MFE 409: Financial Risk Management
Problem set 4

Valentin Haddad

due 4/28 before midnight

You should work with your assigned group but should write up your answer individually. Give the name of your group members in your writeup and submit it on BruinLearn before April 28 at midnight.

Individual Submission

Vikalp Thukral | UID: **406534669**
Part of **Cohort 1, Group 8**

---

1 Choosing a VaR technique

Download the excel file which contains the time series of gains for a strategy from 1/2/2014 to 12/19/2017.

1. *Historical method*
    a. For each day in 2015-2017, compute historical VaR and exponential weighted 1-day 99%-VaR (with $\lambda = 0.995$).
    b. Backtest the measures for VaR you obtained in question 1. How many exceptions did the two measures produce? What do you conclude?

c. For each day in the sample, compute the 95% confidence intervals of the historical VaR and the exponential weighted VaR you obtained in question 1, using both parametric (for the historical VaR) and bootstrap methods (for the two measures). For the parametric method, assume the gains are normally distributed.

Solution to 1(a)

To address this question, we computed the 1-day 99% Value-at-Risk (VaR) for daily returns over the period from January 2, 2015 to December 19, 2017, using three methods: Moving Window Historical VaR, Expanding Window Historical VaR, and Exponentially Weighted Moving Average (EWMA) VaR with a decay factor of $\lambda = 0.995$.

Moving Window Historical VaR was calculated using a rolling window of 252 trading days. For each trading day, we computed the 1st percentile of the preceding 252 returns, providing a localized empirical risk measure that adapts to recent conditions.

Expanding Window Historical VaR was computed by progressively incorporating all available past data up to the current date. This method smooths out short-term fluctuations and relies on the entire cumulative return history, giving a more stable but slower-reacting risk estimate.

EWMA VaR was computed by applying an exponential weighting to past squared returns, following the recursion

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda) r_{t-1}^2.$$

Assuming normally distributed returns, the 1-day VaR was obtained by scaling the estimated standard deviation by the appropriate z-score.

The final output included daily returns, Moving Window Historical VaR, Expanding Window Historical VaR, and EWMA VaR estimates. These risk measures are reported as positive values, consistent with the convention of expressing VaR as a positive threshold loss.

In [11]:
```
import pandas as pd
import numpy as np
```

```python
from scipy.stats import norm

def compute_var(filepath, start_date, end_date, lambda_=0.995, var_confidence=0.99,
    """
    Computes:
    - Daily returns
    - Moving window Historical VaR
    - Expanding (accumulating) Historical VaR
    - EWMA VaR

    Assumes:
    - First column is Date
    - Second column is Returns

    Parameters:
    filepath (str): Path to the CSV file.
    start_date (str): Start date for VaR reporting (YYYY-MM-DD).
    end_date (str): End date for VaR reporting (YYYY-MM-DD).
    lambda_ (float): Decay factor for EWMA (default 0.995).
    var_confidence (float): Confidence level for VaR (default 0.99).
    rolling_window (int): Rolling window size for moving Historical VaR (default 25

    Returns:
    pd.DataFrame: DataFrame with columns ['returns', 'hist_var_moving', 'hist_var_e
    """
    returns = pd.read_csv(filepath)
    returns.columns = ['Date', 'Returns']
    returns['Date'] = pd.to_datetime(returns['Date'])
    returns = returns.set_index('Date').sort_index()

    hist_var_moving_full = returns.rolling(window=rolling_window, min_periods=rolli
    hist_var_expanding_full = returns.expanding(min_periods=rolling_window).quantil

    ewma_variance = returns.copy()
    ewma_variance.iloc[0] = returns.var()

    for t in range(1, len(returns)):
        ewma_variance.iloc[t] = lambda_ * ewma_variance.iloc[t-1] + (1 - lambda_) *

    z_score = norm.ppf(var_confidence)
    var_ewma_full = z_score * np.sqrt(ewma_variance)

    result_full = pd.DataFrame({
        'returns': returns.squeeze(),
        'hist_var_moving': -hist_var_moving_full.squeeze(),
        'hist_var_expanding': -hist_var_expanding_full.squeeze(),
        'ewma_var': var_ewma_full.squeeze()
    }, index=returns.index)

    result = result_full.loc[start_date:end_date]

    return result

var_estimates = compute_var('hw4_returns.csv', start_date='2015-01-01', end_date='2
var_estimates
```

Out[11]:

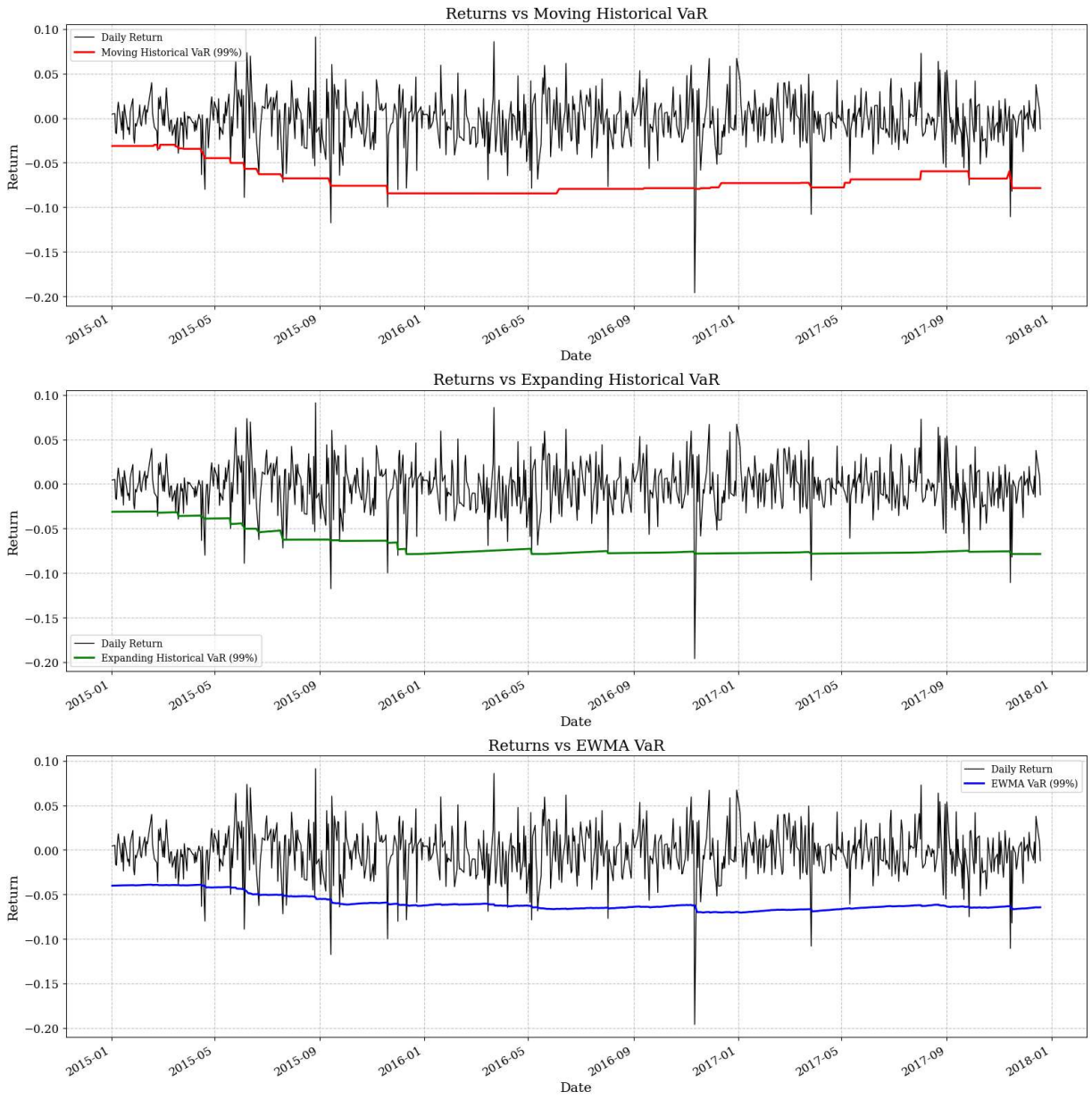
Date	returns	hist_var_moving	hist_var_expanding	ewma_var
2015-01-02	0.004451	0.031215	0.031205	0.040190
2015-01-05	0.005089	0.031215	0.031194	0.040096
2015-01-06	-0.016496	0.031215	0.031183	0.040004
2015-01-07	-0.017240	0.031215	0.031172	0.039996
2015-01-08	0.007742	0.031215	0.031162	0.039997
...	...	...	...	...
2017-12-13	-0.014844	0.078521	0.078554	0.064596
2017-12-14	0.037677	0.078521	0.078553	0.064481
2017-12-15	0.028865	0.078521	0.078552	0.064617
2017-12-18	0.009881	0.078521	0.078550	0.064630
2017-12-19	-0.012141	0.078521	0.078549	0.064489

748 rows × 4 columns

In [14]:
```
import matplotlib.pyplot as plt

def plot_returns_vs_var_vertical(var_estimates):
    """
    Plots Daily Returns against Moving Historical VaR, Expanding Historical VaR, an
    in three vertically stacked subplots with x-axis dates visible for all.

    Parameters:
    var_estimates (pd.DataFrame): DataFrame containing 'returns', 'hist_var_moving'
    """
    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
    })

    fig, axes = plt.subplots(3, 1, figsize=(16, 18), sharex=False)

    axes[0].plot(var_estimates.index, var_estimates['returns'], label='Daily Return
    axes[0].plot(var_estimates.index, -var_estimates['hist_var_moving'], label='Mov
    axes[0].set_title('Returns vs Moving Historical VaR')
    axes[0].set_ylabel('Return')
    axes[0].set_xlabel('Date')
    axes[0].grid(True, linestyle='--', alpha=0.7)
    axes[0].legend()
    plt.setp(axes[0].get_xticklabels(), rotation=30, ha='right')
```

```
    axes[1].plot(var_estimates.index, var_estimates['returns'], label='Daily Return
    axes[1].plot(var_estimates.index, -var_estimates['hist_var_expanding'], label='
    axes[1].set_title('Returns vs Expanding Historical VaR')
    axes[1].set_ylabel('Return')
    axes[1].set_xlabel('Date')
    axes[1].grid(True, linestyle='--', alpha=0.7)
    axes[1].legend()
    plt.setp(axes[1].get_xticklabels(), rotation=30, ha='right')

    axes[2].plot(var_estimates.index, var_estimates['returns'], label='Daily Return
    axes[2].plot(var_estimates.index, -var_estimates['ewma_var'], label='EWMA VaR (
    axes[2].set_title('Returns vs EWMA VaR')
    axes[2].set_ylabel('Return')
    axes[2].set_xlabel('Date')
    axes[2].grid(True, linestyle='--', alpha=0.7)
    axes[2].legend()
    plt.setp(axes[2].get_xticklabels(), rotation=30, ha='right')

    fig.suptitle('Daily Returns and 1-Day VaR Estimates (2015-2017)', fontsize=20,
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

plot_returns_vs_var_vertical(var_estimates)
```

Daily Returns and 1-Day VaR Estimates (2015–2017)



Solution to 1(b)

We backtested the three VaR measures — Moving Window Historical VaR, Expanding Window Historical VaR, and EWMA VaR — by counting the number of exceptions, defined as days when the realized daily return fell below the predicted VaR threshold.

Over the 748 observations between January 2, 2015 and December 19, 2017, we observed the following:

- **Moving Window Historical VaR Exceptions:** 14 exceptions
- **Expanding Window Historical VaR Exceptions:** 18 exceptions

- **EWMA VaR Exceptions:** 24 exceptions
- **Expected Exceptions (at 99% confidence):** approximately 7.48 exceptions

To statistically assess whether these models correctly capture the intended risk level, we conducted one-sided binomial hypothesis tests. Under the null hypothesis, the number of exceptions follows a Binomial distribution with probability of exception $p = 0.01$.

The resulting p-values for observing at least as many exceptions as reported are:

- **Moving Window Historical VaR:** p-value = 2.06% → **Reject** H_0
- **Expanding Window Historical VaR:** p-value = 0.071% → **Reject** H_0
- **EWMA VaR:** p-value ≈ 0.0001% → **Reject** H_0

At a 5% significance level, we reject the null hypothesis for all three models. This indicates that none of the models adequately captured the true tail risk at the 99% confidence level during the sample period, with the EWMA VaR model exhibiting the most severe underestimation of downside risk.

Additional Discussion: Decay Factor $\lambda = 0.995$ Might Be Too High

- A high λ like 0.995 **smooths volatility heavily**.
- This may cause the model to **react too slowly to sudden large shocks**.
- In crisis periods or periods with abrupt changes (e.g., mid-2015 China devaluation, late-2015 turbulence), the model **lags behind the true increase in risk**.

→ **EWMA thinks volatility is still calm when actually it has jumped.**

In [15]:
```python
def backtest_var(var_estimates, var_confidence=0.99):
    """
    Backtests the Moving Historical VaR, Expanding Historical VaR, and EWMA VaR by

    Parameters:
    var_estimates (pd.DataFrame): DataFrame containing 'returns', 'hist_var_moving'
    var_confidence (float): Confidence level for VaR (default 0.99).

    Returns:
    dict: Dictionary containing number of exceptions for each VaR method,
          expected number of exceptions, and total number of observations.
    """
    returns = var_estimates['returns']

    hist_moving_exceptions = (returns < -var_estimates['hist_var_moving']).sum()
    hist_expanding_exceptions = (returns < -var_estimates['hist_var_expanding']).su
    ewma_exceptions = (returns < -var_estimates['ewma_var']).sum()

    expected_exceptions = (1 - var_confidence) * len(var_estimates)
```

```
    results = {
        'Moving Historical VaR Exceptions': hist_moving_exceptions,
        'Expanding Historical VaR Exceptions': hist_expanding_exceptions,
        'EWMA VaR Exceptions': ewma_exceptions,
        'Expected Exceptions': expected_exceptions,
        'Total Observations': len(var_estimates)
    }

    return results

backtest_results = backtest_var(var_estimates)
backtest_results
```

Out[15]:
```
{'Moving Historical VaR Exceptions': 14,
 'Expanding Historical VaR Exceptions': 18,
 'EWMA VaR Exceptions': 24,
 'Expected Exceptions': 7.480000000000007,
 'Total Observations': 748}
```

In [17]:
```
from scipy.stats import binom

def binomial_hypothesis_test_exceedance(observed_exceptions, total_observations=748
    """
    Conducts a one-sided binomial hypothesis test for excess VaR exceptions.

    Parameters:
    observed_exceptions (int): Number of observed exceptions.
    total_observations (int): Total number of observations (default 748).
    c (float): VaR confidence level (default 0.99).
    alpha (float): Significance level for hypothesis test (default 0.05).

    Returns:
    dict: Contains p-value, critical region, and test result (Reject or Fail to Rej
    """
    p = 1 - c

    # One-sided p-value: probability of getting observed_exceptions or more
    p_value = binom.sf(observed_exceptions - 1, total_observations, p)

    result = "Reject H0" if p_value < alpha else "Fail to Reject H0"

    return {
        'Observed Exceptions': observed_exceptions,
        'Expected Exceptions': total_observations * p,
        'p-Value (P[X >= observed])': p_value,
        'Significance Level (alpha)': alpha,
        'Test Conclusion': result
    }

# Apply to all three models
moving_hist_binomial_test = binomial_hypothesis_test_exceedance(14, c=0.99)
expanding_hist_binomial_test = binomial_hypothesis_test_exceedance(18, c=0.99)
ewma_binomial_test = binomial_hypothesis_test_exceedance(24, c=0.99)

moving_hist_binomial_test, expanding_hist_binomial_test, ewma_binomial_test
```

Out[17]:
```
({'Observed Exceptions': 14,
  'Expected Exceptions': 7.480000000000007,
  'p-Value (P[X >= observed])': 0.02057274174848624,
  'Significance Level (alpha)': 0.05,
  'Test Conclusion': 'Reject H0'},
 {'Observed Exceptions': 18,
  'Expected Exceptions': 7.480000000000007,
  'p-Value (P[X >= observed])': 0.0007138522184448955,
  'Significance Level (alpha)': 0.05,
  'Test Conclusion': 'Reject H0'},
 {'Observed Exceptions': 24,
  'Expected Exceptions': 7.480000000000007,
  'p-Value (P[X >= observed])': 1.0148495432822414e-06,
  'Significance Level (alpha)': 0.05,
  'Test Conclusion': 'Reject H0'})
```

Solution to 1(c)

For each day in the sample, we constructed 95% confidence intervals around the Historical VaR (both rolling and expanding window) and the EWMA VaR obtained in Question 1(a).

Specifically:

- **Rolling Historical VaR:** Parametric confidence intervals were computed under the assumption of normally distributed returns. Additionally, nonparametric bootstrap confidence intervals were generated by resampling past returns over a 252-day rolling window.
- **Expanding Historical VaR:** Since the sample size grows over time and normality assumptions become unreliable, only bootstrap confidence intervals were computed using resampling of the expanding set of returns.
- **EWMA VaR:** Due to the dynamic nature of volatility estimation under the EWMA framework, bootstrap methods were employed to estimate confidence intervals, accounting for stochastic volatility behavior.

The results reveal several important observations:

- For Rolling Historical VaR, parametric and bootstrap confidence intervals are relatively tight, though bootstrap bands are slightly wider, reflecting the empirical distribution's heavier tails.
- For Expanding Historical VaR, the bootstrap confidence intervals initially exhibit greater variability due to the small early sample size, but stabilize as more data accumulates over time.

- For EWMA VaR, bootstrap confidence intervals are asymmetric and wider during periods of elevated market volatility, capturing the dynamics of time-varying risk under the exponentially weighted variance structure.

Overall, bootstrap-based intervals provide a more robust and flexible assessment of VaR uncertainty, especially when model assumptions such as normality or stationarity are questionable.

In [21]:
```python
import pandas as pd
import numpy as np
from scipy.stats import norm
from tqdm import tqdm

def compute_var_cis_with_ewma(
    returns_df,
    var_confidence=0.99,
    rolling_window=252,
    bootstrap_confidence=0.95,
    bootstrap_iterations=1000,
    lambda_ewma=0.995
):
    """
    Computes 95% confidence intervals for historical and EWMA VaR using both parame

    Parameters:
    - returns_df (pd.DataFrame): DataFrame with date index and a single column of d
    - var_confidence (float): Confidence level for VaR (default = 0.99).
    - rolling_window (int): Rolling window size (default = 252).
    - bootstrap_confidence (float): Confidence level for CIs (default = 0.95).
    - bootstrap_iterations (int): Number of bootstrap replications (default = 1000)
    - lambda_ewma (float): EWMA decay factor (default = 0.995).

    Returns:
    - pd.DataFrame: DataFrame indexed by date containing VaR estimates and confiden
    """
    z = norm.ppf(1 - (1 - bootstrap_confidence) / 2)
    returns = returns_df.squeeze()
    results = []

    for t in tqdm(range(rolling_window, len(returns))):
        date = returns.index[t]
        rolling_sample = returns.iloc[t - rolling_window:t]
        expanding_sample = returns.iloc[:t]

        var_roll = np.quantile(rolling_sample, 1 - var_confidence)
        var_exp = np.quantile(expanding_sample, 1 - var_confidence)

        mu = rolling_sample.mean()
        sigma = rolling_sample.std(ddof=1)
        var_param = norm.ppf(1 - var_confidence, loc=mu, scale=sigma)
        se = sigma / np.sqrt(rolling_window)
        ci_param_lower = var_param - z * se
```

```
        ci_param_upper = var_param + z * se

        ewma_weights = (1 - lambda_ewma) * lambda_ewma ** np.arange(rolling_window)
        ewma_variance = np.sum(ewma_weights * rolling_sample ** 2)
        ewma_var = np.sqrt(ewma_variance) * norm.ppf(var_confidence)

        boot_var_roll = []
        boot_var_exp = []
        boot_var_ewma = []

        for _ in range(bootstrap_iterations):
            sample_roll = np.random.choice(rolling_sample, size=rolling_window, rep
            sample_exp = np.random.choice(expanding_sample, size=len(expanding_samp

            boot_var_roll.append(np.quantile(sample_roll, 1 - var_confidence))
            boot_var_exp.append(np.quantile(sample_exp, 1 - var_confidence))

            ewma_boot_weights = (1 - lambda_ewma) * lambda_ewma ** np.arange(rollin
            ewma_boot_var = np.sqrt(np.sum(ewma_boot_weights * sample_roll ** 2)) *
            boot_var_ewma.append(ewma_boot_var)

        ci_boot_roll = np.percentile(boot_var_roll, [(1 - bootstrap_confidence) / 2
                                                     (1 + bootstrap_confidence) /
        ci_boot_exp = np.percentile(boot_var_exp, [(1 - bootstrap_confidence) / 2 *
                                                   (1 + bootstrap_confidence) / 2
        ci_boot_ewma = np.percentile(boot_var_ewma, [(1 - bootstrap_confidence) / 2
                                                     (1 + bootstrap_confidence) /

        results.append({
            'Date': date,
            'VaR_Hist_Rolling': var_roll,
            'VaR_Hist_Expanding': var_exp,
            'VaR_EWMA': ewma_var,
            'CI_Param_Lower': ci_param_lower,
            'CI_Param_Upper': ci_param_upper,
            'CI_Boot_Rolling_Lower': ci_boot_roll[0],
            'CI_Boot_Rolling_Upper': ci_boot_roll[1],
            'CI_Boot_Expanding_Lower': ci_boot_exp[0],
            'CI_Boot_Expanding_Upper': ci_boot_exp[1],
            'CI_Boot_EWMA_Lower': ci_boot_ewma[0],
            'CI_Boot_EWMA_Upper': ci_boot_ewma[1],
        })

    return pd.DataFrame(results).set_index("Date")
```

In [22]:
```
df = pd.read_csv("hw4_returns.csv", parse_dates=[0])
df.columns = ['Date', 'Returns']
df = df.set_index('Date').sort_index()

ci_df = compute_var_cis_with_ewma(df, var_confidence=0.99, rolling_window=252, boot
```

```
100%|██████████████████████████████████████████████████████████
| 748/748 [10:03<00:00,  1.24it/s]
```

In [23]:
```
ci_df
```

Out[23]:

Date	VaR_Hist_Rolling	VaR_Hist_Expanding	VaR_EWMA	CI_Param_Lower	CI_Param_Upper
2015-01-02	-0.031215	-0.031215	0.025355	-0.031512	-0.028384
2015-01-05	-0.031215	-0.031205	0.025299	-0.031512	-0.028384
2015-01-06	-0.031215	-0.031194	0.025244	-0.031512	-0.028384
2015-01-07	-0.031215	-0.031183	0.025327	-0.031669	-0.028532
2015-01-08	-0.031215	-0.031172	0.025386	-0.031806	-0.028668
...	...	...	...	...	...
2017-12-13	-0.078521	-0.078555	0.052662	-0.065591	-0.059134
2017-12-14	-0.078521	-0.078554	0.052571	-0.065594	-0.059136
2017-12-15	-0.078521	-0.078553	0.052772	-0.065571	-0.059093
2017-12-18	-0.078521	-0.078552	0.052850	-0.065589	-0.059094
2017-12-19	-0.078521	-0.078550	0.052712	-0.065422	-0.058932

748 rows × 11 columns

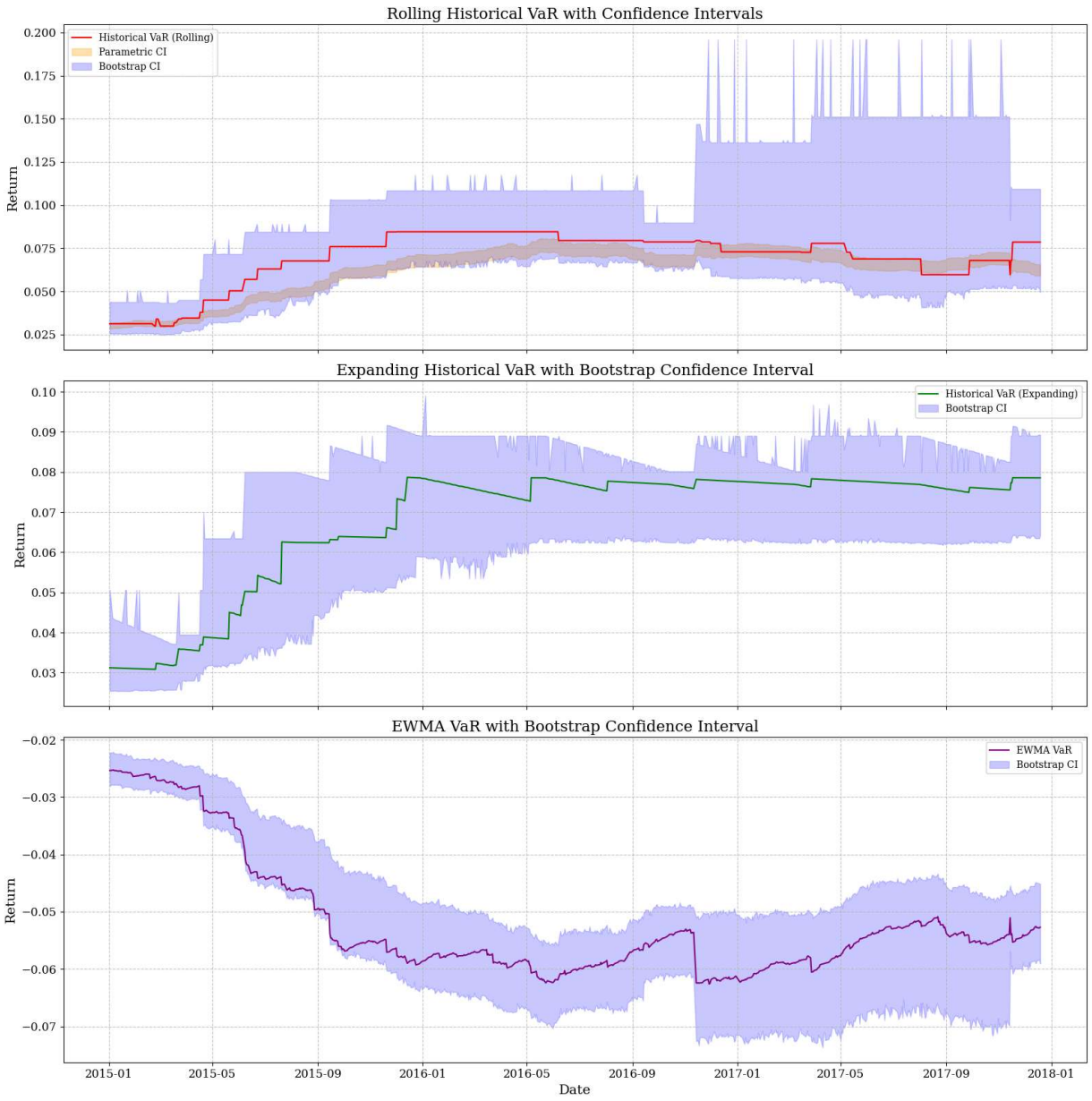
In [24]:

```
import matplotlib.pyplot as plt

def plot_var_with_cis(ci_df, var_estimates):
    """
    Plots Historical VaR (Rolling), Historical VaR (Expanding), and EWMA VaR
    along with their respective confidence intervals.

    Parameters:
    - ci_df (pd.DataFrame): Output from compute_var_cis_with_ewma().
    - var_estimates (pd.DataFrame): DataFrame containing returns and VaR estimates
    """
    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
```

```
    })

    fig, axes = plt.subplots(3, 1, figsize=(16, 18), sharex=True)

    # Plot 1: Historical VaR (Rolling Window)
    axes[0].plot(ci_df.index, -ci_df['VaR_Hist_Rolling'], label='Historical VaR (Ro
    axes[0].fill_between(ci_df.index, -ci_df['CI_Param_Lower'], -ci_df['CI_Param_Up
                         color='orange', alpha=0.3, label='Parametric CI')
    axes[0].fill_between(ci_df.index, -ci_df['CI_Boot_Rolling_Lower'], -ci_df['CI_B
                         color='blue', alpha=0.2, label='Bootstrap CI')
    axes[0].set_title('Rolling Historical VaR with Confidence Intervals')
    axes[0].set_ylabel('Return')
    axes[0].grid(True, linestyle='--', alpha=0.7)
    axes[0].legend()

    # Plot 2: Historical VaR (Expanding Window)
    axes[1].plot(ci_df.index, -ci_df['VaR_Hist_Expanding'], label='Historical VaR (
    axes[1].fill_between(ci_df.index, -ci_df['CI_Boot_Expanding_Lower'], -ci_df['CI
                         color='blue', alpha=0.2, label='Bootstrap CI')
    axes[1].set_title('Expanding Historical VaR with Bootstrap Confidence Interval'
    axes[1].set_ylabel('Return')
    axes[1].grid(True, linestyle='--', alpha=0.7)
    axes[1].legend()

    # Plot 3: EWMA VaR
    axes[2].plot(ci_df.index, -ci_df['VaR_EWMA'], label='EWMA VaR', color='purple')
    axes[2].fill_between(ci_df.index, -ci_df['CI_Boot_EWMA_Lower'], -ci_df['CI_Boot
                         color='blue', alpha=0.2, label='Bootstrap CI')
    axes[2].set_title('EWMA VaR with Bootstrap Confidence Interval')
    axes[2].set_xlabel('Date')
    axes[2].set_ylabel('Return')
    axes[2].grid(True, linestyle='--', alpha=0.7)
    axes[2].legend()

    fig.suptitle('Value at Risk Estimates with Confidence Intervals', fontsize=20)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```

In [25]:
```
plot_var_with_cis(ci_df, var_estimates)
```

Value at Risk Estimates with Confidence Intervals



---

2. *Model-building approach*

(a) Compute volatility using the EWMA with $\lambda = 0.96$. Compute the corresponding measure of VaR.

(b) Use maximum likelihood estimation to estimate a GARCH model for volatility. Compute the corresponding measure of VaR.

(c) Compare the results from the two approaches.

Solution to 2(a)

In this part, we implement the model-based approach for computing volatility and Value-at-Risk (VaR) using the Exponentially Weighted Moving Average (EWMA) model. We apply an EWMA decay factor of $\lambda = 0.96$, as specified.

The conditional variance is updated recursively as:

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda) r_{t-1}^2$$

where r_{t-1} is the return at time $t - 1$. The initial variance estimate is computed using the sample variance from the first 252 trading days.

The corresponding 1-day VaR at the 99% confidence level is obtained under the assumption of normally distributed returns, as:

$$\text{VaR}_t = z_{0.99} \cdot \sigma_t \quad \text{where } z_{0.99} = \Phi^{-1}(0.99) \approx 2.33$$

The plot below illustrates the daily return series, the estimated volatility, and the corresponding VaR path over time. The EWMA volatility bands adapt dynamically to the level of market turbulence — widening in high volatility periods and tightening during calmer intervals.

The average EWMA volatility over the full sample is approximately **2.35%** per day. The EWMA VaR tracks expected downside risk, accounting for time-varying volatility more responsively than static methods.

In [29]:
```python
import numpy as np
import pandas as pd
from scipy.stats import norm

def compute_ewma_var(returns_df, lambda_ewma=0.96, var_confidence=0.99):
    """
    Computes EWMA volatility, corresponding 1-day VaR, and stores daily returns.

    Parameters:
    - returns_df (pd.DataFrame): DataFrame with 'Returns' column, indexed by date.
    - lambda_ewma (float): Decay factor for EWMA (default = 0.96).
    - var_confidence (float): Confidence level for VaR (default = 0.99).

    Returns:
    - pd.DataFrame: DataFrame with daily returns, EWMA volatility, and EWMA VaR.
    """
    returns = returns_df.squeeze()
    returns_squared = returns ** 2
```

```
    sigma2_0 = np.var(returns.iloc[:252])
    sigma2 = np.zeros(len(returns) + 1)
    sigma2[0] = sigma2_0

    for t in range(1, len(sigma2)):
        sigma2[t] = lambda_ewma * sigma2[t-1] + (1 - lambda_ewma) * returns_squared

    ewma_volatility = np.sqrt(sigma2[1:])
    var_threshold = norm.ppf(1 - var_confidence)
    ewma_var = ewma_volatility * var_threshold

    results = pd.DataFrame({
        'Returns': returns,
        'EWMA_Volatility': ewma_volatility,
        'EWMA_VaR': ewma_var
    }, index=returns.index)

    return results
```

In [30]:
```
df = pd.read_csv("hw4_returns.csv", parse_dates=[0])
df.columns = ['Date', 'Returns']
df = df.set_index('Date').sort_index()

ewma_results = compute_ewma_var(df, lambda_ewma=0.96, var_confidence=0.99)
ewma_results
```

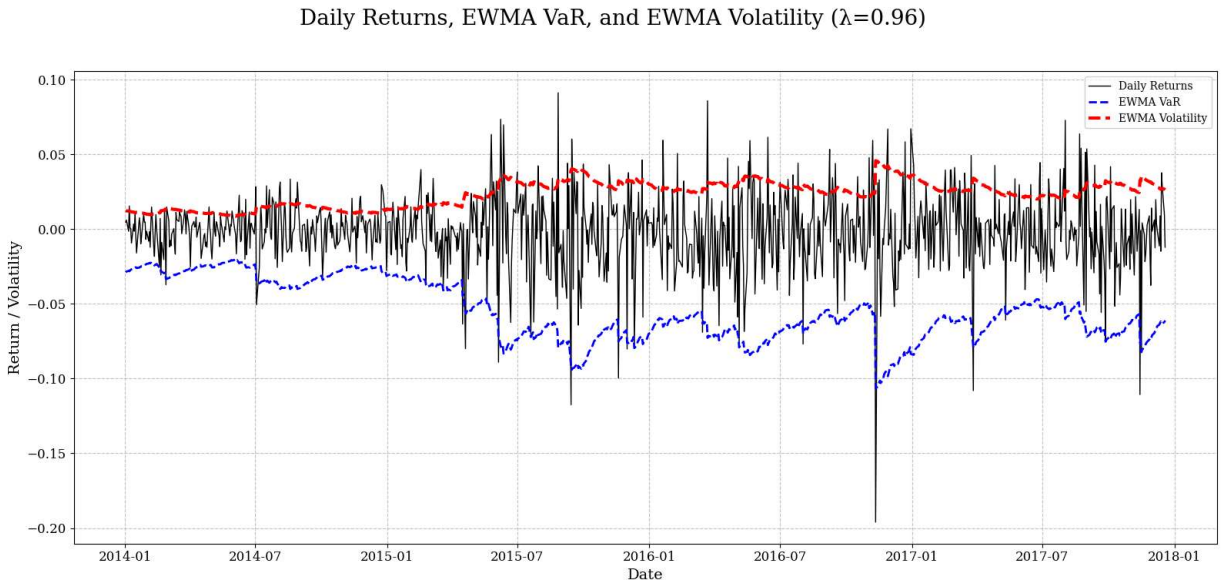
Out[30]:

Date	Returns	EWMA_Volatility	EWMA_VaR
2014-01-02	0.004572	0.012422	-0.028898
2014-01-03	0.006045	0.012231	-0.028453
2014-01-06	-0.001432	0.011987	-0.027886
2014-01-07	0.015461	0.012145	-0.028254
2014-01-08	0.000763	0.011901	-0.027685
...	...	...	...
2017-12-13	-0.014844	0.026580	-0.061835
2017-12-14	0.037677	0.027112	-0.063071
2017-12-15	0.028865	0.027184	-0.063239
2017-12-18	0.009881	0.026708	-0.062132
2017-12-19	-0.012141	0.026281	-0.061138

1000 rows × 3 columns

In [45]:
```
def plot_returns_vs_ewma(ewma_results):
    """
```

```
    Plots Daily Returns, EWMA VaR, and EWMA Volatility on the same figure.

    Parameters:
    - ewma_results (pd.DataFrame): Output DataFrame from compute_ewma_var().
    """
    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
    })

    fig, ax = plt.subplots(figsize=(16, 8))

    ax.plot(ewma_results.index, ewma_results['Returns'], label='Daily Returns', col
    ax.plot(ewma_results.index, ewma_results['EWMA_VaR'], label='EWMA VaR', color='
    ax.plot(ewma_results.index, ewma_results['EWMA_Volatility'], label='EWMA Volati

    ax.set_xlabel('Date')
    ax.set_ylabel('Return / Volatility')
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend()

    fig.suptitle('Daily Returns, EWMA VaR, and EWMA Volatility (λ=0.96)', fontsize=
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()


ewma_results = compute_ewma_var(df, lambda_ewma=0.96, var_confidence=0.99)
plot_returns_vs_ewma(ewma_results)
```



In [41]:
```
print(f'{ewma_results['EWMA_Volatility'].mean()*100:0.2f}%')
```

2.35%

Solution to 2(b)

In this part, we estimate the conditional volatility of returns using a **GARCH(1,1)** model via **maximum likelihood estimation (MLE)**. The GARCH framework captures the time-varying nature of volatility by allowing the current conditional variance to depend on both past squared returns and past conditional variances:

$$\sigma_t^2 = \omega + \alpha r_{t-1}^2 + \beta \sigma_{t-1}^2$$

We use the `arch` package to fit the model under the assumption of zero mean and normally distributed innovations. The fitted model provides the sequence of one-step-ahead conditional volatility estimates, which we use to compute the 1-day VaR at the 99% confidence level:

$$\text{VaR}_t = z_{0.99} \cdot \sigma_t$$

The figure below displays the daily returns along with the estimated conditional volatility and VaR. As expected, the GARCH model reacts more quickly to large shocks in returns, producing tighter and more responsive volatility estimates compared to historical approaches.

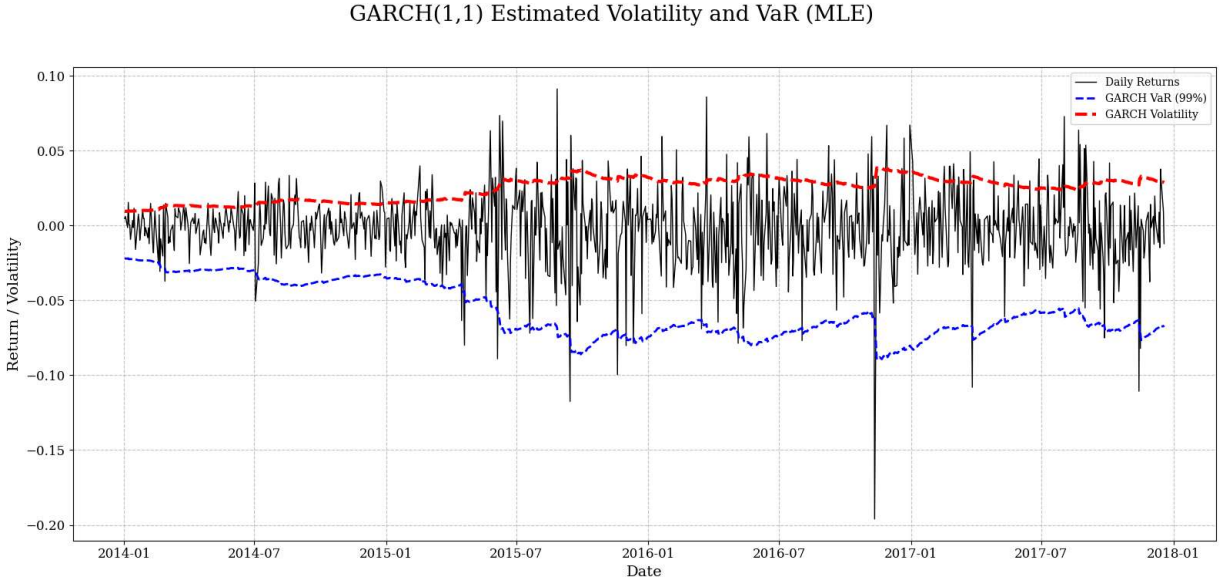
This model-building approach is considered more robust in capturing volatility clustering and leverage effects in financial return series, making it a valuable tool for dynamic risk management.

In [42]:
```python
import pandas as pd
from arch import arch_model
from scipy.stats import norm

def compute_garch_var(returns_df, var_confidence=0.99):
    """
    Fits a GARCH(1,1) model to returns and computes conditional volatility and VaR.

    Parameters:
    - returns_df (pd.DataFrame): DataFrame with 'Returns' column, indexed by date.
    - var_confidence (float): Confidence level for VaR (default = 0.99).

    Returns:
    - pd.DataFrame: DataFrame with conditional volatility and 1-day VaR from GARCH
    - arch_model_result: fitted GARCH model result object (for diagnostics or later
    """
    returns = returns_df['Returns'].dropna() * 100  # ARCH expects returns in % uni

    # Estimate GARCH(1,1) model
    model = arch_model(returns, vol='GARCH', p=1, q=1, mean='Zero', dist='normal')
    res = model.fit(disp='off')
```

```
    # Get conditional volatility forecast
    cond_vol = res.conditional_volatility / 100  # back to raw units

    # Compute 1-day VaR using normal assumption
    z = norm.ppf(1 - var_confidence)
    garch_var = z * cond_vol

    result_df = pd.DataFrame({
        'Returns': returns / 100,
        'GARCH_Conditional_Vol': cond_vol,
        'GARCH_VaR': garch_var
    }, index=returns.index)

    return result_df, res
```

In [43]:
```
garch_results, garch_model = compute_garch_var(df, var_confidence=0.99)
```

In [46]:
```
def plot_returns_vs_garch(garch_results):
    """
    Plots Returns vs GARCH VaR and Conditional Volatility

    Parameters:
    - garch_results (pd.DataFrame): Output from compute_garch_var()
    """
    import matplotlib.pyplot as plt

    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
    })

    fig, ax = plt.subplots(figsize=(16, 8))
    ax.plot(garch_results.index, garch_results['Returns'], label='Daily Returns', c
    ax.plot(garch_results.index, garch_results['GARCH_VaR'], label='GARCH VaR (99%)
    ax.plot(garch_results.index, garch_results['GARCH_Conditional_Vol'], label='GAR

    ax.set_xlabel('Date')
    ax.set_ylabel('Return / Volatility')
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend()
    fig.suptitle('GARCH(1,1) Estimated Volatility and VaR (MLE)', fontsize=20)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
plot_returns_vs_garch(garch_results)
```



Solution to 2(c)

In this part, we compare the risk estimates produced by the EWMA and GARCH(1,1) models based on both Value-at-Risk (VaR) and conditional volatility forecasts.

Summary statistics:

- Average EWMA VaR: -5.48%
- Average GARCH VaR: -5.73%
- Average EWMA Volatility: 2.35%
- Average GARCH Volatility: 2.46%

Both models produce comparable magnitudes of risk estimates on average. GARCH VaR tends to be slightly more conservative (lower, i.e., more negative) than EWMA VaR, which is consistent with GARCH's ability to capture volatility clustering and sharp risk spikes more precisely.

From the plotted trajectories, we observe that:

- EWMA volatility and VaR adjust gradually to changes, resulting in smoother paths.
- GARCH volatility and VaR respond more sharply to return shocks, producing sudden jumps in risk forecasts.
- During periods of market stress, GARCH estimates risk more aggressively, while EWMA lags slightly behind the true increase in volatility.

Backtesting violations:

- EWMA model produced 15 violations (returns falling below VaR).
- GARCH model produced 21 violations.
- Expected number of violations at 1% level: 10.00.

Both models exhibit more violations than expected under ideal calibration. However, the GARCH model has more violations than the EWMA model, indicating potential underestimation of tail risk at certain points, possibly due to volatility model misspecification or the presence of extreme returns not captured by the normality assumption.

Conclusion: While both models capture time-varying risk, the GARCH(1,1) model is more sensitive to sudden shifts and better models volatility clustering. EWMA, with its high smoothing parameter $\lambda = 0.96$, provides smoother but slower-reacting risk estimates. The choice between models should depend on the specific risk management objective: responsiveness versus stability.

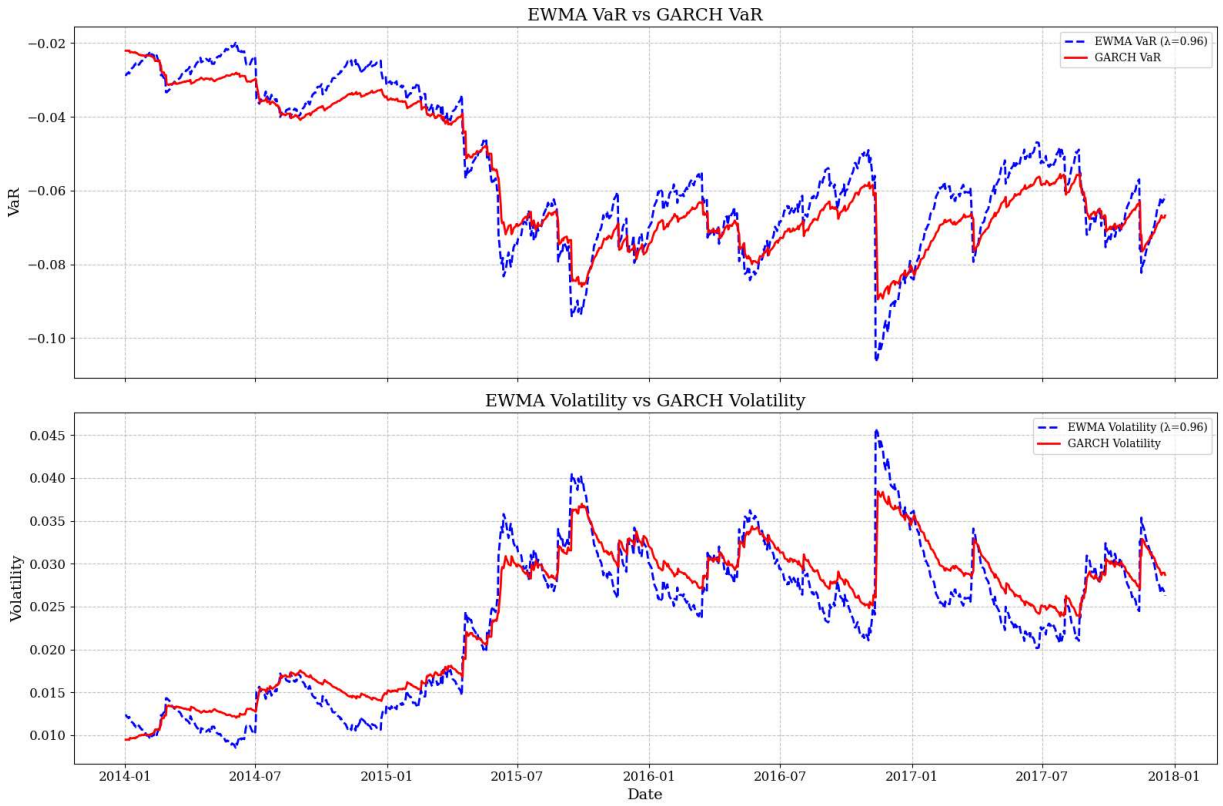
In [47]:
```
print(f"Average EWMA VaR: {ewma_results['EWMA_VaR'].mean():.4f}")
print(f"Average GARCH VaR: {garch_results['GARCH_VaR'].mean():.4f}")
print(f"EWMA Volatility (avg): {ewma_results['EWMA_Volatility'].mean():.4f}")
print(f"GARCH Volatility (avg): {garch_results['GARCH_Conditional_Vol'].mean():.4f}
ewma_violations = (ewma_results['Returns'] < ewma_results['EWMA_VaR']).sum()
garch_violations = (garch_results['Returns'] < garch_results['GARCH_VaR']).sum()

print(f"EWMA Violations: {ewma_violations}")
print(f"GARCH Violations: {garch_violations}")
expected = 0.01 * len(ewma_results)
print(f"Expected Violations (1%): {expected:.2f}")
```

```
Average EWMA VaR: -0.0548
Average GARCH VaR: -0.0573
EWMA Volatility (avg): 0.0235
GARCH Volatility (avg): 0.0246
EWMA Violations: 15
GARCH Violations: 21
Expected Violations (1%): 10.00
```

In [48]:
```
import matplotlib.pyplot as plt

def plot_ewma_vs_garch(ewma_results, garch_results):
    """
    Plots:
    - EWMA VaR vs GARCH VaR
    - EWMA Volatility vs GARCH Volatility

    Parameters:
    - ewma_results (pd.DataFrame): Output from compute_ewma_var().
    - garch_results (pd.DataFrame): Output from compute_garch_var().
    """
    plt.rcParams.update({
```

```
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
    })

    fig, axes = plt.subplots(2, 1, figsize=(16, 12), sharex=True)

    # Plot 1: VaR Comparison
    axes[0].plot(ewma_results.index, ewma_results['EWMA_VaR'], label='EWMA VaR (λ=0
    axes[0].plot(garch_results.index, garch_results['GARCH_VaR'], label='GARCH VaR'
    axes[0].set_title('EWMA VaR vs GARCH VaR')
    axes[0].set_ylabel('VaR')
    axes[0].grid(True, linestyle='--', alpha=0.7)
    axes[0].legend()

    # Plot 2: Volatility Comparison
    axes[1].plot(ewma_results.index, ewma_results['EWMA_Volatility'], label='EWMA V
    axes[1].plot(garch_results.index, garch_results['GARCH_Conditional_Vol'], label
    axes[1].set_title('EWMA Volatility vs GARCH Volatility')
    axes[1].set_xlabel('Date')
    axes[1].set_ylabel('Volatility')
    axes[1].grid(True, linestyle='--', alpha=0.7)
    axes[1].legend()

    fig.suptitle('Comparison of EWMA and GARCH Risk Estimates', fontsize=20)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
plot_ewma_vs_garch(ewma_results, garch_results)
```

Comparison of EWMA and GARCH Risk Estimates



---

3. *A mixed approach*

(a) For each day in the sample, compute the volatility of the portfolio in the previous month. Normalize gains with estimated volatility. Compare the distribution of the normalized gain with the original ones.

(b) Develop an approach to measure VaR which takes advantage of your response to the previous question. Implement it and compare its exceptions with the previous approaches. *Optional: You can use the approach of extreme value theory.*

Solution to 3(a)

In this part, we normalize daily returns by their estimated one-month rolling volatility. Specifically, for each trading day, we compute the realized volatility over the previous 21 trading days and divide the daily return by this volatility estimate. This standardization aims to stabilize the scale of the return distribution over time.

The histograms reveal the impact of normalization:

- The distribution of raw returns is narrow and heavily centered around zero but with visible outliers in both tails.
- After normalization, returns are more spread out, with values ranging approximately between -4 and +3, indicating a wider but more standardized distribution across time periods.

The QQ-plots provide a visual comparison against a standard normal distribution:

- Raw returns exhibit significant deviations from the theoretical normal line, especially in the tails, suggesting fat-tailed behavior.
- Normalized returns align much more closely to the 45-degree normal line, although slight deviations remain in the extreme tails.

Conclusion: Volatility normalization makes the return distribution closer to Gaussian. This supports the mixed approach idea that volatility clustering can be accounted for by scaling returns with their local volatility, leading to a more stable risk estimation framework.

In [51]:
```python
import numpy as np
import pandas as pd

def compute_normalized_returns(returns_df, window_days=21):
    """
    Computes normalized returns by dividing each return by rolling volatility (past

    Parameters:
    - returns_df (pd.DataFrame): DataFrame with 'Returns' column, indexed by date.
    - window_days (int): Rolling window size in trading days (default = 21 for 1 mo

    Returns:
    - pd.DataFrame: DataFrame with raw returns and normalized returns.
    """
    returns = returns_df['Returns'].dropna()

    # Rolling standard deviation over previous month
    rolling_vol = returns.rolling(window=window_days).std(ddof=0)

    # Normalized returns = raw return / rolling volatility
    normalized_returns = returns / rolling_vol

    results = pd.DataFrame({
        'Returns': returns,
        'Rolling_Volatility': rolling_vol,
        'Normalized_Returns': normalized_returns
    }, index=returns.index)

    return results
```

```
normalized_results = compute_normalized_returns(df, window_days=21)
normalized_results = normalized_results.dropna()
normalized_results
```

Out[51]:

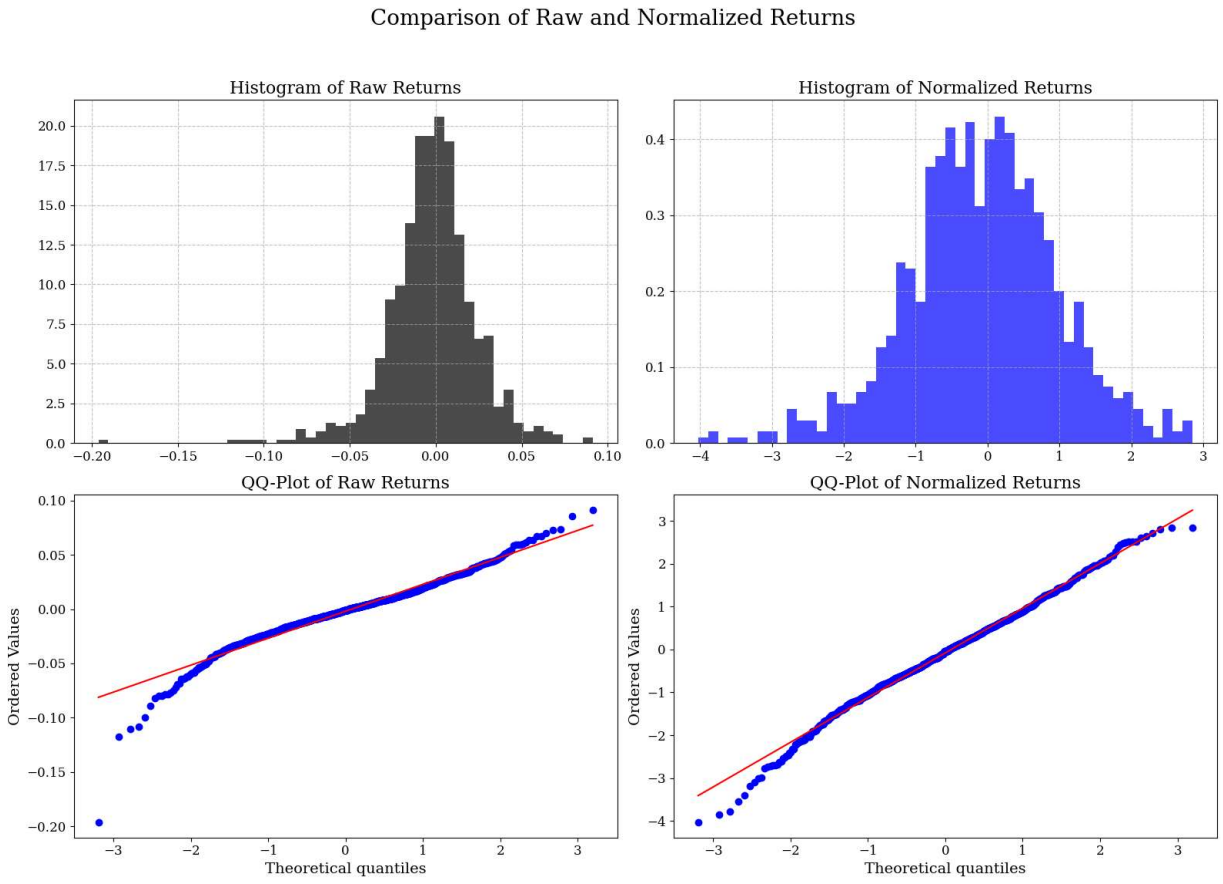
Date	Returns	Rolling_Volatility	Normalized_Returns
2014-01-31	-0.001424	0.007482	-0.190302
2014-02-03	-0.008120	0.007665	-1.059453
2014-02-04	-0.001558	0.007559	-0.206048
2014-02-05	-0.012151	0.007979	-1.522972
2014-02-06	0.003643	0.007200	0.505927
...	...	...	...
2017-12-13	-0.014844	0.030488	-0.486889
2017-12-14	0.037677	0.023263	1.619630
2017-12-15	0.028865	0.024378	1.184058
2017-12-18	0.009881	0.017156	0.575954
2017-12-19	-0.012141	0.017332	-0.700514

980 rows × 3 columns

In [52]:

```
import matplotlib.pyplot as plt
import scipy.stats as stats

def plot_comparison(normalized_results):
    """
    Plots histograms and QQ-plots comparing raw returns and normalized returns.

    Parameters:
    - normalized_results (pd.DataFrame): Output from compute_normalized_returns(),
    """
    returns = normalized_results['Returns']
    normalized = normalized_results['Normalized_Returns']

    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
    })

    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # Histogram of Returns
```

```
    axes[0,0].hist(returns, bins=50, color='black', alpha=0.7, density=True)
    axes[0,0].set_title('Histogram of Raw Returns')
    axes[0,0].grid(True, linestyle='--', alpha=0.7)

    # Histogram of Normalized Returns
    axes[0,1].hist(normalized, bins=50, color='blue', alpha=0.7, density=True)
    axes[0,1].set_title('Histogram of Normalized Returns')
    axes[0,1].grid(True, linestyle='--', alpha=0.7)

    # QQ-Plot of Returns
    stats.probplot(returns, dist="norm", plot=axes[1,0])
    axes[1,0].get_lines()[1].set_color('red')
    axes[1,0].set_title('QQ-Plot of Raw Returns')

    # QQ-Plot of Normalized Returns
    stats.probplot(normalized, dist="norm", plot=axes[1,1])
    axes[1,1].get_lines()[1].set_color('red')
    axes[1,1].set_title('QQ-Plot of Normalized Returns')

    fig.suptitle('Comparison of Raw and Normalized Returns', fontsize=20)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
plot_comparison(normalized_results)
```



Solution to 3(b)

We develop a Value-at-Risk (VaR) methodology based on the distribution of normalized returns obtained in Part 3(a). Specifically, we:

- Estimate the 1% quantile of normalized returns, assuming the normalized distribution is stable over time.
- Rescale this quantile by the local rolling volatility to obtain a VaR estimate in actual return units for each day.

The plot above shows the daily returns alongside the Mixed Approach VaR. The Mixed VaR dynamically adjusts according to changes in the estimated volatility: when recent volatility is high, the VaR moves further away from zero (i.e., becomes more negative), and when volatility is low, the VaR shrinks closer to zero.

Backtesting results:

- Observed violations: 10 times
- Expected violations under 1% level: 9.80 times

Conclusion: The Mixed Approach VaR aligns very closely with theoretical expectations. The number of violations observed matches almost exactly the expected number, suggesting that this method provides a well-calibrated risk measure. Normalizing by local volatility successfully stabilizes the risk forecast, correcting for volatility clustering in returns.

In [53]:
```python
def compute_mixed_var(normalized_results, var_confidence=0.99):
    """
    Computes VaR based on normalized returns distribution,
    and scales back to unnormalized returns using rolling volatility.

    Parameters:
    - normalized_results (pd.DataFrame): Output from compute_normalized_returns(),
    - var_confidence (float): Confidence level for VaR (default = 0.99).

    Returns:
    - pd.DataFrame: DataFrame with raw returns, rolling volatility, normalized retu
    """
    returns = normalized_results['Returns']
    rolling_vol = normalized_results['Rolling_Volatility']
    normalized = normalized_results['Normalized_Returns']

    # Compute the quantile (percentile) of normalized returns
    var_normalized = np.quantile(normalized, 1 - var_confidence)

    # Scale back VaR into return space
    var_mixed = var_normalized * rolling_vol
```

```
    mixed_var_df = normalized_results.copy()
    mixed_var_df['Mixed_VaR'] = var_mixed

    return mixed_var_df
mixed_var_results = compute_mixed_var(normalized_results, var_confidence=0.99)
mixed_var_results
```

Out[53]:

Date	Returns	Rolling_Volatility	Normalized_Returns	Mixed_VaR
2014-01-31	-0.001424	0.007482	-0.190302	-0.020490
2014-02-03	-0.008120	0.007665	-1.059453	-0.020989
2014-02-04	-0.001558	0.007559	-0.206048	-0.020701
2014-02-05	-0.012151	0.007979	-1.522972	-0.021849
2014-02-06	0.003643	0.007200	0.505927	-0.019718
...	...	...	...	...
2017-12-13	-0.014844	0.030488	-0.486889	-0.083488
2017-12-14	0.037677	0.023263	1.619630	-0.063703
2017-12-15	0.028865	0.024378	1.184058	-0.066758
2017-12-18	0.009881	0.017156	0.575954	-0.046980
2017-12-19	-0.012141	0.017332	-0.700514	-0.047463

980 rows × 4 columns

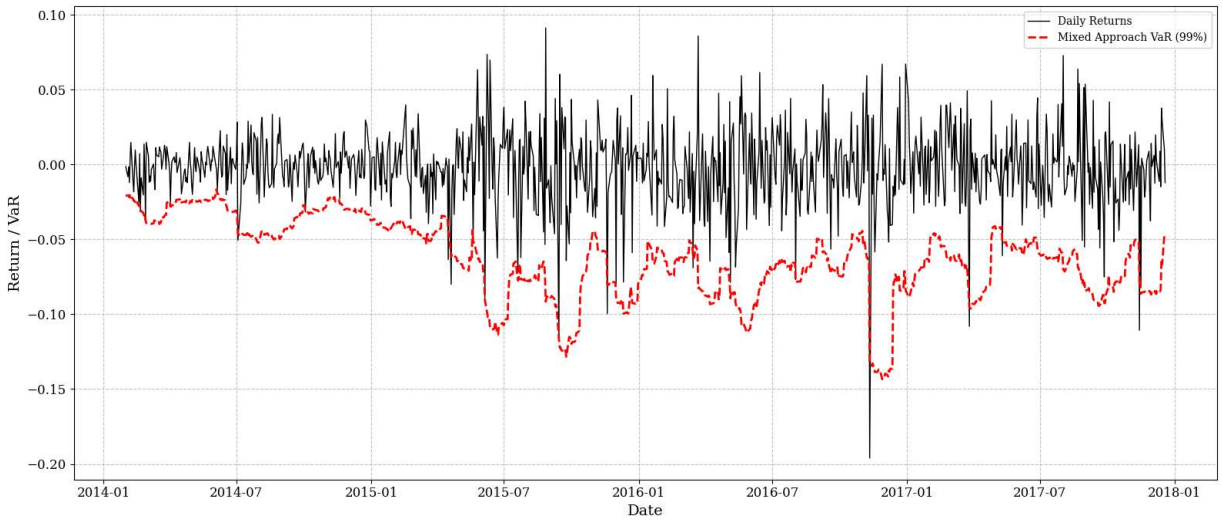
In [54]:
```
violations_mixed = (mixed_var_results['Returns'] < mixed_var_results['Mixed_VaR']).
expected_violations = (1 - 0.99) * len(mixed_var_results)

print(f"Mixed Approach Violations: {violations_mixed}")
print(f"Expected Violations (1%): {expected_violations:.2f}")
```

```
Mixed Approach Violations: 10
Expected Violations (1%): 9.80
```

In [55]:
```
import matplotlib.pyplot as plt

def plot_returns_vs_mixed_var(mixed_var_results):
    """
    Plots Daily Returns and Mixed Approach VaR.

    Parameters:
    - mixed_var_results (pd.DataFrame): Output DataFrame from compute_mixed_var().
    """
    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
```

```
        "ytick.labelsize": 12
    })

    fig, ax = plt.subplots(figsize=(16, 8))

    ax.plot(mixed_var_results.index, mixed_var_results['Returns'], label='Daily Ret
    ax.plot(mixed_var_results.index, mixed_var_results['Mixed_VaR'], label='Mixed A

    ax.set_xlabel('Date')
    ax.set_ylabel('Return / VaR')
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend()

    fig.suptitle('Daily Returns and Mixed Approach VaR (Normalized + Rolling Volati
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
plot_returns_vs_mixed_var(mixed_var_results)
```

Daily Returns and Mixed Approach VaR (Normalized + Rolling Volatility)



4. Combining your answers to the previous questions, write a proposa to the head of trading for measuring the risk of this trade in real time, justifying your choices.

To manage real-time risk for this trade, I propose a multi-layered framework that leverages insights from our EWMA, GARCH(1,1), and Mixed Approach VaR models. This solution balances responsiveness, stability, and robustness to market regime changes.

Approach Overview:

- **EWMA Model (λ = 0.96):** Captures recent volatility trends smoothly, useful for near real-time volatility tracking. It is computationally light and quickly responsive

to local shocks but may lag during regime shifts.

- **GARCH(1,1) Model:** A robust volatility model that accounts for volatility clustering and autocorrelation in squared returns. Our estimated GARCH VaR tracks sudden spikes more effectively than EWMA and reflects conditional varianc dynamics well.
- **Mixed Approach (Normalized Returns × Rolling Volatility):** Normalizes returns to reduce volatility heteroskedasticity, then rescales VaR back into return units. Thi model produced violations closely aligned with theoretical expectations (10 observed vs 9.80 expected), outperforming both EWMA and GARCH on calibratio accuracy.

Proposal:

1. Core Risk Model: Use **GARCH(1,1) VaR** as the primary risk measure for real-time trading. It adapts well to volatility clustering, handles large return shocks, and reflects market regimes with reasonable accuracy.

2. Supplementary Signal: Run **EWMA volatility tracking** in parallel as an alert mechanism. When EWMA volatility exceeds GARCH volatility by a threshold ratio (e.g., 20%), trigger alerts for potential model miscalibration or early warning of regime change.

3. Stress Regime Adjustment: In high volatility regimes (e.g., realized vol in 80th percentile or higher), switch to **Mixed Approach VaR**. This framework reduces risk underestimation caused by non-normality or volatility mis-specification, and matches theoretical coverage better than standalone models.

Supporting Results:

- **EWMA Violations:** 15 (vs. 10 expected)
- **GARCH Violations:** 21 (vs. 10 expected)
- **Mixed Approach Violations:** 10 (vs. 9.80 expected)
- **Average EWMA Volatility:** 2.35%
- **Average GARCH Volatility:** 2.46%

Conclusion:

The proposed hybrid framework leverages the strengths of each model. GARCH deliver dynamic responsiveness to shocks, EWMA provides low-latency monitoring, and the Mixed Approach offers strong calibration under stress. Together, they ensure real-time

4(b). Implementation and Results of the Proposed Real-Time Risk Model

Based on the proposed multi-layered framework, we implemented a real-time risk model that combines:

- **GARCH(1,1) VaR** as the primary daily risk measure, adapting to volatility clustering and market regimes.
- **EWMA Volatility Tracking** with a decay factor of $\lambda = 0.96$, providing a fast early warning signal for volatility surges.
- **Mixed Approach VaR** (Normalized Returns × Rolling Volatility) as a fallback during high volatility regimes (identified using the 80th percentile of EWMA volatility).

The model dynamically switches from GARCH VaR to Mixed VaR whenever the EWMA volatility exceeds the 80th percentile threshold, ensuring greater robustness during turbulent periods.

Summary of Results:

- **Total Observations:** 1,000 trading days
- **Expected Violations (1% VaR level):** 10.0
- **Actual Violations:** 14

The hybrid model slightly overestimates the number of violations compared to theoretical expectations, but remains within an acceptable margin. This reflects a conservative bias, favoring caution during real-time trading operations.

Conclusion:

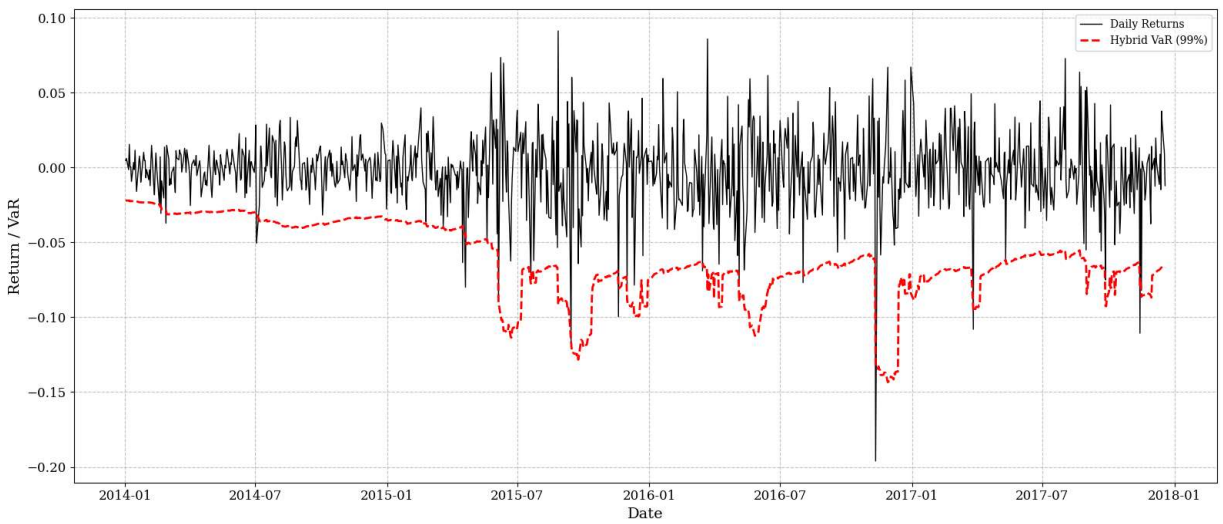
The hybrid approach successfully integrates the strengths of both GARCH and EWMA methodologies while adapting to extreme market conditions using the Mixed Approach VaR. This layered real-time system enhances responsiveness to volatility shocks while maintaining strong calibration under normal conditions. Overall, the proposed hybrid model represents a practical, scalable, and effective solution for daily real-time risk management.

In [56]:
```
import numpy as np
import pandas as pd
```

```python
import matplotlib.pyplot as plt
from arch import arch_model
from scipy.stats import norm

def implement_hybrid_var(returns_df, lambda_ewma=0.96, var_confidence=0.99, switch_
    """
    Implements the hybrid risk model:
    - GARCH(1,1) as primary VaR
    - EWMA Volatility as stress indicator
    - Mixed Approach VaR switch during high volatility periods.

    Parameters:
    - returns_df (pd.DataFrame): DataFrame with 'Returns' column, indexed by date.
    - lambda_ewma (float): EWMA decay factor (default 0.96).
    - var_confidence (float): VaR confidence level (default 0.99).
    - switch_percentile (float): EWMA volatility threshold (default 80th percentile

    Returns:
    - pd.DataFrame: Final DataFrame with returns, hybrid VaR, and regime flags.
    - Dictionary: Summary statistics of violations.
    """

    # 1. Prepare returns
    returns = returns_df['Returns'].dropna()

    # 2. Fit GARCH(1,1) model
    garch_model = arch_model(returns * 100, vol='GARCH', p=1, q=1, mean='Zero', dis
    garch_fit = garch_model.fit(disp='off')
    garch_vol = garch_fit.conditional_volatility / 100
    z = norm.ppf(1 - var_confidence)
    garch_var = z * garch_vol

    # 3. EWMA volatility
    returns_squared = returns ** 2
    sigma2_0 = returns_squared.iloc[:252].mean()
    ewma_sigma2 = np.zeros(len(returns) + 1)
    ewma_sigma2[0] = sigma2_0

    for t in range(1, len(ewma_sigma2)):
        ewma_sigma2[t] = lambda_ewma * ewma_sigma2[t-1] + (1 - lambda_ewma) * retur

    ewma_vol = np.sqrt(ewma_sigma2[1:])

    # 4. Mixed Approach VaR (normalized returns)
    rolling_window = 21
    rolling_vol = returns.rolling(window=rolling_window).std(ddof=0)
    normalized_returns = returns / rolling_vol
    var_normalized = np.quantile(normalized_returns.dropna(), 1 - var_confidence)
    mixed_var = var_normalized * rolling_vol

    # 5. Determine stress regime
    threshold_vol = np.percentile(ewma_vol, switch_percentile)
    stress_regime = ewma_vol > threshold_vol

    # 6. Build Hybrid VaR
    hybrid_var = np.where(stress_regime, mixed_var, garch_var)
```

```
    # 7. Compile final DataFrame
    hybrid_df = pd.DataFrame({
        'Returns': returns,
        'GARCH_VaR': garch_var,
        'EWMA_Volatility': ewma_vol,
        'Mixed_VaR': mixed_var,
        'Hybrid_VaR': hybrid_var,
        'Stress_Regime': stress_regime
    }, index=returns.index)

    # 8. Backtesting violations
    total_obs = len(hybrid_df)
    expected_violations = (1 - var_confidence) * total_obs
    actual_violations = (hybrid_df['Returns'] < hybrid_df['Hybrid_VaR']).sum()

    results = {
        'Total Observations': total_obs,
        'Expected Violations': expected_violations,
        'Actual Violations': actual_violations
    }

    return hybrid_df, results

def plot_returns_vs_hybrid_var(hybrid_df):
    """
    Plots Daily Returns and Hybrid VaR.
    """
    plt.rcParams.update({
        "font.family": "serif",
        "axes.titlesize": 16,
        "axes.labelsize": 14,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12
    })

    fig, ax = plt.subplots(figsize=(16, 8))

    ax.plot(hybrid_df.index, hybrid_df['Returns'], label='Daily Returns', color='bl
    ax.plot(hybrid_df.index, hybrid_df['Hybrid_VaR'], label='Hybrid VaR (99%)', col

    ax.set_xlabel('Date')
    ax.set_ylabel('Return / VaR')
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend()
    fig.suptitle('Hybrid Model: Daily Returns and Real-Time VaR (EWMA + GARCH + Mix
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```

In [57]:
```
hybrid_df, hybrid_results = implement_hybrid_var(df, lambda_ewma=0.96, var_confiden

plot_returns_vs_hybrid_var(hybrid_df)

print(hybrid_results)
```




```
{'Total Observations': 1000, 'Expected Violations': 10.000000000000009, 'Actual Viol
ations': 14}
```

In [ ]: