University of Texas at Austin

McCombs School of Business

RM 294: Optimization I

Project 1 Report – Linear Programming

Portfolio Optimization using Conditional Value-at-Risk (CVaR)

Team Members:

Melissa Cai Shi, Grace Lin, Abhigyan Pandya, Franco Salinas

September 28th, 2025

EXECUTIVE SUMMARY

This report details the construction and analysis of investment portfolios optimized to minimize Conditional Value-at-Risk (CVaR), a more robust risk measure than traditional Value-at-Risk (VaR). Using a linear programming model, we determined optimal portfolio allocations based on the 100 stocks from the NASDAQ index using historical daily returns from 2019 and evaluated their performance against 2020, a year marked by the COVID-19 pandemic with volatile market conditions.

Our analysis explored several risk management strategies. We found that a static portfolio optimized with 2019 data performed poorly out-of-sample in 2020's volatile market. We also determined that adjusting risk tolerance (β) and the choice between minimizing average daily versus maximum monthly risk significantly altered portfolio composition and resilience. A dynamic monthly re-optimization strategy using a rolling one-year window of historical data proved most effective at managing risk and more adaptive to changing market conditions compared to the static approach, but it also introduced portfolio instability, highlighting a key trade-off.

This analysis recommends a dynamic, monthly re-optimization approach as the most effective strategy for mitigating tail risk in a fluctuating market. However, it also suggests that for practical implementation, constraints must be considered to ensure portfolio stability and costs. This project provides a quantitative framework for building resilient, low-risk portfolios adaptable to changing market conditions.

METHODOLOGY OVERVIEW

Part 1: Data Cleaning and preparation

The first step in our analysis was to prepare the historical stock data. We utilized two datasets corresponding to the daily prices of the 100 stocks from the NASDAQ index for the years 2019 and 2020. This data, loaded from provided CSVs with dates parsed as the index, was converted into daily returns using the percentage change method. This transformation normalizes the data and represents the daily performance of each stock. Note that first row is dropped since there is no previous day to calculate the percentage change for the daily returns.

Additionally, the 'NDX' (NASDAQ 100 index) column was remove, as our goal was to optimize a portfolio of individual stocks, not include the index itself (correlation). This process yielded two clean dataframes: 2019 returns for trained model optimization and 2020 returns for out-of-sample testing.

```python
# Load 2019 and 2020 data
stocks_2019_df = pd.read_csv('stocks2019.csv', index_col=0, parse_dates=True)
stocks_2020_df = pd.read_csv('stocks2020.csv', index_col=0, parse_dates=True, date_format='%m/%d/%y')

# Rename the index column for formating
stocks_2019_df = stocks_2019_df.rename_axis('Date')
stocks_2020_df = stocks_2020_df.rename_axis('Date')

# Calculate returns; formula build in in python
returns_2019 = stocks_2019_df.pct_change().dropna()
returns_2020 = stocks_2020_df.pct_change().dropna()

# Remove NDX - this is the NASDAQ index, we want the individual stocks that make up the index/portfolio
returns_2019_noNDX = returns_2019.drop('NDX', axis=1)
returns_2020_noNDX = returns_2020.drop('NDX', axis=1)
```

Part 2: Static Base CVaR Optimization & Out-of-Sample Analysis

<u>Optimization Model (β =95%, R=0.02%)</u>

To construct the initial portfolio, we developed a linear programming model to minimize the 95% CVaR based on the 2019 daily return data. The objective is to find a portfolio allocation that minimizes the average loss on the worst 5% of trading days, subject to a set of constraints.

<u>Objective Function</u>

Minimize the 95% Conditional Value-at-Risk (CVaR).

$$CVaR_{\beta}(x) = min\ \alpha + \frac{1}{(1-\beta)q}\sum_{k=1}^{q} u_k$$

- $\alpha = VaR$
- $\beta =$ confidence level (e.g. 95%) - given
- $q =$ number of return days
- $u_k =$ dummy/helper variable representing the amount by which loss in given day k exceeds the VaR

<u>Decision variables</u>

- $\alpha =$ the 95% Value-at-Risk (VaR), representing the loss threshold
- $x_j =$ the weight allocated to each stock j in the portfolio.
- $u_k =$ a helper variable representing the "excess loss" for each trading day t (the loss that exceeds the VaR)

<u>Constraints</u>

1. <u>Minimum return:</u> The portfolio's average daily return must be at least R=0.02%

$$x^T * \bar{y} \geq R, \qquad \bar{y} = mean\ returns\ for\ each\ stock$$

2. <u>Excess loss constraint:</u> for each trading day k, we need to ensure that u_k correctly captures excess loss

$$-x^T * y_k - \alpha \leq u_k$$

3. <u>Non-negative excess loss:</u> for each excess loss variable cannot be negative

$$u_k \geq 0$$

4. <u>Full investment:</u> The weights of all stocks must sum to 1

$$x_1 + x_2 + \cdots + x_{100} = 1$$

5. <u>No short selling:</u> The weight of any individual stock cannot be negative

$$x_j \geq 0, \qquad j = 1,..,100$$

```
# ----------  Part 2a: Base CVaR Optimization ----------

# Set Inputs
beta = 0.95
R = 0.0002

# Get number of stocks variable; NASDAQ consist of 100 stocks
num_stocks_2019 = returns_2019_noNDX.shape[1]

# Get number of days variable q; 250 trading days
q_num_days_2019 = len(returns_2019_noNDX)

# ----------  β-CVaR Minimization (β=0.95, R=0.02%) ----------

# Create the Gurobi model
CVaR_model_2019 = gp.Model("CVaR_Optimization_2019")

# Add/Define decision variables
x_2019 = CVaR_model_2019.addMVar(num_stocks_2019, lb=0.0, name="weights") # here set non-negative contraint
alpha_2019 = CVaR_model_2019.addMVar(1, name="VaR")
u_2019 = CVaR_model_2019.addMVar(q_num_days_2019, lb=0.0, name="excess_loss") # here set non-negative contraint

# Set the objective function: Minimize CVaR
CVaR_model_2019.setObjective(alpha_2019 + (1 / ((1 - beta) * q_num_days_2019)) * u_2019.sum(), gp.GRB.MINIMIZE)

# Add constraints (@ is for matrix operation)
CVaR_model_2019.addConstr(x_2019.sum() == 1, name="weights_sum_one_constraint")
CVaR_model_2019.addConstr(returns_2019_noNDX.mean().values @ x_2019 >= R, name="min_return_constraint")
CVaR_model_2019.addConstr(-returns_2019_noNDX.values @ x_2019 - alpha_2019 <= u_2019, name="excess_loss_constraint")

# Solve the model
CVaR_model_2019.Params.OutputFlag = 0 # Suppress Gurobi output
CVaR_model_2019.optimize()
```

<u>Insights</u>

The model was first solved using 2019 data to create an optimal "in-sample" portfolio. This static portfolio was then tested against 2020's market data to evaluate its "out-of-sample" performance. The results are summarized below, comparing the optimized portfolio against the NASDAQ 100 (NDX) index as a benchmark.

	Metric	Optimized Portfolio	NDX Index
2019 (In-Sample)	CVaR	1.11%	2.44%
	VaR	0.85%	1.60%
2020 (Out-of-Sample	CVaR	4.58%	5.59%
	VaR	2.53%	3.90%

1. <u>In-Sample:</u> The optimization was highly effective within the 2019 dataset. The optimized portfolio's CVaR of 1.11% was less than half that of the NDX index (2.44%), which demonstrates the model's ability to significantly reduce tail risk. Looking at the graph, you can see that the optimized portfolio is smoother and less volatile to the upsides/downsides than the NDX but following the same trend movement.

2. <u>Concentrated Portfolio:</u> The optimal 2019 portfolio was not broadly diversified, with only 13 of the 100 available stocks receiving an allocation. Over 56% of the capital was allocated to just two stocks: XEL (30.4%) and CHTR (26.5%). This high concentration suggests these assets had very strong historical risk-return profiles in 2019, but it also introduces concentration risk.

3. <u>Static Model Failure in Volatile Markets:</u> The primary finding is the dramatic divergence between in-sample and out-of-sample performance. The portfolio's CVaR

quadrupled from 1.11% in 2019 to 4.58% in 2020. This underscores a critical weakness of static optimization: a portfolio optimized for a calm market period (2019) is not prepared for a systemic shock and high volatility (the 2020 COVID-19 market).

4. Relative Outperformance: Despite its significant underperformance relative to expectations, the optimized portfolio was still less risky than the benchmark in 2020. Its out-of-sample CVaR of 4.58% was better than the NDX index's 5.59%, indicating that the stock selection logic provided some downside protection even during the market crash, but not as much as what we see in 2019 cut by half. Also, note when looking at the graph the optimized portfolio with 2019 weights deviates from the NDX index at the end of 2020.

The significant difference between the 2019 in-sample and 2020 out-of-sample CVaR reveals a core challenge in financial modeling: ***market non-stationarity***. Because the statistical properties (mean and variance) of stock returns change unpredictably, a portfolio optimized for one period does not remain optimal for the next. This result demonstrates that a ***static strategy is insufficient for managing risk in dynamic market environments***, especially during periods of high volatility/market stress.

```python
# --- Display Optimization Results ---
if CVaR_model_2019.status == gp.GRB.OPTIMAL:
    print("--- In-sample Optimization Results (2019 Data) ---")

    # Extract and print VaR and CVaR
    optimal_VaR_2019 = alpha_2019.X[0]
    optimal_CVaR_2019 = CVaR_model_2019.ObjVal
    print(f"In-sample Optimal VaR (2019): {optimal_VaR_2019:.6f}")
    print(f"In-sample Optimal CVaR (2019): {optimal_CVaR_2019:.6f}\n")

    # Extract, filter, and print non-zero weights
    weights_2019 = pd.Series(x_2019.X, index=returns_2019_noNDX.columns)
    non_zero_weights_2019 = weights_2019[weights_2019 > 1e-6] # Use a small threshold to avoid floating point noise

    print("Portfolio Composition (Non-Zero Weights):")
    # Sort weights in descending order for better presentation
    for stock, weight in non_zero_weights_2019.sort_values(ascending=False).items():
        print(f"  {stock}: {weight:.2%}")


    # ---------- Part 2b: Re-evaluating the portfolio with 2020 data (Out-of-Sample Test) ----------
    # Use the optimal weights from 2019 to test on 2020 data
    # Calculate portfolio returns for 2020 using 2019 weights
    portfolio_returns_2020 = returns_2020_noNDX @ weights_2019
    # Calculate portfolio losses for 2020
    portfolio_losses_2020 = -portfolio_returns_2020

    # Calculate out-of-sample VaR and CVaR
    out_of_sample_VaR_2020 = portfolio_losses_2020.quantile(beta)
    out_of_sample_CVaR_2020 = portfolio_losses_2020[portfolio_losses_2020 >= out_of_sample_VaR_2020].mean()

    print("\n--- Out-of-Sample Performance (2020 Data) ---")
    print(f"Out-of-Sample VaR (2020): {out_of_sample_VaR_2020:.6f}")
    print(f"Out-of-Sample CVaR (2020): {out_of_sample_CVaR_2020:.6f}")


    # ---------- Part 2c: Calculate CVaR for the NDX Index ----------
    print("\n--- NDX Index Performance ---")

    # --- 2019 NDX Performance ---
    # Isolate NDX returns for 2019 and get losses
    ndx_losses_2019 = -returns_2019['NDX']
    # Calculate NDX VaR
    ndx_VaR_2019 = ndx_losses_2019.quantile(beta)
    # Calculate NDX CVaR
    ndx_CVaR_2019 = ndx_losses_2019[ndx_losses_2019 >= ndx_VaR_2019].mean()
    print("\n--- 2019 ---")
    print(f"NDX VaR (2019): {ndx_VaR_2019:.6f}")
    print(f"NDX CVaR (2019): {ndx_CVaR_2019:.6f}")

    # --- 2020 NDX Performance ---
    ndx_losses_2020 = -returns_2020['NDX']
    ndx_VaR_2020 = ndx_losses_2020.quantile(beta)
    ndx_CVaR_2020 = ndx_losses_2020[ndx_losses_2020 >= ndx_VaR_2020].mean()
    print("\n--- 2020 ---")
    print(f"NDX VaR (2020): {ndx_VaR_2020:.6f}")
    print(f"NDX CVaR (2020): {ndx_CVaR_2020:.6f}")
```

Part 3: Impact of Beta (Risk Tolerance) on Portfolio Construction

To understand how risk tolerance affects the portfolio, we re-ran the optimization using different confidence levels (β) of 0.90, 0.95, and 0.99. A higher β indicates a more risk-averse stance, focusing the optimization on more extreme, less frequent tail-risk events.

As the confidence level β increases, the optimal CVaR of the portfolio also increases. This is because a higher β forces the model to average losses from a more extreme/higher-loss tail of the distribution.

- CVaR (β = 0.90): 0.89%
- CVaR (β = 0.95): 1.11%
- CVaR (β = 0.99): 1.25%

The most significant impact of changing β is on the ***portfolio's composition and diversification***. As risk aversion increases (higher β), the portfolio becomes more concentrated in fewer assets.

- At β = 0.90, the portfolio is the most diversified, with 16 stocks holding non-zero weights. The top holdings are CHTR (24.1%) and XEL (21.3%).
- At β = 0.95, the portfolio becomes slightly more concentrated, holding 13 stocks. The weights in XEL (30.4%) and CHTR (26.5%) increased.
- At β = 0.99, the portfolio is highly concentrated in just 9 stocks. The allocation to XEL surges to 44.7%, indicating that the model identifies this stock as the most effective for mitigating the worst 1% of potential losses.

The cumulative return plot for 2019 shows that despite the significant differences in composition, the in-sample performance of the portfolios optimized with different β values was very similar. No single portfolio consistently outperformed the others, and all tracked the general movement of the NDX index, but with less volatility.

This analysis reveals the trade-off of increasing risk aversion (a higher β) and a less diversified portfolio that is theoretically better protected against extreme events. However, this concentration also makes the portfolio more sensitive to the performance of its few holdings. The choice of β is therefore a critical decision that must align with the investor's specific risk appetite and view on diversification.

```
# Define the beta values to test
beta_values = [0.90, 0.95, 0.99]
cvar_results = []
portfolio_compositions = {}
portfolio_returns_by_beta = {} # To store daily returns for each beta portfolio

# Loop through each beta value
for beta_val in beta_values:
    # Create the Gurobi model for each beta
    beta_model = gp.Model(f"CVaR_Optimization_Beta_{beta_val}")

    # Add/Define decision variables
    x = beta_model.addMVar(num_stocks_2019, lb=0.0, name="weights")
    alpha = beta_model.addMVar(1, name="VaR")
    u = beta_model.addMVar(q_num_days_2019, lb=0.0, name="excess_loss")

    # Set the objective function: Minimize CVaR
    beta_model.setObjective(alpha + (1 / ((1 - beta_val) * q_num_days_2019)) * u.sum(), gp.GRB.MINIMIZE)

    # Add constraints
    beta_model.addConstr(x.sum() == 1, name="weights_sum_one_constraint")
    beta_model.addConstr(returns_2019_noNDX.mean().values @ x >= R, name="min_return_constraint")
    beta_model.addConstr(-returns_2019_noNDX.values @ x - alpha <= u, name="excess_loss_constraint")

    # Suppress Gurobi output and solve the model
    beta_model.Params.OutputFlag = 0
    beta_model.optimize()

    # Store results if optimization is successful
    if beta_model.status == gp.GRB.OPTIMAL:
        cvar_results.append(beta_model.ObjVal)
        weights = pd.Series(x.X, index=returns_2019_noNDX.columns)
        non_zero_weights = weights[weights > 1e-6]
        portfolio_compositions[beta_val] = non_zero_weights

        # Calculate and store daily returns for this portfolio
        daily_returns = returns_2019_noNDX @ weights
        portfolio_returns_by_beta[beta_val] = daily_returns
    else:
        print(f"Optimization for Beta = {beta_val} was not successful.")
        cvar_results.append(None)
        portfolio_compositions[beta_val] = pd.Series()

# --- Display Beta Sensitivity Results ---
print("\n--- Impact of Beta on Portfolio Composition (2019 Data) ---")
for beta_val, composition in portfolio_compositions.items():
    print(f"\n--- Beta = {beta_val} ---")
    if cvar_results[beta_values.index(beta_val)] is not None:
        print(f"Optimal CVaR: {cvar_results[beta_values.index(beta_val)]:.6f}")
        print("Portfolio Composition (Non-Zero Weights):")
        for stock, weight in composition.sort_values(ascending=False).items():
            print(f"  {stock}: {weight:.2%}")
    else:
        print("Could not find an optimal solution.")
```

Part 4: CVaR Optimization on Monthly Returns

In this section we focused on minimizing the maximum monthly β-CVaR. Our approach works as follows:

1. **Monthly CVaR Definition**: For each month, we calculate β-CVaR using only that month's daily returns. Each monthly β-CVaR represents the expected daily loss in the worst 5% of trading days within that specific month.
2. **Min-Max Optimization**: Instead of minimizing the average β-CVaR across all days, we minimize the worst-case monthly performance. Mathematically, we introduce a variable T that must be greater than or equal to each month's β-CVaR. Then we minimize T.

   See the mathematic formulation below:

   Decision Variables:

   - $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ portfolio weights
   - α_m VaR approximation for month m, $m = 1, 2, \ldots, 12$
   - u_k auxiliary variables, $k = 1, 2, \ldots, q$
   - T maximum monthly CVaR (objective variable)

   Objective:

$$\min \quad T$$

   Constraints:

   1. $\sum_{j=1}^{n} x_j = 1$ (budget constraint)
   2. $\sum_{j=1}^{n} \mu_j x_j \geq R$ (minimum expected return)
   3. $u_k \geq -\sum_{j=1}^{n} Y_{k,j} x_j - \alpha_{m(k)}, \quad k = 1, 2, \ldots, q$ (tail constraints)
   4. $T \geq \alpha_m + \frac{1}{(1-\beta) \cdot q_m} \sum_{k \in \mathcal{M}_m} u_k, \quad m = 1, 2, \ldots, 12$ (monthly CvaR<=T)

$$u_k \geq 0, \quad k = 1, 2, \ldots, q$$

$$x_j \geq 0, \quad j = 1, 2, \ldots, n$$

   Where:

   - $m(k) =$ month index for day k
   - $\mathcal{M}_m =$ set of day indices belonging to month m
   - $q_m =$ number of trading days in month m
   - $Y_{k,j} =$ return of asset j on day k
   - $\mu_j =$ expected return of asset j
   - u_k auxiliary variables, $k = 1, 2, \ldots, q$

See the mathematical formulation coded below:

```python
# --- prepare assets and arrays ---
assets = list(returns_2019_noNDX.columns)  # Get asset list from column names
Y = returns_2019_noNDX.to_numpy()   # shape (q_total, n)
q_total, n = Y.shape
mu = returns_2019_noNDX.mean().to_numpy()

print(f"Data shape: {Y.shape}")
print(f"Number of assets: {n}")
print(f"Number of trading days: {q_total}")

# --- map each row to a month period (periods preserve order) ---
dates = pd.to_datetime(returns_2019_noNDX.index)
month_labels = dates.to_period('M')                # e.g. Period('2019-01', 'M')
months = month_labels.unique().tolist()            # list of months in order

# mapping month -> row indices (0-based into Y)
month_to_rows = {m: np.where(month_labels == m)[0].tolist() for m in months}
q_m = {m: len(month_to_rows[m]) for m in months}

print(f"Months in data: {[str(m) for m in months]}")
print(f"Days per month: {[q_m[m] for m in months]}")
```

```python
# --- Build Gurobi model ---
model = gp.Model("min_max_monthly_cvar")
model.Params.OutputFlag = 0

# decision vars
x = model.addVars(n, lb=0.0, name="x")                    # asset weights (n vars)
alpha = {m: model.addVar(lb=-gp.GRB.INFINITY, name=f"alpha_{str(m)}") for m in months}
u = model.addVars(q_total, lb=0.0, name="u")              # one slack per day
T = model.addVar(lb=-gp.GRB.INFINITY, name="T")             # maximum monthly CVaR to minimize

# budget and mean floor
model.addConstr(gp.quicksum(x[j] for j in range(n)) == 1.0, name="budget")
model.addConstr(gp.quicksum(mu[j] * x[j] for j in range(n)) >= R, name="mean_floor")

# tail constraints: for each day k use the alpha of that day's month
for k in range(q_total):
    m_label = month_labels[k]
    # -Y[k,:]@x - alpha_m - u_k <= 0  <=> u_k >= L_k(x) - alpha_m
    model.addConstr(
        -gp.quicksum(Y[k, j] * x[j] for j in range(n)) - alpha[m_label] - u[k] <= 0.0,
        name=f"tail_day_{k}"
    )

# monthly CVaR definition and link to T
for m_label in months:
    rows = month_to_rows[m_label]
    qm = q_m[m_label]
    if qm <= 0:
        raise RuntimeError(f"Month {m_label} has no observations.")
    cvar_coef = 1.0 / ((1.0 - beta) * qm)
    # alpha_m + cvar_coef * sum_{k in rows} u_k <= T
    model.addConstr(
        alpha[m_label] + cvar_coef * gp.quicksum(u[k] for k in rows) <= T,
        name=f"cvar_le_T_{str(m_label)}"
    )
```

```python
# objective: minimize T (the maximum monthly CVaR)
model.setObjective(T, gp.GRB.MINIMIZE)

# solve
print("Solving min-max monthly CVaR optimization...")
model.optimize()

if model.Status != gp.GRB.OPTIMAL:
    raise RuntimeError(f"Gurobi did not find an optimal solution (status {model.Status}).")

# extract weights
w_part4 = pd.Series([x[j].X for j in range(n)], index=assets, name="weight")
part4_objective = model.ObjVal

print(f"Optimization completed successfully!")
print(f"Objective value (max monthly CVaR): {part4_objective:.6f}")
```

Comparison with portfolio that minimizes the daily average CVaR using the 2019 data:

This method where we focus on the maximum monthly CVaR rather than the average daily CvaR protects against a particularly bad month. Even if the average risk is slightly higher, no month experiences extreme tail losses relative to other months as we create a more defensive portfolio.

The picture below shows the CVaR value calculated from our optimized portfolios using the daily average method (labeled as part 2) and the minimum maximum monthly method. We computed the CVaR separately for each month for both portfolios. You can see that the conservative method decreases the worse month CVaR by 22.2% but at the expense of an increase in Overall CVaR of 12.2%. This method minimizes the worse losses from abnormal months like August 2019 by increasing the overall expected loss daily.

We also see that the ***monthly method provides more stable risk*** as we see a reduction of 23.67% in volatility relative to the daily approach. With the monthly min max approach, we create a portfolio that reduces the risk of catastrophic losses during hard period. The portfolio calculated using the conservative method also is expected to have better daily returns (0.13% relative to 0.12%). We see that for a higher risk overall we also have a higher expected return.

You can find all the values used to compute the differences in the table below.

	Objective Value	Max Monthly CVaR	CVaR Std Dev	Expected Return
Avg Daily	1.11%	1.60%	0.30%	0.12%
Conservative Monthly (Min-Max)	1.24%	1.24%	0.23%	0.13%

For the monthly analysis, we see that the weights are concentrated on less companies (9) than with the average daily method. This introduces more concentration risk than in the average daily portfolio. Similar to the daily average method most capital was allocated to XEL (45%) and CHTR (24.3%). Although this method showed even a higher concentration for this two companies (69%). This high concentration supports our hypothesis that these assets had very strong historical risk-return profiles in 2019.

Like the daily average portfolio, the conservative optimized portfolio's CVaR of 1.2% was less than of the NDX index (2.44%), which demonstrates the model's ability to reduce tail risk. Looking at the graph, you can see that the ***monthly optimized portfolio is also, smoother and less volatile to the upsides/downsides*** than the NDX and the average daily portfolio. It also produces more cumulative return during December and January than our daily average method and the NDX index.

Part 5: Dynamic Monthly Re-optimization Strategy

To address the shortcomings of the static portfolio, we implemented a dynamic strategy. At the beginning of each month in 2020, we re-optimized the portfolio using a rolling 12-month window of historical daily returns. This approach allows the portfolio to adapt to changing market conditions.

Summary of Monthly Realized CVaRs:

- Mean: 2.94%
- Standard deviation: 2.09%
- Minimum: 0.81% (January 2020)
- Maximum: 8.93% (March 2020)

The dynamic strategy proved far ***more effective at managing risk*.** Its mean monthly CVaR of 2.94% is significantly lower than the static portfolio's out-of-sample CVaR of 4.58%. While the risk spiked in March 2020 (8,93%) during the market crash due to Covid-19, the strategy adapted in subsequent months, keeping the average risk well below the static alternative.

The cumulative performance graph provides a clear visual confirmation of the dynamic strategy's superiority. It not only generated a higher overall return by the end of 2020 but, more importantly, it experienced a significantly smaller drawdown during the March crash compared to both the static portfolio and the NDX index. This resilience is a direct result of its ***ability to re-optimize and adapt away from escalating risks***. Therefore, reoptimizing the portfolio across months is a better approach from both a risk and performance perspective.

The standard deviation of 2.09% is relatively high compared to the mean CVaR of 2.94%. This indicates that while the ***strategy was effective on average, the realized risk was not stable, fluctuating significantly from month to month.*** This volatility underscores the unpredictable nature of 2020's market and the challenge of maintaining consistent risk levels.

```python
# optimizer function to get weights
def solve_cvar_weights(train_returns_df, beta_val=beta, R_floor=R):
    """Solve the standard CVaR LP on a training DataFrame."""
    Yt = train_returns_df.to_numpy()
    qt, nt = Yt.shape
    mut = train_returns_df.mean().to_numpy()

    m = gp.Model("cvar_monthly_train")
    m.Params.OutputFlag = 0

    x = m.addMVar(nt, lb=0.0, name="weights")
    a = m.addMVar(1, lb=-gp.GRB.INFINITY, name="VaR")
    u = m.addMVar(qt, lb=0.0, name="excess_loss")

    m.addConstr(x.sum() == 1)
    m.addConstr(mut @ x >= R_floor)
    m.addConstr(-Yt @ x - a <= u)

    coef = 1 / ((1 - beta_val) * qt)
    m.setObjective(a + coef * u.sum(), gp.GRB.MINIMIZE)
    m.optimize()

    if m.status == gp.GRB.OPTIMAL:
        return pd.Series([var.X for var in x.tolist()], index=train_returns_df.columns)
    else:
        return pd.Series(dtype='float64')

# --- Helper Function to Calculate Empirical VaR and CVaR ---
def var_cvar(wts, returns_df, beta_val=beta):
    """Calculates empirical VaR and CVaR for a given set of weights and returns."""
    if wts.empty or returns_df.empty:
        return np.nan, np.nan
    portfolio_returns = returns_df[wts.index].to_numpy() @ wts.to_numpy()
    losses = -portfolio_returns
    v = np.quantile(losses, beta_val)
    c = losses[losses >= v - 1e-12].mean()
    return float(v), float(c)
```

```python
# --- Main Dynamic Analysis Loop ---
# Combine all returns for the rolling window
all_returns = pd.concat([returns_2019_noNDX, returns_2020_noNDX]).sort_index()
analysis_year = 2020

monthly_results = []
dynamic_portfolio_weights = {} # Renamed for consistency with Part 6
dynamic_daily_returns_list = [] # To store daily returns for the cumulative plot
months_2020 = returns_2020.index.to_period("M").unique().tolist()

for mth in months_2020:
    # Define the rolling window
    end_of_prev_month = (mth.start_time - pd.offsets.Day(1))
    start_12m = end_of_prev_month - DateOffset(years=1) + pd.offsets.Day(1)
    train_mask = (all_returns.index >= start_12m) & (all_returns.index <= end_of_prev_month)
    train_df = all_returns.loc[train_mask, assets]

    # Solve for this month's portfolio
    w_m = solve_cvar_weights(train_df)

    # Store weights for Part 6 analysis
    month_key = end_of_prev_month.strftime('%Y-%m')
    dynamic_portfolio_weights[month_key] = w_m

    # Evaluate on the actual returns for month mth
    test_mask = (returns_2020_noNDX.index.to_period("M") == mth)
    test_df = returns_2020_noNDX.loc[test_mask, assets]
    _, cvar_m = var_cvar(w_m, test_df)

    monthly_results.append({
        "month": str(mth),
        "cvar": float(cvar_m)
    })

    # Calculate and store daily returns for the cumulative performance plot
    if not w_m.empty and not test_df.empty:
        daily_p_returns = test_df[w_m.index] @ w_m
        dynamic_daily_returns_list.append(daily_p_returns)


# --- Display Results ---
monthly_cvar_df = pd.DataFrame(monthly_results).set_index("month")
print("\n--- Monthly Realized Out-of-Sample CVaR (2020) ---")
print(monthly_cvar_df)

print("\n--- Summary of Monthly CVaRs ---")
print(f"Mean  : {monthly_cvar_df['cvar'].mean():.6f}")
print(f"Std   : {monthly_cvar_df['cvar'].std(ddof=1):.6f}")
print(f"Min   : {monthly_cvar_df['cvar'].min():.6f}  (in month {monthly_cvar_df['cvar'].idxmin()})")
print(f"Max   : {monthly_cvar_df['cvar'].max():.6f}  (in month {monthly_cvar_df['cvar'].idxmax()})")
```

Part 6: Portfolio Stability Analysis

While the dynamic strategy demonstrated superior risk management, its practical feasibility depends on its stability. ***A strategy that requires constant, drastic changes incurs high transaction costs and may be difficult to implement***. To measure this, we analyzed the month-to-month changes in portfolio weights, defining a "stability violation" as any change in a single stock's weight greater than 5%.

The analysis clearly shows that the **dynamic portfolio is not stable**. Over the course of 2020, there were a total of 43 instances where a stock's weight was adjusted by more than 5% in a single month. The instability peaked in March (9 violations) and August (8 violations). The March re-optimization, reacting to the COVID-19 crash, resulted in the largest single-asset change of the year: the weight of CTXS was drastically increased by over 45%. This shows the model making a massive defensive shift in response to the market turmoil. Conversely, the high number of violations in August likely reflects the model reacting to the sharp tech-led market recovery. ***Model is overfitting to noise in the most recent data rather than capturing true market trends.***

Proposed Constraint to Enforce Stability

The portfolio's instability highlights a critical trade-off between risk optimization and transaction costs. To create a more practical and stable portfolio, we can add stability constraints to the optimization model. For each rebalancing period, we would add an additional linear constraint for every stock in the portfolio:

$$\left|x_{ik,t} - x_{ik,t-1}\right| \leq 0.05$$

We can limit how much the weight of any single stock j can change from the previous month (t-1) to the current month (t).

In Gurobi, we would implement this as two linear constraints for each stock j:

1. $weight_{j,t} - weight_{j,t-1} \leq 0.05$
2. $weight_{j,t} - weight_{j,t-1} \geq -0.05$

These two constraints would force the model to find the optimal portfolio while ensuring that no single asset's weight can change by more than our defined 5% threshold. It reduces transaction costs and makes the portfolio more stable, but it also restricts the model's ability to react to new information, which could slightly increase risk compared to the unconstrained dynamic model.

```
if 'dynamic_portfolio_weights' in locals() and dynamic_portfolio_weights:

    # Order the months chronologically to ensure correct difference calculation
    ordered_months = sorted(dynamic_portfolio_weights.keys())

    # Create a DataFrame from the weights dictionary. Rows=assets, cols=months
    W = pd.DataFrame({m: dynamic_portfolio_weights[m] for m in ordered_months}).fillna(0)

    # Calculate the month-to-month absolute change in weights
    W_diff = W.diff(axis=1).abs()

    # Define the threshold for what constitutes a "large" change (e.g., 5 percentage points)
    STABILITY_THRESHOLD = 0.05

    # Identify violations, excluding the first month which has no prior data to compare
    violations = (W_diff.iloc[:, 1:] > STABILITY_THRESHOLD)

    # Count the number of violations for each month
    violations_count_by_month = violations.sum(axis=0)

    print(f"\nAnalysis of weight changes greater than {STABILITY_THRESHOLD:.0%}:")

    stab_df = pd.DataFrame({
        "month": violations.columns,
        "num_violations": violations_count_by_month.values
    }).set_index("month")

    print(stab_df)

    total_violations = int(violations_count_by_month.sum())
    print(f"\nTotal asset-level violations across {analysis_year}: {total_violations}")
```

CONCLUSION/FINAL RECOMMENDATION

This analysis confirms that a static, buy-and-hold strategy is inadequate for managing tail risk in a volatile and non-stationary market. A dynamic, monthly re-optimization strategy using a CVaR minimization model is demonstrably superior, reducing the average realized risk by over a third compared to the static approach during the turbulent 2020 period.

However, the unconstrained dynamic model is too unstable for practical implementation due to high turnover. Therefore, the final recommendation is to adopt a ***constrained dynamic CVaR optimization strategy***. By adding stability constraints that limit the maximum allowable change in any single asset's weight each month, we can strike an effective balance, creating a portfolio that is both adaptive to changing market conditions and cost-efficient to manage.