ENSIIE & Université Paris-Saclay

Portfolio Optimization Review

Authors:
Lorenzo Abate
Luca Betti
Ettore Davide Brignoli

February 23, 2026

Contents

1 Abstract

This report provides an in-depth analysis of the paper Deep Learning for Portfolio Optimization by Z. Zhang, S. Zohren, and S. Roberts. The study proposes an end-to-end framework based on Long Short-Term Memory (LSTM) neural networks to optimize the Sharpe ratio of a portfolio comprising four different ETFs. The report is structured as follows: first, it introduces classical methodologies for portfolio optimization to provide a theoretical benchmark. Second, it evaluates the methodology presented in the paper by reproducing the original analysis and code, while also proposing additional performance metrics. Finally, the report discusses further developments in the field and describes current state-of-the-art models for portfolio management. In addition, the work was carried out collaboratively and consistently by all three members of the team, with each contributor providing an equal and substantive input across every section of the report and the code.

2 Classical Approaches

2.1 Mean–Variance Optimization (Markowitz)

Mean–variance optimization (MVO), introduced by Markowitz, is one of the foundational frameworks in modern portfolio theory. Its goal is to construct portfolios that trade off expected return and risk in an explicit, quantitative way, where risk is typically measured by the variance of portfolio returns. By varying the required level of expected return (or, equivalently, the investor's risk tolerance), MVO traces the *efficient frontier*: the set of portfolios that achieve the lowest possible variance for a given expected return (or the highest expected return for a given variance).

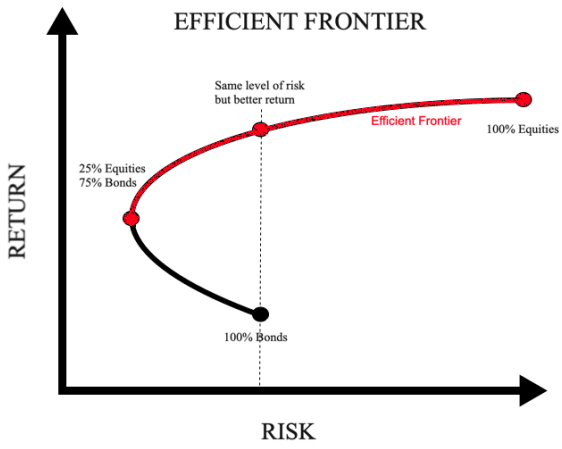


Figure 1: Markowitz efficient frontier

Let N be the number of assets, with portfolio weights $w \in \mathbb{R}^N$ such that $\mathbf{1}^\top w = 1$ (fully invested) and let $\mu \in \mathbb{R}^N$ be the vector of expected returns and $\Sigma \in \mathbb{R}^{N \times N}$ the covariance matrix of returns.

The Markowitz's mean–variance optimization (MVO) can be stated in one of the following ways:

$$\min_{w \in \mathbb{R}^N} \quad w^\top \Sigma w \tag{1}$$

$$\text{s.t.} \quad \mu^\top w \geq \mu_\star, \tag{2}$$

$$\mathbf{1}^\top w = 1, \tag{3}$$

$$w \in \mathcal{W}, \tag{4}$$

where $\mu_\star$ is a target expected return and $\mathcal{W}$ encodes constraints.

Even though this could be very elegant, when talking about quadratic optimization (including Markowitz's Critical Line Algorithm, CLA) it is important to notice that it can be unreliable in practice:

- **Instability:** small perturbations in inputs (especially expected returns) can lead to very

different optimal portfolios, making solutions highly sensitive to estimation error.

- **Concentration:** the optimizer may place extreme weights on a small subset of assets when it exploits noisy covariance/mean estimates.
- **Out-of-sample underperformance:** portfolios that are optimal in-sample may generalize poorly because the optimization amplifies estimation noise.

A key technical contributor is that quadratic programs typically rely on a well-conditioned, positive definite covariance matrix. So when Σ is ill-conditioned, its inversion and related numerical operations become error-prone, magnifying estimation noise. Because expected returns are notoriously difficult to forecast accurately, many approaches drop μ altogether and focus on risk-based allocations such as (hierarchical) risk parity.

2.2 Mean–Variance with a CVaR Constraint

Mean–Variance captures second-moment risk but is not tail-focused, so a common extension is to keep the Mean–Variance objective while constraining tail loss via Conditional Value-at-Risk (CVaR). We can consider the scenario of portfolio returns $\{r_t\}_{t=1}^{T}$, with $r_t \in \mathbb{R}^N$ and portfolio return $R_t(w) = w^\top r_t$ and define the portfolio *loss* as $L_t(w) = -R_t(w)$. It's known that for confidence level $\beta \in (0,1)$ the CVaR admits the convex representation:

$$\mathrm{CVaR}_\beta(w) \;=\; \min_{\eta\in\mathbb{R}} \left\{ \eta + \frac{1}{(1-\beta)T} \sum_{t=1}^{T} \big(L_t(w) - \eta\big)_+ \right\}, \tag{5}$$

where $(u)_+ = \max\{u, 0\}$ and η plays the role of a VaR-like threshold.

2.3 Covariance Estimation and Shrinkage

As said MVO provides theoretical guarantees and can be implemented efficiently, but it requires good inputs, especially a reliable covariance estimator. The sample covariance matrix S is a default choice, but it can have large estimation errors, particularly when the number of assets is comparable to or exceeds the number of observations. In MVO, these errors are dangerous because the optimizer can place large bets based on noisy covariance coefficients.

To avoid this problem, a remedy is *shrinkage*, which blends the sample covariance S with a structured target F (e.g., a factor model), using a shrinkage intensity $\delta \in [0,1]$:

$$\widehat{\Sigma} \;=\; \delta F + (1-\delta)S, \qquad 0 \le \delta \le 1. \tag{6}$$

The shrinkage reduces extreme sample estimates by pulling them toward the target, often improving out-of-sample stability of optimized portfolios.

2.4 Hierarchical Risk Parity (HRP)

Hierarchical Risk Parity (HRP) was introduced to address three major shortcomings of classical quadratic optimizers. In particular, it combines ideas from graph theory, via hierarchical clustering, and machine-learning-inspired recursive procedures to construct diversified portfolios using

information contained in the covariance or correlation matrix, without requiring the inversion of Σ. This feature allows HRP to avoid common failure modes associated with ill-conditioned or noisy covariance estimates.

2.4.1 Core steps

Given a universe of N assets with correlation matrix ρ, define the distance matrix D as

$$d_{ij} = \sqrt{\frac{1-\rho_{ij}}{2}}, \qquad i,j = 1,\dots,N. \tag{7}$$

The HRP algorithm then proceeds as follows:

1. **Hierarchical clustering:** a linkage method, such as single, complete, or average linkage, is applied to the distance matrix D in order to construct a dendrogram representing the hierarchical dependence structure among assets.
2. **Quasi-diagonalization:** assets are reordered according to the tree structure so that highly correlated assets are placed next to each other, resulting in a quasi-diagonal form of the covariance matrix.
3. **Recursive bisection and allocation:** the ordered set of assets is recursively split into two subclusters at each node of the tree, and capital is allocated across clusters based on their estimated variances.

2.4.2 Allocation rule for a two-cluster split

Consider a node that splits into two clusters, denoted by A and B, with associated covariance submatrixes Σ_A and Σ_B. Within each cluster, construct an inverse-variance portfolio:

$$w_A^{\mathrm{IVP}} \propto \mathrm{diag}(\Sigma_A)^{-1}, \qquad w_B^{\mathrm{IVP}} \propto \mathrm{diag}(\Sigma_B)^{-1}, \tag{8}$$

where the weights are normalized to sum to one within each cluster. The corresponding cluster variances are then given by

$$\sigma_A^2 = \left(w_A^{\mathrm{IVP}}\right)^\top \Sigma_A w_A^{\mathrm{IVP}}, \qquad \sigma_B^2 = \left(w_B^{\mathrm{IVP}}\right)^\top \Sigma_B w_B^{\mathrm{IVP}}. \tag{9}$$

Capital is allocated between the two clusters inversely proportional to their variances:

$$\alpha_A = \frac{\sigma_B^2}{\sigma_A^2+\sigma_B^2}, \qquad \alpha_B = \frac{\sigma_A^2}{\sigma_B^2+\sigma_A^2}. \tag{10}$$

The weights of assets in cluster A are multiplied by α_A, while those in cluster B are multiplied by α_B. Repeating this procedure recursively along the tree yields the final HRP portfolio weights.

2.4.3 Why HRP helps

Hierarchical Risk Parity exhibits several desirable properties:

- **Improved stability:** since no matrix inversion is required, the method is less sensitive to estimation error and ill-conditioned covariance matrices.
- **Reduced concentration:** diversification emerges from the hierarchical structure of asset dependencies rather than from the direct solution of a global quadratic optimization problem.
- **Enhanced out-of-sample robustness:** the recursive allocation mechanism mitigates the tendency of traditional optimizers to overfit noisy covariance inputs.

3 Critics

3.1 Deep Learning for Portfolio Optimization (Zhang, Zohren, Roberts)

3.1.1 Replica of the paper's methodology (end-to-end Sharpe optimisation)

Positioning and goal. Zhang, Zohren, and Roberts propose an end-to-end deep learning framework that directly outputs portfolio weights and *optimises the investment objective itself*, rather than a proxy forecasting loss. This is motivated by a key critique of the classical Modern Portfolio Theory pipeline: (i) forecast expected returns, (ii) estimate a covariance matrix and solve a constrained optimisation problem. In contrast, the paper bypasses the intermediate return-forecasting stage and updates model parameters to directly maximise the portfolio Sharpe ratio over a training horizon.

Asset universe and dataset. Instead of selecting individual stocks, the paper trades four highly liquid ETFs representing broad market indices: VTI (US total stock market), AGG (US aggregate bonds), DBC (commodities), and VIX (volatility). The dataset contains daily observations from 2006 to 2020. The evaluation focuses on the out-of-sample testing period from 2011 to the end of April 2020 and includes the 2020 COVID-19 shock. The model is retrained every two years using all data available up to the retraining date (expanding-window retraining).

1. Input construction (lookback window and stacking across assets)

Following the paper, we build inputs from **close prices** and **daily returns** for each ETF. Given a lookback window of $k = 50$ trading days, the input at time t is formed by concatenating prices and returns across the n assets, yielding a tensor with shape

$$x_t \in \mathbb{R}^{k\times(2n)}.$$

In the notebook implementation, this corresponds to (i) aligning calendars, (ii) forward-filling missing observations, (iii) computing returns $r_{i,t} = p_{i,t}/p_{i,t-1} - 1$, and (iv) creating supervised samples with a rolling window.

2. Differentiable EWMA volatility (ex-ante risk estimate)

A key technical component is the ex-ante volatility estimate $\sigma_{i,t-1}$ used for volatility targeting. The paper uses an exponentially weighted moving standard deviation with a 50-day window. In our replica, we implement it *differentiably* (TensorFlow graph) via a recursion (e.g. `tf.scan`) over

the squared returns in the lookback window, so that volatility scaling remains compatible with gradient-based learning.

3. Modified portfolio return with volatility targeting and turnover costs (Eq. (7))

The learning objective is based on a *modified* realised portfolio return that includes: (i) volatility scaling to enforce a target risk level σ_{tgt} and (ii) transaction costs proportional to turnover. For n assets and long-only weights $w_{i,t-1}$, the paper defines the modified return as:

$$R_{p,t} = \sum_{i=1}^{n} \frac{\sigma_{\text{tgt}}}{\sigma_{i,t-1}} w_{i,t-1}\, r_{i,t} \;-\; C \sum_{i=1}^{n} \left| \frac{\sigma_{\text{tgt}}}{\sigma_{i,t-1}} w_{i,t-1} - \frac{\sigma_{\text{tgt}}}{\sigma_{i,t-2}} w_{i,t-2} \right|, \tag{11}$$

where C is the cost rate. This formulation is central: it couples portfolio decisions across time (via turnover) and enforces a stable risk profile (via volatility targeting), which materially changes what the optimiser learns compared to training on gross returns.

4. LSTM portfolio network (architecture + long-only constraint)

The portfolio weights are produced directly by a neural network:

$$w_{t-1} = f_\theta(x_t), \qquad w_{i,t-1} \in [0,1], \quad \sum_{i=1}^{n} w_{i,t-1} = 1.$$

To satisfy the long-only simplex constraint, the paper uses a **softmax** output layer:

$$w_{i,t} = \frac{\exp(\tilde{w}_{i,t})}{\sum_{j=1}^{n} \exp(\tilde{w}_{j,t})}.$$

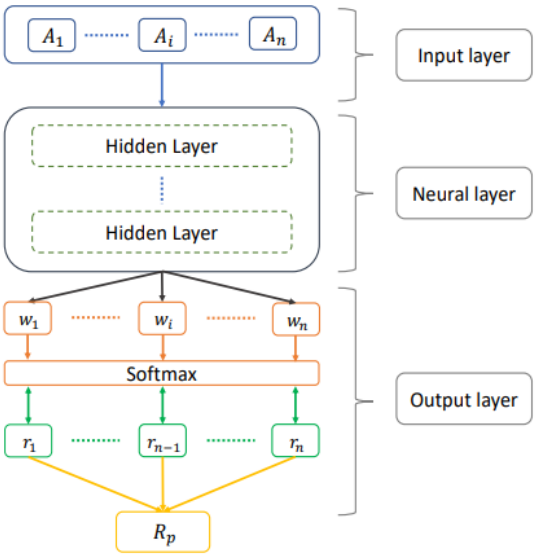


Figure 2: Model architecture: stacked multi-asset input → neural feature extraction (LSTM) → softmax portfolio weights.

5. Objective function: direct Sharpe maximisation (Eq. (1)–(6))

The paper optimises the Sharpe ratio over a training horizon T:

$$L_T = \frac{\mathbb{E}(R_{p,t})}{\mathrm{Std}(R_{p,t})}, \qquad R_{p,t} = \sum_{i=1}^{n} w_{i,t-1}\, r_{i,t},$$

and performs gradient ascent on L_T with learning rate α:

$$\theta_{\text{new}} := \theta_{\text{old}} + \alpha \frac{\partial L_T}{\partial \theta}.$$

In practice, our implementation follows the same principle by minimising $-L_T$ with Adam.

6. Training scheme and validation split (paper: last 10% validation)

The paper keeps the network intentionally simple (one LSTM layer with 64 units), trains with Adam, mini-batches (size 64, no shuffling), and typically stops after 100 epochs. Crucially, it uses **the last 10% of the available training data as a temporal validation set** to tune hyperparameters and control overfitting, while keeping the test set for final evaluation.

Our replica (core implementation)

```
class LSTMPortfolioEq7(tf.keras.Model):
    def __init__(self, n_assets, lstm_units=64, sigma_tgt=0.10, C=1e-4,
                 use_scaling=True, train_on_eq7=True):
        super().__init__()
        self.n_assets = int(n_assets)
        self.sigma_tgt = float(sigma_tgt)
        self.C = float(C)
        self.use_scaling = bool(use_scaling)
        self.train_on_eq7 = bool(train_on_eq7)

        self.lstm = tf.keras.layers.LSTM(lstm_units)
        self.out  = tf.keras.layers.Dense(self.n_assets, activation=
   softmax )

    def call(self, x, training=False):
        h = self.lstm(x, training=training)
        return self.out(h)

    def train_step(self, data):
        x, r_t = data
        with tf.GradientTape() as tape:
            w_tm1 = self(x, training=True)

            if self.train_on_eq7:
                Rp, turnover, cost = modified_return_eq7_sequence(
                    x, r_t, w_tm1,
                    n_assets=self.n_assets,
                    sigma_tgt=self.sigma_tgt,
                    C=self.C,
```

```
                    use_scaling=self.use_scaling
                )
            else:
                # GROSS return: no scaling, no costs
                Rp = tf.reduce_sum(w_tm1 * r_t, axis=1)  # (batch,)
                turnover = tf.zeros_like(Rp)
                cost = tf.zeros_like(Rp)

            J = sharpe_ratio(Rp)
            loss = -J

        grads = tape.gradient(loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(grads, self.trainable_variables
    ))

        return {
             loss : loss,
             sharpe_obj : J,
             avg_turnover : tf.reduce_mean(turnover),
             avg_cost : tf.reduce_mean(cost),
        }

    def test_step(self, data):
        x, r_t = data
        w_tm1 = self(x, training=False)

        Rp_mod, turnover, cost = modified_return_eq7_sequence(
            x, r_t, w_tm1,
            n_assets=self.n_assets,
            sigma_tgt=self.sigma_tgt,
            C=self.C,
            use_scaling=self.use_scaling
        )

        J = sharpe_ratio(Rp_mod)
        loss = -J
        return {
             loss : loss,
             sharpe_obj : J,
             avg_turnover : tf.reduce_mean(turnover),
             avg_cost : tf.reduce_mean(cost),
        }

def fit_on_period_with_val(
    X_train_full, Y_train_full, n_assets,
    lstm_units=64, sigma_tgt=0.10, C=1e-4,
    lr=1e-3, epochs=100, val_frac=0.10,
    use_early_stopping=False,
    use_scaling=True,
    train_on_eq7=True,
    batch_size=64
):
    N = len(X_train_full)
    n_val = max(1, int(np.floor(val_frac * N)))
    if N - n_val < 50:
```

```
        raise ValueError(f Training too short after split val: N={N}, n_val={n_val} )

    # temporal split: train = initial part, val = last 10%
    X_tr, Y_tr = X_train_full[:-n_val], Y_train_full[:-n_val]
    X_va, Y_va = X_train_full[-n_val:], Y_train_full[-n_val:]

    model = LSTMPortfolioEq7(
        n_assets=n_assets,
        lstm_units=lstm_units,
        sigma_tgt=sigma_tgt,
        C=C,
        use_scaling=use_scaling,
        train_on_eq7=train_on_eq7
    )
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr), run_eagerly=False)

    # mini-batch time-ordered (NO shuffle)
    ds_tr = tf.data.Dataset.from_tensor_slices((X_tr, Y_tr)).batch(batch_size, drop_remainder=False)
    ds_va = tf.data.Dataset.from_tensor_slices((X_va, Y_va)).batch(batch_size, drop_remainder=False)

    callbacks = []
    if use_early_stopping:
        callbacks.append(tf.keras.callbacks.EarlyStopping(
            monitor= val_loss , mode= min , patience=10, restore_best_weights=True
        ))

    hist = model.fit(ds_tr, validation_data=ds_va, epochs=epochs, verbose=0, callbacks=callbacks)
    return model, hist
```

Listing 1: LSTM Portfolio (paper-consistent implementation)

7. Backtesting protocol and metrics (paper-consistent reporting)

Backtests follow the paper's walk-forward scheme: retrain biennially on an expanding window and evaluate on the subsequent test interval. Performance is summarised using the same family of metrics as in the paper: annualised expected return, annualised volatility, Sharpe and Sortino ratios, downside deviation, maximum drawdown, percentage of positive-return days, average gain/loss ratio, and (importantly) turnover and transaction cost impact when Eq. (11) is used.

8. Evaluation: replication results and robustness notes

Evaluation: replication results under alternative training objectives

Tables 1–2 summarize our replicated backtests over 2011–2020 (including the 2020 high-volatility regime), distinguishing between two training setups.

Training on simple (gross) returns. We first train the LSTM by maximising the Sharpe ratio computed on *gross* portfolio returns, i.e. without volatility targeting and without transaction costs (no Eq. (7) terms). This setting provides a useful baseline, but it is *not* fully aligned with the paper's trading objective. As shown in Table 1, the resulting strategy can achieve strong risk-adjusted performance, but it does so without explicitly internalising turnover penalties or risk targeting during learning. In practice, this often leads to less controlled exposure dynamics and makes the comparison to the paper's Deep Learning Strategy (DLS) less direct.

Training on modified returns with volatility targeting and transaction costs (Eq. (7)). We then replicate the paper's core training objective by maximising Sharpe on the *modified* return $R_{p,t}$ that includes volatility scaling to a target risk level and turnover costs. Results are reported in Table 2. In this configuration, performance more closely matches the DLS behaviour described in the paper: the Sharpe ratio remains competitive under realistic transaction cost levels, while exposure is stabilised by volatility targeting. Increasing costs from $C = 1$bp to $C = 10$bp reduces turnover and lowers performance, as expected, illustrating the trade-off between trading aggressiveness and implementability.

Robustness and training noise. Across both setups, training remains inherently noisy and exhibits non-negligible dependence on random initialisation (seed). Individual runs may therefore display variability in out-of-sample metrics even when architecture and hyperparameters are fixed. This is consistent with the non-convex optimisation landscape of deep models and motivates repeated runs and robust validation protocols when drawing conclusions.

Table 1: Backtest summary (2011–2020) — training on *simple (gross)* returns (no volatility targeting, no transaction costs).

Scenario	Use scaling	σ_{tgt} (daily)	C	$E(R)$ (ann.)	Std(R) (ann.)	Sharpe (ann.)	DD(R) (ann.)	Sortino (ann.)	MDD	% + Ret	Ave. P / Ave. L	Avg Turnover	Avg Cost	N	Start	End
No scaling, C=0	False	0.006299	0.0000	0.271105	0.191401	1.416423	0.106622	2.542664	-0.106671	0.541046	1.267893	0.061468	0.000000	2351	2011-01-03	2020-04-30
Scaling 10% ann, C=1bp	True	0.006299	0.0001	0.084522	0.067197	1.257831	0.051745	1.633428	-0.099651	0.548703	1.027695	0.101364	0.000010	2351	2011-01-03	2020-04-30
Scaling 10% ann, C=10bp	True	0.006299	0.0010	0.052629	0.067839	0.775787	0.053931	0.975855	-0.146575	0.544875	0.957161	0.095960	0.000096	2351	2011-01-03	2020-04-30

Table 2: Backtest summary (2011–2020) — training on *modified* returns with volatility targeting and transaction costs (Eq. (7)).

Scenario	Use scaling	σ_{tgt} (daily)	C	$E(R)$ (ann.)	Std(R) (ann.)	Sharpe (ann.)	DD(R) (ann.)	Sortino (ann.)	MDD	% + Ret	Ave. P / Ave. L	Avg Turnover	Avg Cost	N	Start	End
No scaling, C=0	False	0.006299	0.0000	0.233429	0.133630	1.746829	0.072065	3.239141	-0.096617	0.553382	1.237770	0.044910	0.000000	2351	2011-01-03	2020-04-30
Scaling 10% ann, C=1bp	True	0.006299	0.0001	0.105332	0.060117	1.752095	0.036633	2.875279	-0.090930	0.544875	1.227120	0.031419	0.000003	2351	2011-01-03	2020-04-30
Scaling 10% ann, C=10bp	True	0.006299	0.0010	0.084130	0.061178	1.375176	0.035960	2.339521	-0.079772	0.529137	1.188374	0.025769	0.000026	2351	2011-01-03	2020-04-30

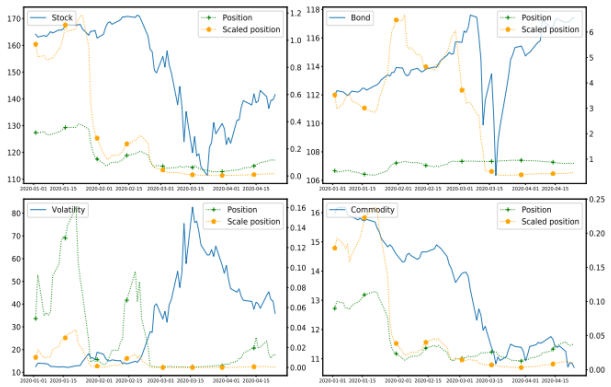


Figure 3: Portfolio weights (replica) over the test period.

3.1.2 Verification of paper assumptions via last-10% temporal CV

Why we need a dedicated verification stage. After reproducing the core pipeline, we use the paper's own validation principle—**a temporal split with the last 10% held out**—as a controlled way to test which design choices are robust and which are artefacts of noise. This stage is intentionally separated from the final test set (2011–2020): all comparisons are performed inside each training window, using the last-10% block as a pseudo out-of-sample validation set.

Robustness checks under last-10% temporal validation

Using a **last-10% temporal validation split**, we stress-test the main modelling choices of the paper while keeping the final test set untouched. To make repeated comparisons feasible within each training window, we adopt practical constraints (e.g. fewer epochs and, when appropriate, full-batch optimisation), so that multiple configurations can be evaluated under the *same* time-ordered validation protocol.

- **Objective alignment:** optimising a Sharpe-ratio objective built on the *modified* portfolio returns (volatility targeting + turnover costs) provides a more informative and generally more stable training signal than optimising gross returns, and is better aligned with the portfolio construction goal.
- **Input features:** additional feature engineering beyond raw prices and returns often increases variance in validation performance and can amplify training noise; in our experiments, the plain price/return representation is the most reliable input for the LSTM.
- **Model choice:** under strict temporal validation, LSTMs remain the most robust among simple daily-data architectures, whereas more flexible parameterisations may achieve higher in-sample objectives but exhibit sharper overfitting and less consistent validation behaviour.

Final evaluation (selected configuration)

After selecting the most stable configuration according to last-10% temporal validation, we report the corresponding out-of-sample backtest on the full 2011–2020 test window. The key takeaway is that configurations that look strong under the last-10% temporal CV are also the ones that are most likely to remain competitive out-of-sample, while overly engineered features or unstable objectives tend to fail to generalise.

3.1.3 Cross-Validation for Portfolio Construction: Why Standard CV Fails

The purpose of cross-validation (CV) is to estimate the generalization error of a learning algorithm, thereby preventing overfitting. In standard machine learning, observations are assumed to be drawn from an IID process, and CV creates train/test splits so that each observation belongs to one and only one set, avoiding information leakage between training and testing. A popular scheme is k-fold CV:

1. Partition the dataset into k disjoint folds.
2. For $i = 1, \ldots, k$: train on all folds except i, then test on fold i.

When applied naïvely to financial datasets, however, CV may fail to detect overfitting and can even contribute to it through repeated hyper-parameter tuning.

Why naive CV can be misleading in finance. A central reason is that financial observations are rarely IID, features are typically serially correlated and labels often depend on overlapping time intervals. As a consequence, random (or standard k-fold) splits can place nearly identical information in both the train and test sets, creating *leakage*. This leakage inflates backtest performance, especially in the presence of irrelevant features, leading to false discoveries.

Formally, consider features $\{X_t\}$ and labels $\{Y_t\}$ built from time series data, in many financial settings:

$$X_t \approx X_{t+1} \quad \text{(serial correlation)}, \qquad Y_t \approx Y_{t+1} \quad \text{(overlapping label construction)}. \tag{12}$$

If observation t is assigned to training and t+1 to testing, a model trained on (X_t, Y_t) will be evaluated on (X_{t+1}, Y_{t+1}), which contains overlapping information. The model can appear to predict well even if X is not truly predictive. In other words, the test set is no longer "unseen" in an informational sense.

Importantly, overlap in features alone is not sufficient for leakage, leakage requires that both feature-label pairs overlap informationally, i.e., $(X_i, Y_i) \approx (X_j, Y_j)$, not merely $X_i \approx X_j$ or $Y_i \approx Y_j$.

Multiple testing and selection bias. Another failure mode is that the testing set is used repeatedly during model development (feature selection, hyper-parameter tuning, model comparison). Even if each split is constructed correctly, reusing the same data for many decisions introduces selection bias. This can turn CV into an overfitting engine: performance improves because we search over more configurations, not because the strategy truly generalizes.

3.1.4 Leakage Mechanism with Overlapping Labels

Let label Y_j be decided using an information set Φ_j (the set of random draws or time points that the label depends on). For example, a label can be a function of returns over a time interval:

$$Y_j = f([t_{j,0}, t_{j,1}]) \,, \tag{13}$$

meaning that Y_j depends on observations in the closed time range $[t_{j,0}, t_{j,1}]$. Two labels $Y_i = f([t_{i,0}, t_{i,1}])$ and $Y_j = f([t_{j,0}, t_{j,1}])$ are *concurrent* (overlap in time) if any of the following sufficient conditions holds:

$$t_{j,0} \le t_{i,0} \le t_{j,1}, \tag{14}$$

$$t_{j,0} \le t_{i,1} \le t_{j,1}, \tag{15}$$

$$t_{i,0} \le t_{j,0} \le t_{j,1} \le t_{i,1}. \tag{16}$$

If such overlaps occur across the train/test boundary, the model effectively sees in training information that will reappear in testing.

A practical symptom is that measured performance increases monotonically as $k \to T$ (with T the number of time bars/observations), because more splits create more train/test neighbors and hence more overlaps. This is a red flag: the backtest is likely profiting from leaks.

3.1.5 A Solution: Purged k-Fold CV and Embargo

A finance-aware approach modifies k-fold CV to reduce leakage caused by overlapping labels and serial correlation.

Purging. For each test fold, *purge* from the training set all observations whose labels overlap in time with any label in the test set. Using the information-set notation: if Y_j in the test set depends on Φ_j, then remove from training any observation i whose label depends on Φ_i such that

$$\Phi_i \cap \Phi_j \neq \emptyset. \tag{17}$$

Operationally, this corresponds to removing training observations whose label intervals $[t_{i,0}, t_{i,1}]$ overlap with the test label intervals $[t_{j,0}, t_{j,1}]$ according to the concurrency conditions above.

Purging often suffices: if performance improves with k up to some value $k^\star$ and then plateaus, it suggests recalibration benefits without systematic leakage.

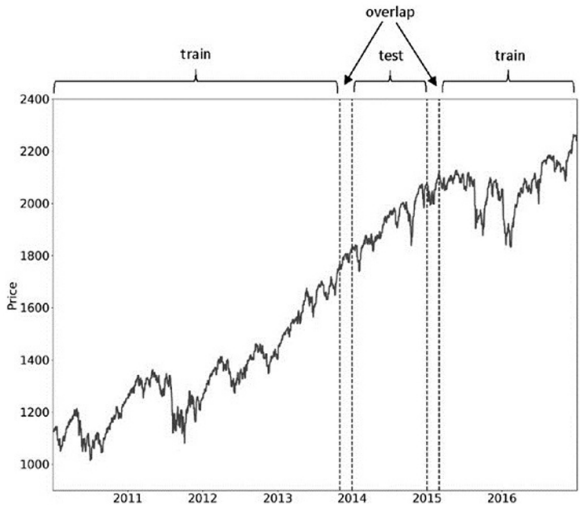


Figure 4: Purged k-fold cross-validation

Embargo. When purging alone is insufficient (e.g., due to strong serial dependence in features), impose an *embargo* period immediately after each test fold: remove from training observations that occur shortly after the test window, because their features can still be correlated with the test set. Let a test label span $[t_{j,0}, t_{j,1}]$. Embargo removes training labels with start times $t_{i,0}$ satisfying

$$t_{j,1} \leq t_{i,0} \leq t_{j,1} + h, \tag{18}$$

where h is the embargo length (often a small fraction of the sample, e.g., $h \approx 0.01T$). A convenient implementation is to extend the test label horizon to $t_{j,1} + h$ before purging, so that those post-test observations are automatically excluded.

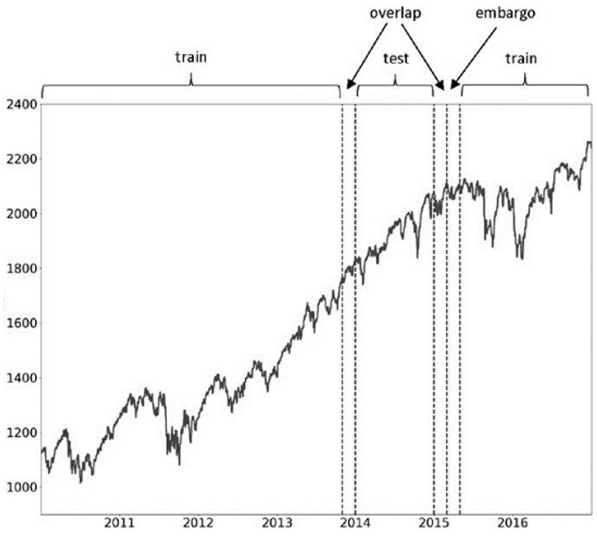


Figure 5: Embargo of post-test train observations

3.1.6 Implications for Portfolio Construction

When ML is used for portfolio construction (e.g., forecasting returns, estimating risk, selecting assets, or tuning regularization), naive CV can cause the entire pipeline to overfit and then validate itself due to leakage. Purged k-fold CV with embargo aims to restore the meaning of "out-of-sample" by ensuring that:

- no training label reuses information that determines testing labels (purging);
- training samples immediately following the test fold are removed to mitigate serial dependence (embargo).

These modifications do not guarantee success, but they substantially reduce the most common source of inflated performance in financial ML: leakage induced by temporal dependence and overlapping labels.

3.2 Beyond Volatility: Sharpe Ratio as a Risk Lens for Deep Portfolio Optimization

In the referenced paper, the authors optimize the Sharpe Ratio as their risk-adjusted performance objective, consistent with standard practice in the asset-management industry. Let r_t denote the portfolio return at time t, $r_{f,t}$ the risk-free return, and $x_t = r_t - r_{f,t}$ the excess return. The (population) Sharpe Ratio is

$$SR = \frac{\mu_x}{\sigma_x}, \tag{19}$$

where $\mu_x = \mathbb{E}[x_t]$ and $\sigma_x = \sqrt{\mathbb{V}[x_t]}$. Given a sample of n observations, the usual estimator is

$$\widehat{SR} = \frac{\overline{x}}{s_x}, \tag{20}$$

with $\overline{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$ and $s_x = \sqrt{\frac{1}{n-1}\sum_{t=1}^{n}(x_t - \overline{x})^2}$.

However, the choice of risk metric should ultimately be aligned with the portfolio's primary mandate and a well-known limitation of the Sharpe Ratio is that its denominator is the total return volatility (standard deviation), which treats positive and negative deviations from the mean symmetrically. As a consequence, a strategy that experiences large upside fluctuations (e.g., strong gains occurring irregularly) may be penalized through a higher estimated volatility, potentially lowering the Sharpe Ratio even when such variability is desirable from an investor's perspective.

A common alternative within the "downside-risk" framework is the Sortino Ratio, originally proposed to address the fact that investors are typically more concerned with outcomes falling below a target than with upside variability. Instead of total volatility, the Sortino Ratio scales excess performance by the downside deviation, i.e., the semi-deviation of returns that fall below the chosen target. In doing so, it penalizes only "bad" volatility (shortfalls) while not penalizing "good" volatility (outperformance above the target).

For long-horizon, capital-preservation investors such as pension funds, drawdown-based metrics can be more informative than moment-based measures. In particular, the Calmar Ratio links performance to the maximum drawdown, thus directly discouraging deep peak-to-trough losses. Complementarily, the Ulcer Index focuses on both the depth and the persistence of drawdowns by aggregating squared percentage drawdowns over time, making it especially relevant when the stability of the equity curve is a key objective.

Let V_t denote the portfolio value (equity curve) and $M_t = \max_{1\le s\le t} V_s$ the running peak. Define the drawdown at time t as

$$DD_t \;=\; \frac{V_t - M_t}{M_t} \le 0. \tag{21}$$

The maximum drawdown over the sample is

$$MDD \;=\; \min_{1\le t\le n} DD_t, \qquad \text{so } |MDD| \text{ is the peak-to-trough loss magnitude.} \tag{22}$$

Let T_{yrs} be the length of the sample in years; the compound annual growth rate is

$$CAGR \;=\; \left(\frac{V_n}{V_0}\right)^{\frac{1}{T_{\text{yrs}}}} - 1. \tag{23}$$

The Calmar Ratio is commonly defined as

$$Calmar \;=\; \frac{CAGR}{|MDD|}. \tag{24}$$

Since the authors of the paper under discussion adopt the Sharpe Ratio, we focus the remainder of this section on strengthening inference around that metric.

3.2.1 Sampling uncertainty and the Probabilistic Sharpe Ratio (PSR)

A crucial limitation of the classical Sharpe Ratio is that the population parameters μ_x and σ_x are unknown and must be estimated from finite samples. Consequently, the commonly reported Sharpe Ratio $\widehat{SR}$ is itself an estimator and should be interpreted together with its sampling uncertainty. Under mild regularity conditions, the estimator admits an asymptotic Normal approximation. In particular, allowing for non-Normal returns through skewness γ_3 and kurtosis γ_4, a widely used

standard-error approximation is

$$\widehat{\sigma}\Big(\widehat{SR}\Big) = \sqrt{\frac{1}{n-1}\left(1 + \frac{1}{2}\widehat{SR}^2 - \gamma_3\widehat{SR} + \frac{\gamma_4 - 3}{4}\widehat{SR}^2\right)}. \tag{25}$$

Equivalently, one may write

$$(\widehat{SR} - SR) \xrightarrow{a} \mathcal{N}\left(0,\ \frac{1 + \frac{1}{2}SR^2 - \gamma_3 SR + \frac{\gamma_4 - 3}{4}SR^2}{n-1}\right), \tag{26}$$

which highlights how higher moments affect confidence bands even when the point estimate $\widehat{SR}$ is unchanged.

Building on this observation, Bailey and López de Prado introduce the *Probabilistic Sharpe Ratio* (PSR), which expresses performance as the probability that the *true* Sharpe Ratio exceeds a user-defined benchmark SR^*:

$$PSR(SR^*) = \mathbb{P}(SR > SR^*)\,. \tag{27}$$

Using the Normal approximation of $\widehat{SR}$, this probability can be estimated as

$$\widehat{PSR}(SR^*) = \Phi\left(\frac{\widehat{SR} - SR^*}{\widehat{\sigma}(\widehat{SR})}\right), \tag{28}$$

where $\Phi(\cdot)$ denotes the CDF of the standard Normal distribution. Substituting eq. (25) into eq. (28) yields the explicit expression

$$\widehat{PSR}(SR^*) = \Phi\left[\frac{(\widehat{SR} - SR^*)\sqrt{n-1}}{\sqrt{1 + \frac{1}{2}\widehat{SR}^2 - \gamma_3\widehat{SR} + \frac{\gamma_4 - 3}{4}\widehat{SR}^2}}\right]. \tag{29}$$

For a fixed benchmark SR^*, eq. (29) implies that $\widehat{PSR}$ increases with a larger $\widehat{SR}$ and a longer track record n, increases with more positive skewness γ_3, and decreases with fatter tails (higher γ_4).

3.2.2 A real World Application

Suppose we observe two return-generating processes whose *true* distributional moments are as reported in Table 3. It is important to emphasize that, in practice, these population parameters are not directly observable and can only be inferred from finite samples. Therefore, unless one has an exceptionally long track record, the corresponding empirical estimates may be affected by substantial sampling error.

Table 3: True moments of the two underlying return distributions.

	Strategy 1	Strategy 2
True mean	0.20%	0.32%
True stDev	0.60%	0.60%
True skewness	-0.99	0.00
True kurtosis	3.86	3.00

Figure 6: True long term return distributions

The harsh reality is that, when deciding which portfolio should receive an allocation of our savings, we often face severe data limitations. Assume that only a one-year track record of *weekly* returns is available; hence, we observe $n = 52$ weekly returns for each portfolio. Table 4 reports the corresponding summary statistics.

Table 4: Summary statistics computed from one year ($n = 52$) of weekly returns.

	Hedge Fund 1	Hedge Fund 2
Mean	0.145%	0.136%
StDev	0.639%	0.633%
Skewness	-0.78	0.95
Kurtosis	2.87	5.92
SR	0.226	0.215
SR Annualized	1.632	1.551

At first glance, Hedge Fund 1 appears preferable due to its higher Sharpe Ratio: "Hedge Fund 1 has the larger Sharpe ratio—let us invest in it." However, this conclusion ignores the sampling uncertainty inherent in $\widehat{SR}$ when the available track record is short. A more informative assessment is provided by the *Probabilistic Sharpe Ratio* (PSR), which quantifies how confident we can be that the *true* Sharpe Ratio exceeds a benchmark level SR^*.

Table 5 reports the estimated Sharpe Ratio, its estimated standard error, and the Probabilistic Sharpe Ratio computed with benchmark $SR^* = 0$.

Table 5: Sharpe Ratio uncertainty and Probabilistic Sharpe Ratio with benchmark $SR^* = 0$.

	Hedge Fund 1	Hedge Fund 2
$\widehat{SR}$	0.2263	0.2151
$\widehat{\sigma}(\widehat{SR})$	0.1534	0.1293
$\widehat{PSR}(SR^* = 0)$	92.99%	95.19%

These results suggest that, despite the slightly larger $\widehat{SR}$ of Hedge Fund 1, an allocation to Hedge Fund 2 may be more reasonable. The rationale is that we cannot attach the same level of confidence to the two Sharpe estimates: for Hedge Fund 1, the probability that the future true Sharpe ratio

exceeds 0 is "only" 92.99%, whereas for Hedge Fund 2 this probability is higher, at 95.19%. In other words, Hedge Fund 2 exhibits stronger statistical evidence of a positive risk-adjusted performance under the benchmark $SR^* = 0$.

3.3 Anomaly Detection: Identifying Regime Shifts

To improve the portfolio performance reported in the paper, we propose adding an anomaly-detection stage. This step aims to identify adverse episodes, such as severe drawdowns, and would therefore allow the asset manager to reduce risk exposure during such periods. We frame anomaly detection as a binary classification problem and introduce a target variable Y that serves as a market proxy. In our idea, we suggest the Vanguard LifeStrategy 60% Equity (a 60/40 Vanguard portfolio), which we consider a reasonable representation of the broader market.

We then suggest to apply standard feature scaling and split the sample into training, validation, and test sets while preserving the chronological order of the time series, in order to avoid look-ahead bias. For the classification task, we propose an MLP classifier and control model complexity through a grid search over the number of hidden layers and the number of neurons per layer and the model selection is performed on the validation set using the ROC–AUC criterion. In the portfolio-construction step, whenever the classifier signals an anomaly, the strategy scales down exposure to risky assets/factors to mitigate drawdowns and overall volatility.

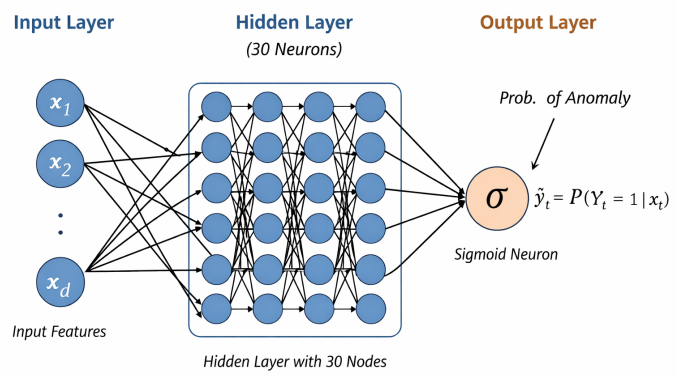


Figure 7: MLP classifier architecture used for anomaly detection

Cycle-based alternative to deep learning. In addition to a deep-learning approach, we note that a more classical framework based on business-cycle analysis could be equally effective. A key limitation of black-box models is that they often provide limited insight into the economic drivers that trigger an anomaly signal. By contrast, an intuitive regime-classification approach grounded in the macroeconomic cycle can substantially improve the *explainability* of portfolio decisions, which is particularly valuable when communicating the strategy to external stakeholders (e.g., a pool of potential investors).

We consider a four-phase representation of the economic cycle, where regimes are determined by the level of U.S. inflation and growth using CPI and GDP as proxy. Let π_t denote inflation (CPI) and g_t denote real growth (GDP) at time t. Given two thresholds $\bar{\pi}$ and $\bar{g}$, we define the regime

mapping

$$\mathcal{R}_t = \begin{cases} \text{Recovery}, & g_t \geq \bar{g} \text{ and } \pi_t < \bar{\pi}, \\ \text{Overheat}, & g_t < \bar{g} \text{ and } \pi_t \geq \bar{\pi}, \\ \text{Reflective}, & g_t < \bar{g} \text{ and } \pi_t < \bar{\pi}, \\ \text{Stagflation}, & g_t < \bar{g} \text{ and } \pi_t \geq \bar{\pi}, \end{cases} \tag{30}$$

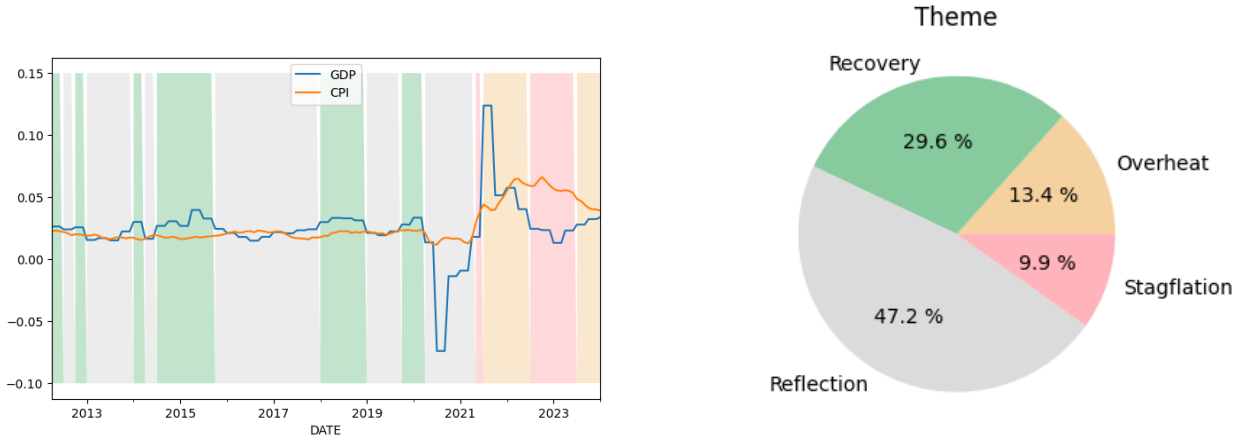
where the four quadrants correspond to distinct combinations of growth and inflation.[1] In this stylized allocation logic, each regime is associated with a preferred asset class:

$$\mathcal{R}_t = \text{Reflective} \;\Rightarrow\; \text{bonds are typically favored}, \tag{31}$$
$$\mathcal{R}_t = \text{Recovery} \;\Rightarrow\; \text{equities are typically favored}, \tag{32}$$
$$\mathcal{R}_t = \text{Overheat} \;\Rightarrow\; \text{commodities are typically favored}, \tag{33}$$
$$\mathcal{R}_t = \text{Stagflation} \;\Rightarrow\; \text{cash (or defensive positioning) is typically favored}. \tag{34}$$

Combining regime information with the anomaly model. This four-quadrant framework can be used to inform the deep-learning component by providing a regime-conditioned allocation bias. Concretely, let $\mathbf{w}_t^{\mathrm{DL}}$ be the baseline portfolio weights produced by the deep-learning pipeline, and let $\mathbf{b}(\mathcal{R}_t)$ be a regime-specific tilt vector that increases the weight of the asset class favored in regime $\mathcal{R}_t$. A simple combination is

$$\mathbf{w}_t \propto \mathbf{w}_t^{\mathrm{DL}} \odot \mathbf{b}(\mathcal{R}_t), \qquad \mathbf{w}_t \leftarrow \frac{\mathbf{w}_t}{\mathbf{1}^\top \mathbf{w}_t}, \tag{35}$$

where $\odot$ denotes element-wise multiplication and the normalization ensures that weights sum to one. In this way, we combine the computational power and pattern-recognition capabilities of deep learning with an economically interpretable regime structure, thereby improving decision transparency and potentially enhancing overall portfolio performance.



(a) U.S. CPI and GDP used to define regime thresholds.

(b) Distribution of observations across the four economic regimes.

Figure 8: Macroeconomic inputs and regime composition for the four-phase business-cycle classification.

[1] In our terminology, *Reflective* denotes weak growth and low inflation.

4 From the Source: Talking with the Author on VIX Term Structure Forecasting

After contacting the paper's author, we engaged in an insightful discussion on how to improve the model's performance by leveraging well-known properties of volatility and the VIX, such as volatility clustering and volatility autocorrelation. In line with the author's suggestions, we therefore propose using an econometric model to forecast the VIX term structure. Given the nature of the VIX, this forecast can provide a strong signal of potential market turbulence, which the strategy could seek to avoid in order to improve its risk-adjusted returns.

Following a targeted review of the literature and additional guidance from the author, we opted for a parsimonious two-step approach: (i) fit a Nelson–Siegel term-structure model to the cross-section of VIX futures across maturities, and (ii) forecast the latent Nelson–Siegel factors through a locally homogeneous autoregressive model, so as to explicitly account for volatility persistence while remaining robust to structural changes.

4.1 Step 1: Fitting the VIX Term Structure with a Nelson–Siegel Specification

Let $y_t(\tau)$ denote the VIX-implied volatility (or a term-structure proxy) observed at date t for maturity τ. In the Nelson–Siegel framework, the term structure is modeled as

$$y_t(\tau) = \beta_{1,t} + \beta_{2,t}\left(\frac{1-e^{-\lambda\tau}}{\lambda\tau}\right) + \beta_{3,t}\left(\frac{1-e^{-\lambda\tau}}{\lambda\tau} - e^{-\lambda\tau}\right) + \epsilon_t(\tau), \qquad \epsilon_t(\tau) \sim \mathcal{N}(0, \sigma_\epsilon^2), \tag{36}$$

where $\lambda > 0$ controls the decay rate.

For each day t, the parameters $\{\beta_{1,t}, \beta_{2,t}, \beta_{3,t}\}$ can be estimated by (non-)linear least squares across the available maturities $\tau \in \{\tau_1, \ldots, \tau_M\}$, typically either fixing λ globally or re-estimating it under additional regularization. Denoting by $\widehat{\beta}_t$ the fitted factor vector at time t,

$$\widehat{\beta}_t = \begin{pmatrix} \widehat{\beta}_{1,t} \\ \widehat{\beta}_{2,t} \\ \widehat{\beta}_{3,t} \end{pmatrix}, \tag{37}$$

we obtain a low-dimensional representation of the entire term structure that is convenient for time-series forecasting.

4.2 Step 2: Forecasting the Nelson–Siegel Factors via a Local AR(1) Model

To incorporate volatility clustering and the time-varying persistence of volatility regimes, we suggest to forecast the factors using a *locally homogeneous* autoregressive model of order one (LAR(1)). For a given factor process β_t (e.g., any component of $\widehat{\beta}_t$), the LAR(1) at time t is defined as

$$\beta_t = \theta_{0,t} + \theta_{1,t}\beta_{t-1} + \mu_t, \qquad \mu_t \sim \mathcal{N}(0, \sigma_t^2), \tag{38}$$

4.3 From Forecasts to a Trading Signal and Dynamic Asset Allocation

The key output of our approach is a forecast of implied volatility at a specific horizon (for instance, the standard 30-day VIX horizon). To turn this forecast into an actionable signal, we compare

what our model expects for that future horizon with what the market is currently implying at the same maturity through today's term structure.

The intuition is straightforward: if the model anticipates higher volatility than what is priced by the market, we interpret this gap as a warning that risk conditions may deteriorate. In that case, the strategy becomes more defensive by reducing exposure to risky assets and increasing allocations to safer assets. Conversely, if the model forecasts lower volatility than what is currently implied, this suggests that markets may be overpricing turbulence, and the strategy can adopt a more risk-on stance.

5 Further Approaches

Now that we discussed possible flaws and improvements of the original paper, we focused on finding other studies that could improve what analysed in the paper. We found two relevant papers that were published more recently and we will also discuss some possible state-of-the-art approaches.

5.1 A Universal End-to-End Approach to Portfolio Optimization via Deep Learning

Later the same year, two of the authors and others published a paper improving the analysis, titled "A Universal End-to-End Approach to Portfolio Optimization via Deep Learning". This second contribution represents an important evolution of the previous framework and transforms it into a study into a universal and scalable for asset management.

The main methodological innovation can be immediately seen in the more complex two-layer neural architecture, which allows to overcome some structural limitations of the first model. The original work used a single-stage LSTM to output weights directly, while in this framework there are two separated functional blocks. The first block is as a scoring layer (using LSTMs or fully connected networks) to extract features from historical data and assign a relative "preference score" to each asset. Then the second block consists of specialized constraint and sorting layers that map these scores into final portfolio weights. The clear improvement lies in the use of the NeuralSort operator, that is differentiable and enables the management of cardinality constraints.

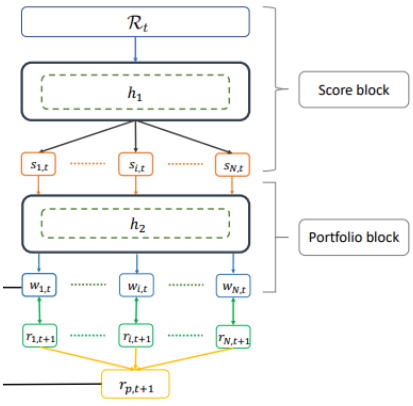


Figure 9: Network architecture

This improved network also allows to have short positions, maximum position limits and leverage control and has the flexibility of optimizing various objective functions. The model is tested on a big number of stocks from the Russel 3000 Index, this shows both the scalability of the new model and also the robustness of the results. This deep learning strategy significantly outperforms benchmarks and behaves well also during stress periods showing how and end-to-end framework can be a real and powerful tool.

	E(R)	Std(R)	Sharpe	DD(R)	Sortino	MDD	% of +Ret	Frobenius norm	Turnover
MSRP									
CS-SAMPLE	0.011	0.018	0.639	0.012	0.932	0.072	0.517	4.695	2.006
CS-LM	0.001	0.011	0.087	0.008	0.123	0.037	0.501	4.700	9.635
CS-MLP	0.001	0.017	0.011	0.012	0.016	0.082	0.497	4.873	9.518
CS-LSTM	0.001	0.011	0.053	0.008	0.074	0.086	0.504	4.684	6.913
CS-CNN	0.002	0.011	0.160	0.008	0.227	0.053	0.500	4.685	7.214
E2E-LM	0.004	0.031	0.146	0.022	0.206	0.119	0.513	**4.253**	7.203
E2E-MLP	0.003	0.012	0.238	0.009	0.339	0.056	0.516	4.280	5.849
E2E-LSTM	0.014	0.012	**1.216**	0.008	**1.810**	0.008	0.529	4.407	0.692
E2E-CNN	0.014	0.014	0.977	0.009	1.441	0.027	0.534	4.426	3.604

Table 2: Experimental results for the synthetic data set.

Figure 10: Performance summary

5.2 A Deep Learning-Based Model for Return Rate Prediction and Portfolio Optimization

A different perspective is offered by the paper "A Deep Learning-Based Model for Return Rate Prediction and Portfolio Optimization" by C. Ellery. While the previously discussed works by Zhang et al. advocates for an end-to-end strategy to bypass return forecasting, this study revisits the traditional two-step optimization process, enhancing it with state-of-the-art architectures. The methodology introduces transformer models for high-precision return rate prediction, followed by a classical Mean-Variance optimization.

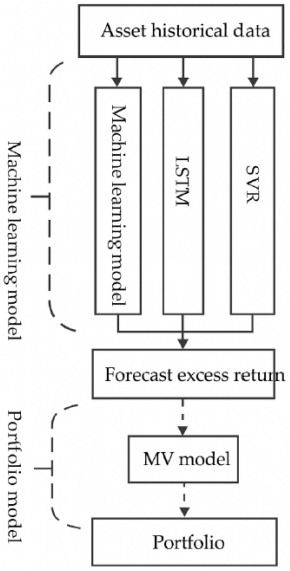


Figure 11: Network architecture

The core innovation is that the Transformer can capture global dependencies and "long-range" correlations within market data more effectively than the classical LSTM avoiding the vanishing gradient issues typical of recurrent networks. Furthermore, this model introduces a more granular approach to data, incorporating not only historical prices but also technical indicators, thereby providing a more robust input feature set than the 4-ETF model. One important characteristic is that this approach provides greater interpretability by allowing the analyst to evaluate the accuracy of the return predictions before the portfolio is constructed.

In the paper the author makes a comparison between Transformer, LSTM and SVR models, the results demonstrate that the Transformer-based model achieves superior predictive accuracy (lower MSE) and higher cumulative returns. In terms of portfolio performance, while Zhang's first paper showed the resilience of LSTMs during the 2020 crash, Ellery's work suggests that a Transformer-driven two-step approach can lead to even higher Sharpe Ratios by more accurately identifying market turning points. However, this comes at the cost of increased computational complexity and the reintroduction of the "estimation risk" associated with the covariance matrix, which the second paper by Zhang et al. had specifically sought to eliminate. This comparison highlights a fundamental trade-off in the literature: the choice between the seamless efficiency of end-to-end models and the granular predictive power of hybrid Transformer frameworks.

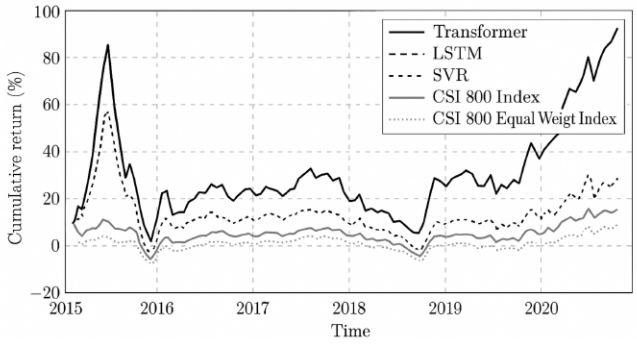


Figure 12: Portfolio performance

5.3 State of the Art

The progression across the three analyzed papers illustrates a clear shift in the financial machine learning landscape: from the use of LSTM networks to directly optimize the Sharpe Ratio in small-scale portfolios, to the development of Universal End-to-End frameworks capable of managing hundreds of assets through differentiable sorting layers like NeuralSort, and finally to the integration of Transformer architectures to capture complex, long-range temporal dependencies in returns.

Beyond the methodologies tested in these works, the current state-of-the-art is moving toward even more sophisticated paradigms:

- Graph Neural Networks (GNNs) that exploit structural correlation between assets treating them as interconnected nodes.
- Deep Reinforcement Learning (DRL) can be used to train adaptive strategies using non-differentiable reward functions like transaction cost or market impact.

- Generative Adversarial Networks (GANs) that are employed for data augmentation in order to stress-test optimization frameworks against "black swan" events not fully captured in historical datasets.
- Hybrid Attention-Optimization Models that maintain the high predictive power of self-attention while reducing the computational overhead for very long sequences.

In conclusion, the analyzed literature marks a definitive transition from traditional econometrics to end-to-end differentiable optimization and attention-driven forecasting. The integration of these techniques allows for a direct mapping from market noise to optimal, risk-adjusted asset allocation, paving the way for the next generation of automated, scalable asset management systems.

References

[1] David H. Bailey and Marcos López de Prado. "The Sharpe Ratio Efficient Frontier". In: *Journal of Risk* 15.2 (2012), pp. 3–44. DOI: 10.21314/JOR.2012.255.

[2] Cormac Ellery. "A Deep Learning-Based Model for Return Rate Prediction and Portfolio Optimization". In: *Preprints.org* (Nov. 2025). Not peer-reviewed version. DOI: 10.20944/preprints202511.1992.v1. URL: https://doi.org/10.20944/preprints202511.1992.v1.

[3] Olivier Ledoit and Michael Wolf. *Honey, I Shrunk the Sample Covariance Matrix*. UPF Economics and Business Working Paper 691. Posted: 18 Sep 2003. Universitat Pompeu Fabra, Department of Economics and Business, June 2003, p. 21. DOI: 10.2139/ssrn.433840. URL: https://ssrn.com/abstract=433840.

[4] Olivier Ledoit and Michael Wolf. "Honey, I Shrunk the Sample Covariance Matrix". In: *The Journal of Portfolio Management* 30.4 (2004), pp. 110–119. DOI: 10.3905/jpm.2004.110.

[5] Harry Markowitz. "Portfolio Selection". In: *The Journal of Finance* 7.1 (1952), pp. 77–91. DOI: 10.1111/j.1540-6261.1952.tb01525.x.

[6] Charles R. Nelson and Andrew F. Siegel. "Parsimonious Modeling of Yield Curves". In: *The Journal of Business* 60.4 (1987), pp. 473–489. DOI: 10.1086/296409.

[7] Marcos López de Prado. *Advances in Financial Machine Learning*. John Wiley & Sons, Feb. 2018, p. 400. ISBN: 9781119482086. URL: https://www.wiley.com/en-us/Advances\%2Bin\%2BFinancial\%2BMachine\%2BLearning-p-9781119482086.

[8] Marcos López de Prado. "Building Diversified Portfolios that Outperform Out of Sample". In: *The Journal of Portfolio Management* 42.4 (2016), pp. 59–69. DOI: 10.3905/jpm.2016.42.4.059.

[9] Chao Zhang et al. *A Universal End-to-End Approach to Portfolio Optimization via Deep Learning*. 2021. DOI: 10.48550/arXiv.2111.09170. arXiv: 2111.09170 [q-fin.PM]. URL: https://arxiv.org/abs/2111.09170.

[10] Zihao Zhang, Stefan Zohren, and Stephen Roberts. "Deep Learning for Portfolio Optimization". In: *The Journal of Financial Data Science* 2.4 (2020), pp. 8–20. DOI: 10.3905/jfds.2020.1.042. URL: https://www.pm-research.com/content/iijjfds/2/4/8.