 0.0, the test strongly rejects the hypothesis that the model is correctly capturing the risk, indicating that the VaR model underestimates the risk significantly.

The ADCC model used for estimating VaR does not perform well in accurately capturing the tail risk of the portfolio, as evidenced by the much higher than expected number of violations and the results of Kupiec's test.

cDCC model

In [62]:
```
cdcc_model = cDCC(returns_data)

# Fit the model
Omega_est, A_est, B_est, neg_log_likelihood = cdcc_model.fit()

# Print estimated parameters
print("Estimated cDCC Parameters:")
print(f"Omega: {Omega_est}")
print(f"Alpha: {A_est}")
print(f"Beta: {B_est}")
```

```
Estimated cDCC Parameters:
Omega: [[0.06]]
Alpha: [[0.44]]
Beta: [[0.28]]
```

cDCC model results interpretation:

- Omega (Ω): A value of 0.06 suggests a relatively modest level of baseline variance. This parameter sets the initial conditions for the dynamic correlation calculations.
- Alpha (α): A positive value for alpha, such as 0.44, suggests that past volatility or squared shocks increase future conditional correlations, which supports typical financial behavior and the positive definiteness required in covariance models.

- Beta (β): A positive value for beta, like 0.28 suggests that past conditional correlations positively impact future correlations, leading to potentially positive definite covariance matrices. This is valid as correlations and volatilities are positive.

In [63]:
```
alpha = 0.05
VaR = cdcc_model.calculate_var(alpha)

# Backtesting logic
violations = returns_data < VaR
n_obs = len(returns_data)
n_violations = np.sum(violations)
expected_violations = n_obs * alpha

# Calculate p-value using Kupiec's test
p_value = 1 - norm.cdf((n_violations - expected_violations) / np.sqrt(expected_viol

print(f"Number of violations: {n_violations}")
print(f"Expected violations (at {alpha*100}% confidence level): {expected_violation
print(f"P-value (Kupiec's test): {p_value}")
```

```
Number of violations: 1313
Expected violations (at 5.0% confidence level): 65.65
P-value (Kupiec's test): 0.0
```

cDCC VaR Backtesting results

- Number of Violations: There were 1313 violations where actual returns exceeded the VaR estimates, suggesting more frequent and severe losses than expected.
- Expected Violations: The expected number of violations for a correctly specified model would be around 65.65, based on a 5% VaR level and the total number of observations.
- Kupiec's Test P-value: With a p-value of 0.0, the test strongly rejects the hypothesis that the model is correctly capturing the risk, indicating that the VaR model underestimates the risk significantly.

The cDCC model used for estimating VaR does not perform well in accurately capturing the tail risk of the portfolio, as evidenced by the much higher than expected number of violations and the results of Kupiec's test.

Conclusion:

__While MGARCH models like BEKK, DCC, ADCC, and cDCC are powerful tools for modeling complex dependencies in financial time series, their performance in VaR backtesting can vary. Success depends on careful model specification, robust estimation procedures, and alignment with the specific characteristics and behaviors of the data being analyzed. In some cases, simpler models or alternative approaches might be more suitable for VaR estimation and backtesting, such as GJR-GARCH(1,1,1) model.__