Generative LSTM + CVaR-Constrained Portfolio Optimization (Notebook & Flask App)

Kumar Shivam 220562

November 13, 2025

GitHub Repository: https://github.com/kshivamiitk/CS787-GEN-AI-PROJECT

Abstract

This report summarizes a system that (1) learns short-term multivariate return dynamics using an LSTM, (2) generates many plausible future return scenarios, and (3) finds portfolio allocations that maximize expected return while guaranteeing a user-specified downside bound (CVaR). The presentation mixes intuitive explanation and a few compact formulas to make the key ideas precise without heavy mathematics. A final figure compares predicted vs. realized H-day profit across several risk limits.

1 Executive summary

I build a reproducible pipeline and a small web app that lets a user specify:

- an investment horizon H (e.g. 21 trading days),
- a loss tolerance (CVaR bound),
- a confidence level α (e.g. 0.95).

The system trains a multivariate LSTM on historical daily returns, creates many synthetic forward paths by combining model predictions with empirically observed model errors, and solves a convex optimization to produce an allocation that maximizes expected H-day return while keeping expected losses in the worst $(1-\alpha)$ fraction of scenarios below the specified limit. The Flask app reports the allocation and the expected profit percentage for the chosen inputs.

2 Data and preprocessing (informal)

I fetched adjusted close prices for a basket of n stocks (10–12 tickers) over multiple years. From prices $P_{t,j}$ I computed simple daily returns

$$r_{t,j} = \frac{P_{t,j} - P_{t-1,j}}{P_{t-1,j}}.$$

I cleaned, forward/back-filled small gaps, aligned all tickers on the same calendar, and split the chronology into a training portion (used to fit the model) and a validation portion (used to estimate prediction errors).

Before training I standardized each asset's returns using the training mean and standard deviation; these scaling parameters are saved so all later operations revert to the original return units when needed.

3 Predictive model (LSTM) — intuition and form

An LSTM is trained to predict the next-day multivariate return vector from the last L days. Formally, if the input sequence at time t is $X_t = [r_{t-L+1}, \ldots, r_t]$, the trained model f_θ outputs a one-step prediction

$$\hat{r}_{t+1} = f_\theta(X_t).$$

I trained by minimizing mean squared error on standardized returns (chronological training, no shuffling). The model captures temporal patterns and cross-asset relationships, but it is not assumed to be perfect — its prediction errors matter and are explicitly used in scenario generation.

4 Residuals (why they matter)

On the validation set we record one-step residuals

$$\varepsilon_{t+1} = r_{t+1} - \hat{r}_{t+1},$$

as vectors across assets. These residuals capture the real-world unpredictability not explained by the model (fat tails, cross-asset co-movements). I keep the empirical set of residual vectors and use bootstrap sampling from them when simulating future paths; this preserves realistic tails and dependence structure.

5 Synthetic scenario generation (practical recipe)

At a decision time t_0 we build many (M) forward scenarios of length H days as follows (high level):

1. Seed the simulation with the most recent L real returns.
2. For each day $h = 1, \ldots, H$: predict the next-day return using the LSTM, unscale to original units, then add a residual vector sampled (with replacement) from the validation residuals.
3. Clamp daily returns to a safe range (e.g. $\pm 20\%$) to avoid unrealistic jumps.
4. At the end of H days compute per-asset cumulative return for that scenario:

$$c_j^{(m)} \;=\; \prod_{h=1}^{H} (1 + r_{t_0+h,j}^{(m)}) - 1.$$

Collect the scenario cumulative returns into a matrix $C \in \mathbb{R}^{M \times n}$ (rows are scenarios, columns are assets).

6 Risk measure and the optimization problem (simple math)

I denoted portfolio weights by $w \in \mathbb{R}^n$ (long-only, $\sum_j w_j = 1$, $w_j \geq 0$). Portfolio H-day return in scenario m is

$$R^{(m)}(w) \;=\; w^\top c^{(m)}.$$

I measured downside risk using **CVaR** at level α. A compact and intuitive expression for CVaR is

$$\mathrm{CVaR}_\alpha(L) \;=\; \min_{t \in \mathbb{R}} \left\{ t \;+\; \frac{1}{1-\alpha} \mathbb{E}[(L - t)_+] \right\},$$

where L denotes loss and $(x)_+ = \max(0, x)$. In my sample-based setting $\mathbb{E}$ is replaced by the empirical average over the M simulated scenarios.

The portfolio optimization solved by the system is therefore:

$$\begin{aligned} \text{maximize}_w \quad & \mathbb{E}_m[\,R^{(m)}(w)\,] \quad \text{(sample mean over scenarios)} \\ \text{subject to} \quad & \text{CVaR}_\alpha\big(-R(w)\big) \le L_{\text{limit}}, \\ & \sum_j w_j = 1, \quad w_j \ge 0. \end{aligned}$$

This is solved via the Rockafellar–Uryasev sample-average formulation, which is convex and tractable with solvers such as ECOS/SCS/OSQP.

7 Feasibility check and practical behavior

Before returning an allocation for a requested loss limit, the system computes the minimum achievable CVaR under the long-only simplex constraint. If the requested loss limit is below this minimum, no feasible allocation exists (the app reports this and suggests the minimum achievable loss or a conservative allocation). This avoids returning allocations that would secretly violate the requested guarantee.

8 Backtesting and comparison

I performed a walk-forward style evaluation: at multiple historical decision dates I computed an allocation from the synthetic scenarios generated at that date, then evaluated how that allocation actually performed over the subsequent H real days. For each chosen loss limit (sweep a grid, e.g. 1% to 12%) we average:

- the model-predicted expected H-day profit (from scenarios), and
- the realized average H-day profit (from the real forward windows),

but only across decision dates where the requested CVaR bound was feasible. The final figure plots these two curves versus the loss limit — this is the main diagnostic used to judge calibration and usefulness.

9 Implementation notes

- **Model:** PyTorch LSTM (multivariate output), trained on standardized returns.
- **Residuals:** stored on validation set; bootstrap sampling preserves empirical tails.
- **Generation:** autoregressive with model+residuals, clamped per-day moves.
- **Optimization:** CVaR SAA via CVXPy; try ECOS then SCS/OSQP if needed.
- **App:** Flask endpoint accepts horizon, loss limit, alpha; returns allocation and expected profit percentage (predicted_expected$\times$100).

10 Results (visual)

Below is the comparison plot produced by the notebook. The horizontal axis shows the CVaR loss limit and the vertical axis shows the H-day portfolio return (fraction). Replace the image path with the PNG exported by the notebook.

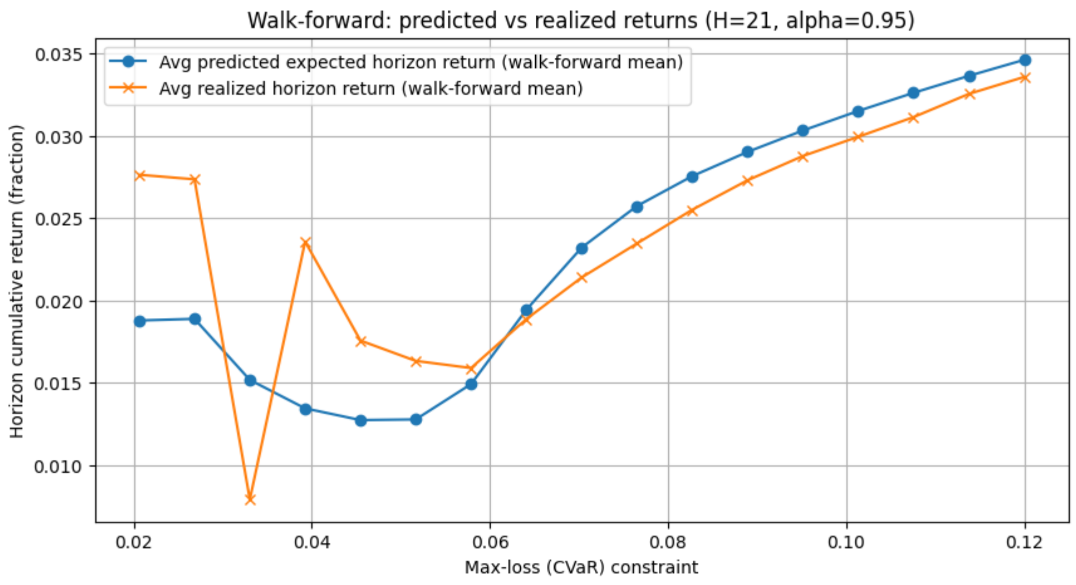

Figure 1: Predicted vs. realized H-day profit vs. CVaR loss limit (H=21, $\alpha = 0.95$). Each point averages only across decision dates where the requested CVaR bound was feasible.

References

- R. T. Rockafellar and S. Uryasev, "Optimization of Conditional Value-at-Risk," *Journal of Risk*, 2000 — an accessible introduction to CVaR and the optimization formulation.
- PyTorch documentation (for LSTM implementation): `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`
- CVXPy documentation (for convex optimization): `https://www.cvxpy.org`