Systematic Trading Strategies with Machine Learning Algorithms

Meta-Labeling, Calibration, and Neural Portfolio Optimization



June 1, 2026

Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

The competition, in one slide

You receive a primary model that emits a daily signal in $\{-1, 0, +1\}$ for 11 instruments across three asset classes.

Your job: build a meta-model that predicts, for each primary signal, the probability that following it would be profitable under a triple-barrier exit.

Judged on methodology, not P&L

- A high mark is possible even if the meta-model does not beat the primary
- Best research is presented to the Alken research team, with an internship interview

The universe: 11 instruments, 3 asset classes

Equity index

- ES1S – S&P 500
- NQ1S – Nasdaq 100
- FESX1S – Euro Stoxx 50

Energy

- CL1S – WTI crude
- HO1S – heating oil
- RB1S – RBOB gasoline
- NG1S – natural gas

Metals

- GC1S – gold
- SI1S – silver
- HG1S – copper
- PL1S – platinum

Covering more than one class (up to all 11) is optional and rewarded by breadth of analysis, not by raw performance.

The pipeline you must implement

1. **Feature engineering** from OHLCV and anything you can derive
2. **Labeling** via the triple-barrier method
3. **Training and comparing** several ML models with hyperparameter tuning
4. **Feature importance** at the cluster level
5. **Evaluation** on a clean out-of-sample period
6. **(Optional)** strategy construction on top of the meta-model probabilities

Marking scheme

Section	Marks
Feature engineering	20
Labeling (triple-barrier method)	20
Model development and comparison	30
Feature importance (cluster level)	10
Model evaluation	20
Total	**100**
Optional: strategy construction (competition)	+10

The bonus is capped so the final mark does not exceed 100. Model development carries the most weight, so the comparison across the three model families is where to invest effort.

What “model development” asks for

At least three models with hyperparameter tuning, one from each family.

- **Linear** – logistic regression with regularization
- **Tree-based** – random forest, XGBoost, LightGBM
- **Neural** – variable selection network or sequential nets

Present a clear comparison: which model wins, on which metric, and why.

The neural family is exactly where the VSN and TFT material from earlier in the course pays off, and where Part 5 of today’s lecture points for ideas.

The dataset and the hidden test

Two CSVs on Insendi:

- `ohlcv_data.csv` – daily OHLCV for all 11 instruments, history from 1990 (equity indices start later: ES1S 1997, FESX1S 1998, NQ1S 1999)
- `primary_signals.csv` – daily $\{-1, 0, +1\}$ signals from January 2020 onwards

The held-out six months

Released data covers up to 30 June 2022. The final six months (July to December 2022) are a hidden test set. We rerun your code on it. Anything that touches that period during development is leakage.

Deliverables: a notebook that runs end-to-end, plus a predictions CSV (date, instrument, probability) and, for the competition track, a weights CSV.

Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

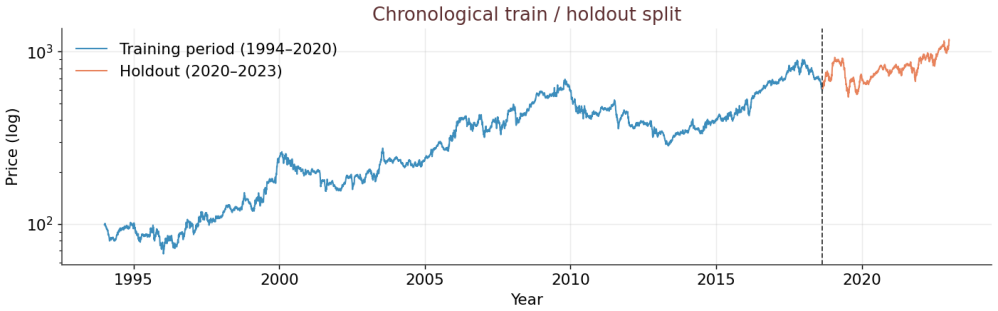
Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

Train versus holdout, before anything else

Carve out the holdout first: a chronologically-last slice the pipeline never sees until submission.



CPCV, hyperparameter selection, calibration, and threshold setting all happen on the training period only.

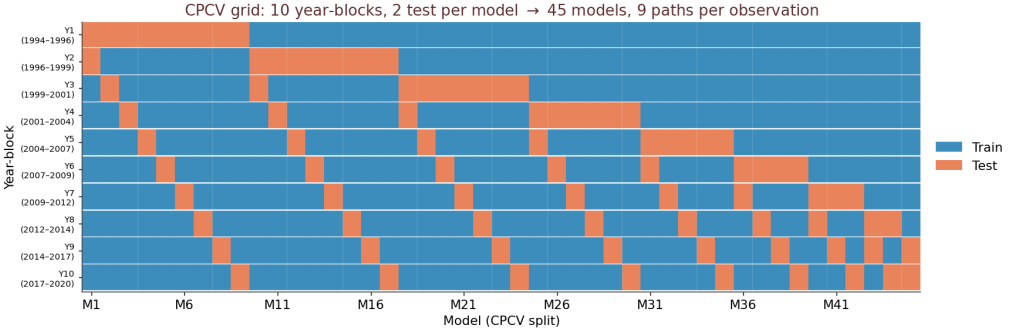
Why standard cross-validation fails on financial data

- Labels span multiple days, so train and test windows that overlap leak information at the sample level
- A single walk-forward path is too noisy to compare model configurations

Combinatorial Purged Cross-Validation (CPCV) solves both:

- Purge: drop training samples whose labels reach into the test window
- Embargo: drop training samples just after each test block
- Combinatorial: partition into N blocks, test on every k-subset

CPCV setup: 10 blocks, 2 test per model



$\binom{10}{2} = 45$ models, each trained on 8 blocks and tested on 2.

Each block is tested by $\binom{9}{1} = 9$ models, giving 9 paths per observation.

The two-stage hyperparameter optimization (a recommendation)

This is a recommended workflow, not a requirement. Use it as a starting point and adapt as you see fit.

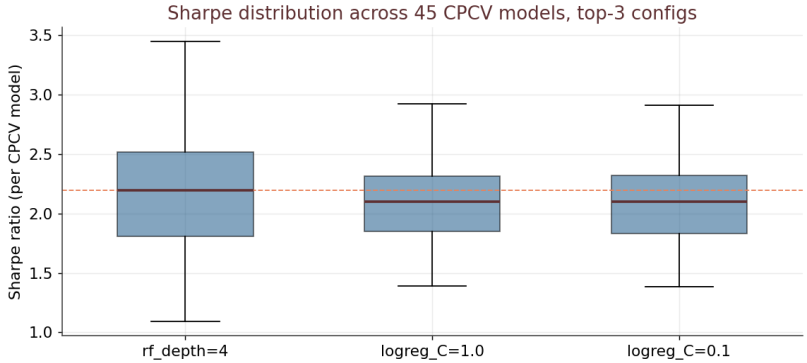
Stage 1 (statistical loss, low variance):

- Rank all configurations by mean AUC across the 45 CPCV models
- Keep the top $K = 3$

Stage 2 (economic loss, tiebreaker among survivors):

- For each of the top-K, plot the distribution of Sharpe ratios across the 45 paths
- Pick the configuration with the highest median and tightest spread

Sharpe distribution on the top-3 configurations (example)



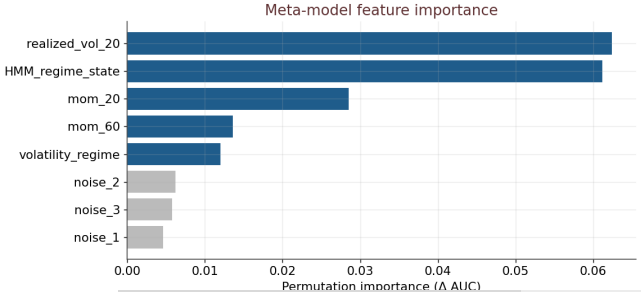
This is an illustrative example. The winner has the highest median Sharpe across the 45 CPCV models, after AUC has already narrowed the field to three credible candidates.

Feature importance: a required sanity check

The coursework asks you to implement feature importance at the cluster level (Section 4 of the marking scheme). The point is to confirm that the meta-model is using economically meaningful structure, not artefacts.

A good submission shows that:

- economically motivated features (e.g. a regime-state estimate, a volatility regime) sit near the top
- deliberately injected noise features fall to the bottom



Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

The core idea

Two-model architecture:

- A primary model decides *direction* $(+1, -1, 0)$
- A meta-model decides *whether to take each primary signal*

Why this works: the primary model often has skill only in certain regimes. The meta-model learns to filter out signals from regimes where the primary is unreliable.

Crucially: the meta-model is binary, not directional. It is far easier to learn "is this trade worth taking?" than "which direction will the market move?"

The meta-model objective, precisely

Let the primary emit a side $s_i \in \{-1, +1\}$ at signal i. The triple-barrier rule returns a binary outcome

$$y_i = \mathbf{1}\{\text{the trade hit its profit barrier first}\}.$$

The meta-model is a probabilistic classifier

$$\hat{p}_i = \Pr(y_i = 1 \mid x_i), \qquad x_i = \text{features at signal } i.$$

This reframes a hard regression into an easy classification.

- The primary owns direction. The meta-model only gates it.
- Output is a probability in $[0, 1]$, exactly the deliverable the coursework asks for.
- Everything downstream (threshold, sizing) is a function of $\hat{p}$.

From primary signal to labeled trades

The triple-barrier method converts each primary signal into a clean ML label.

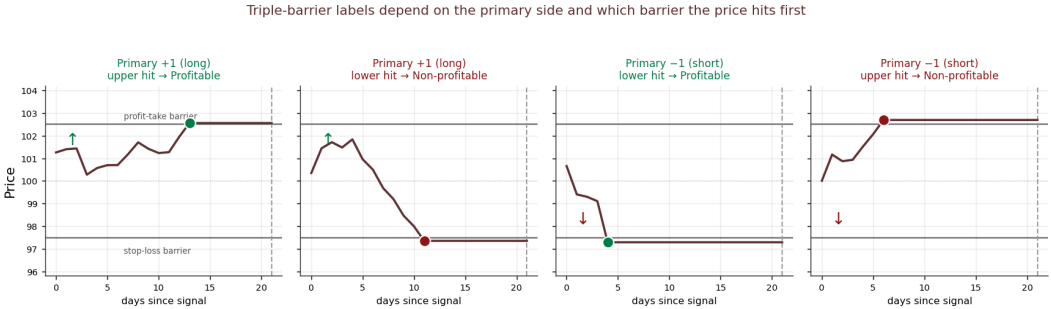
The label depends on the primary side and which barrier the price reaches first:

- **Profitable** ($y = 1$): a long that hits the upper barrier, or a short that hits the lower barrier.
- **Non-profitable** ($y = 0$): a long that hits the lower barrier, or a short that hits the upper barrier.
- **Time barrier** ($t_{\max}$): position closed at horizon, labeled by its sign at exit.

Barriers scale with volatility σ, so labels are comparable across regimes.

Binary meta-label: was the primary signal worth following? The meta-model predicts $\hat{p} = \Pr(y = 1)$.

Triple-barrier labels: side × barrier hit



The primary side (arrow) and which barrier the *price* touches first jointly determine profitability. The same downward move is profitable for a short and a loss for a long.

A first use of the meta-model: a threshold

One simple decision rule: take the trade only when its expected value is positive,

$$p \cdot r_G + (1 - p) \cdot r_L > 0,$$

where r_G is the mean return on true positives and r_L on false positives. Solving for p in the general case (assuming $r_G > r_L$),

$$p > p^* = \frac{-r_L}{r_G - r_L}.$$

When true positives are profitable and false positives lose money ($r_G > 0 > r_L$), writing $G \equiv r_G$ and $L \equiv -r_L$ recovers the familiar form $p^* = \frac{L}{G+L}$.

Estimate r_G and r_L by bootstrapping true-positive and false-positive trade returns from the training folds, so the threshold inherits the bootstrap uncertainty.

This is just one example of a rule. Used this way, the meta-model filters trades: it sets the position to zero below p^* and keeps it otherwise. You are gating direction by adding zeros, not sizing the exposure. Sizing comes later, and needs calibrated probabilities.

The right comparison for evaluation

The question is not "is my meta-model accurate?" but "does my meta-model improve on following the primary blindly?"

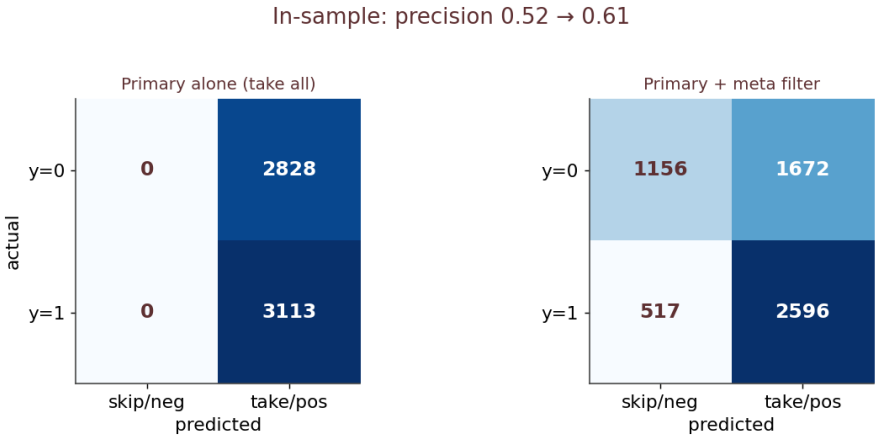
This evaluates the adding-zeros gate from the previous slide. Compare two confusion matrices on out-of-fold predictions:

- **Primary alone**: always predicts "trade". Recall $= 1$, precision $=$ base rate.
- **Primary + meta filter**: trade only when $\hat{p} \geq p^*$.

What to look for:

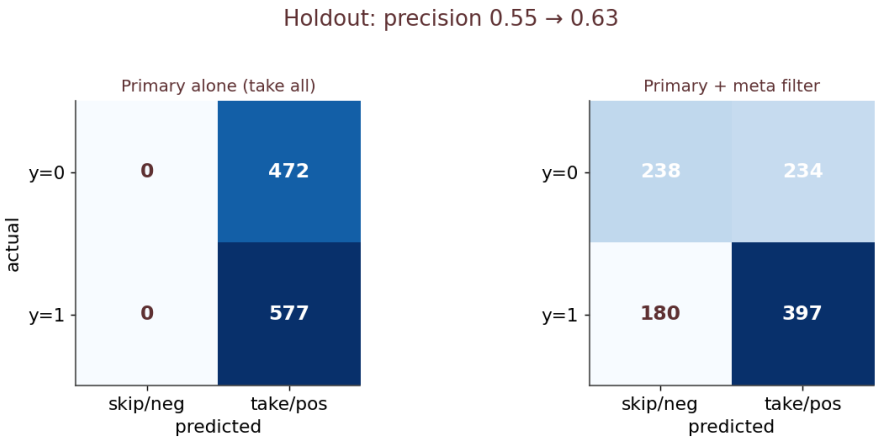
- False positives avoided (good)
- True positives missed (cost)
- Net effect should be positive and the hit rate should rise meaningfully

In-sample confusion: the meta as an "adding-zeros" gate



Used as a filter, the meta-model replaces low-probability trades with zeros. It trades precision for recall: it avoids many false positives at the cost of missing some true positives, and the net is a precision gain.

Holdout confusion (untouched until now)



The same adding-zeros gate, out of sample: the precision improvement holds, which is the result that matters.

From classification to P&L

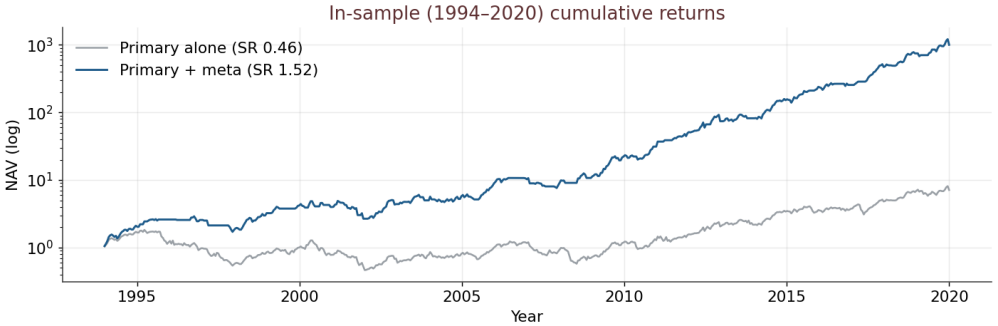
Classification metrics are a means, not the end. What we care about is the wealth path.

Two strategies, same trade pool:

- **Primary alone**: take every signal
- **Primary + meta**: take only signals with $\hat{p} \geq p^*$ (zeros elsewhere)

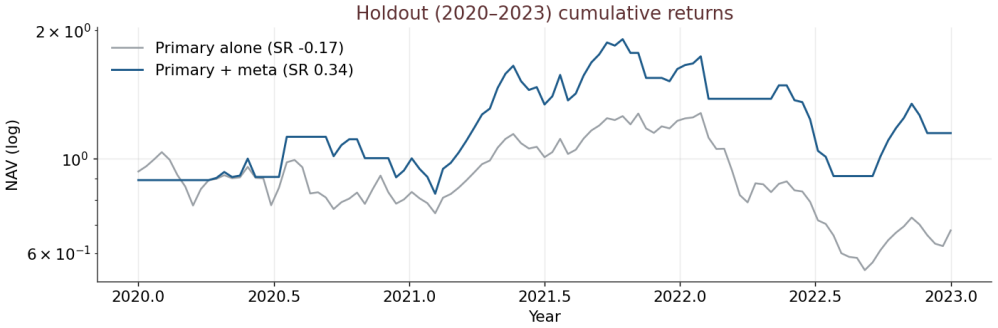
The meta strategy makes fewer but more selective trades. The question is whether dropping the low-probability signals improves the risk-adjusted return of what remains.

In-sample NAV



The meta-filtered strategy lifts Sharpe and reduces drawdown over the training period.

Holdout NAV



The flat sections are correct "sit out" decisions in regimes the meta-model distrusts. Out of sample, selection turns a roughly flat primary into a positive-Sharpe strategy.

From filtering to sizing

So far the meta-model has been used as a gate: keep a trade or replace it with a zero. Every kept trade is taken at full size.

A natural next question: instead of a binary keep/drop, can we let the probability size the position, putting more capital behind trades the meta-model trusts more?

A trade with $\hat{p} = 0.95$ and one with $\hat{p} = 0.55$ both clear $p^* = 0.51$, yet they are not equally good. To size on $\hat{p}$ we first need it to mean what it says: a stated 0.7 should correspond to a 70% hit rate.

That is calibration, the subject of the next section.

Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

Why calibration matters

The meta-model outputs $\hat{p} \in [0, 1]$. To use it in any sizing rule we need it to mean what it says:

$$\Pr(y = 1 \mid \hat{p}) \approx \hat{p}.$$

Most ML models are not calibrated by default:

- Random forests pull toward 0.5 (under-confident at the extremes)
- SVMs and AdaBoost are sigmoid-distorted, also pulled toward 0.5: this is precisely the case Platt scaling was invented to fix
- Naive Bayes pushes toward $\{0, 1\}$ (over-confident)
- Modern gradient boosting (XGBoost, LightGBM) tends to over-confidence too, pushing toward $\{0, 1\}$
- Logistic regression is usually decent

Fix: fit a 1-D monotone map on out-of-fold pairs $(\hat{p}_i, y_i)$.

Calibrating the probabilities: Platt scaling

Platt scaling fits a 1-feature logistic regression predicting y from the raw score $\hat{p}$:

$$f(\hat{p}) = \sigma(a\,\hat{p} + b).$$

Two parameters, one `LogisticRegression.fit`. It assumes the miscalibration is sigmoid-shaped, and is robust on the small samples typical of a single instrument's trade history.

Where to fit it. On out-of-fold pairs $(\hat{p}_i, y_i)$ from CPCV: each $\hat{p}_i$ comes from a model that did not see sample i. Fitting on in-sample predictions learns "already calibrated" and the map is wrong on new data.

An alternative is isotonic regression, a non-parametric monotone step function. It is more flexible but data-hungry and prone to overfitting the steps on small calibration sets, so Platt scaling is the safer default here.

From a binary gate to continuous sizing

Once $\hat{\rho}$ is calibrated, we can use it directly to allocate capital rather than just gate:

$$b_i = g(\hat{\rho}_i), \qquad g : [0,1] \to [0,1].$$

Two families of choices for g: fixed methods that need no training, and estimated methods fit on the training set. We write each one algorithmically next.

Three fixed position-sizing methods

No training. Each maps the calibrated probability $\hat{p}$ to a size $b \in [0, 1]$, and returns 0 when $\hat{p} \leq 0.5$ (the side is wrong on average there).

function ModelConfidence($\hat{p}$) ▷ size equals the probability
    return $\hat{p} \cdot \mathbf{1}\{\hat{p} > 0.5\}$
end function
function AllOrNothing($\hat{p}$) ▷ the binary gate: full size or none
    return $\mathbf{1}\{\hat{p} > 0.5\}$
end function
function NCDF($\hat{p}$) ▷ standardize, then push through the normal CDF
    if $\hat{p} \leq 0.5$ **then**
        return 0
    end if
    $z \leftarrow (\hat{p} - \frac{1}{2}) \,/\, \sqrt{\hat{p}\,(1 - \hat{p})}$
    return $\Phi(z)$
end function

All three are monotone in $\hat{p}$. All-or-nothing ignores *how far* above 0.5 the probability sits; model confidence and NCDF reward higher conviction, NCDF most aggressively near $\hat{p} = 1$.

Three estimated position-sizing methods

Fit on the training probabilities $\{p_{\text{tr}}\}$ and returns $\{r_{\text{tr}}\}$, then applied to a new $\hat{p}$ (return 0 when $\hat{p} \le 0.5$).

function $\textsc{LinearScaling}(\hat{p};\ p_{\text{tr}})$ ▷ stretch the realized range to $[0, 1]$
  return $(\hat{p} - \min p_{\text{tr}}) \,/\, (\max p_{\text{tr}} - \min p_{\text{tr}})$
end function
function $\mathrm{ECDF}(\hat{p};\ p_{\text{tr}})$ ▷ size by the training percentile of $\hat{p}$
  return $\widehat{\mathrm{ECDF}}_{p_{\text{tr}}}(\hat{p})$
end function
function $\mathrm{SOPS}(\hat{p};\ p_{\text{tr}}, r_{\text{tr}})$ ▷ Sharpe-optimal position sizing
  $f_{a,c}(p) \leftarrow \left(1 + e^{-(ap-c)}\right)^{-1}$
  $m_{a,c} \leftarrow f_{a,c}(p_{\text{tr}}) \odot r_{\text{tr}}$
  $(a^\star, c^\star) \leftarrow \arg\max_{a,c}\ \mathrm{mean}(m_{a,c})/\mathrm{std}(m_{a,c})$
  return $f_{a^\star,c^\star}(\hat{p})$
end function

Linear scaling and ECDF are distribution-aware but ignore returns; SOPS fits the sizing curve directly to training Sharpe, learning both the threshold and the steepness from $(p_{\text{tr}}, r_{\text{tr}})$.

Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

Transition: from a single signal to a portfolio

So far we have, for each instrument, a primary side and a calibrated meta-probability. A natural next question:

> *What if we gather everything we know about every instrument, the primary signal, the meta-probability, and any additional features, and learn a portfolio directly?*

This is the program of a recent Oxford benchmark.[1]

Key reframing. The method is agnostic to the features. It does not care whether x_t holds raw technicals or our meta-signals. It is a recipe for portfolio construction: map per-asset feature windows to positions, and train the whole thing by maximizing the Sharpe ratio.

[1] A. Saly-Kaufmann, K. Wood, J.-P. Calliess, S. Zohren, "Deep Learning for Financial Time Series: A Large-Scale Benchmark of Risk-Adjusted Performance", Machine Learning Research Group, Department of Engineering Science, and Oxford-Man Institute of Quantitative Finance, University of Oxford.

Problem setup and notation

- **Universe.** K instruments, indexed $k = 1, \ldots, K$.
- **Time.** Trading days $t = 1, \ldots, T$ in the training period.
- **Features.** For each asset k, a multivariate series $x_{t,k} \in \mathbb{R}^d$ of end-of-day features (prices, returns, technical indicators).
- **Lookback.** A fixed window of length L:

$$X_{t,k} = \left[x_{t-L+1,\,k},\ \ldots,\ x_{t,k} \right] \in \mathbb{R}^{L \times d}.$$

- **Returns.** $r_{t,k} = \dfrac{P_{t,k} - P_{t-1,k}}{P_{t-1,k}}$ is the daily return of asset k.

Goal: an allocation, in two stages.

1. **Per-instrument signal.** Learn $f_\theta : \mathbb{R}^{L \times d} \to [-1, 1]$ mapping each asset's window to a directional conviction $\hat{y}_{t,k}$, with $+1$ a full long and -1 a full short.
2. **Allocation.** Convert the K signals into portfolio weights $w_{t,k}$ (next slides). The final object is a vector of positions across the universe, not a single forecast.

Trading signal generation

A modular, two-stage map f_θ, shared across all architectures. Over the next slides we assemble, piece by piece, the objective the network is trained on.

Stage 1, sequence model. A candidate architecture g_ϕ (LSTM, PatchTST, Mamba2, ...) processes the window into a temporal state of fixed dimension H:

$$h_{t,k} = g_\phi(X_{t,k}), \qquad h_{t,k} \in \mathbb{R}^H.$$

Stage 2, projection head. A shared linear map followed by tanh bounds the output:

$$\hat{y}_{t,k} = \tanh\left(w_{\text{lin}}^\top h_{t,k} + b_{\text{lin}}\right) \in [-1, 1], \qquad w_{\text{lin}} \in \mathbb{R}^H,\ b_{\text{lin}} \in \mathbb{R}.$$

$\hat{y}_{t,k}$ is the model's directional conviction for asset k at time t. A ticker embedding is added so the shared model can specialize per instrument. Only g_ϕ changes across the benchmark; the head, sizing, and loss are identical, which makes the comparison fair.

Volatility estimation (the building block)

Before sizing, we need an ex-ante volatility for each asset. Estimate it with an EWMA. Let $\lambda = \dfrac{2}{\text{span} + 1}$. The conditional mean and variance evolve as

$$\mu_{t,k} = \lambda\, r_{t,k} + (1 - \lambda)\, \mu_{t-1,k},$$
$$\sigma^2_{t,k} = \lambda\, (r_{t,k} - \mu_{t,k})^2 + (1 - \lambda)\, \sigma^2_{t-1,k}.$$

Everything is ex-ante: $\sigma_{t,k}$ uses information up to and including day t, never beyond.

From a model prediction to a portfolio weight. The model **input** is the window; its **output** is the signal $\hat{y}_{t,k} \in [-1, 1]$. Volatility targeting then turns that signal into a weight:

Algorithm Weight from a prediction

Require: window $X_{t,k} \in \mathbb{R}^{L \times d}$, model f_θ, ex-ante vol $\sigma_{t,k}$, target vol σ_{tgt}
Ensure: weight $w_{t,k}$

1: $\hat{y}_{t,k} \leftarrow f_\theta(X_{t,k}) \in [-1, 1]$ ▷ input: window; output: signal
2: $w_{t,k} \leftarrow \hat{y}_{t,k} \cdot \dfrac{\sigma_{\text{tgt}}}{\sigma_{t,k}}$ ▷ scale conviction to target risk
3: **return** $w_{t,k}$

Portfolio construction via volatility targeting

Assets have wildly different baseline volatilities, so a raw signal of $+1$ on natural gas and $+1$ on gold are not comparable risks. Volatility targeting equalizes them.

Define the time-varying leverage factor

$$\text{vs_factor}_{t,k} = \frac{1}{\sigma_{t,k}}.$$

Given a constant per-asset target volatility σ_{tgt} (set to 10%), the realized weight is the conviction scaled to target risk:

$$\boxed{w_{t,k} = \hat{y}_{t,k} \cdot \frac{\sigma_{\text{tgt}}}{\sigma_{t,k}} = \hat{y}_{t,k} \cdot \sigma_{\text{tgt}} \cdot \text{vs_factor}_{t,k}}$$

Low-vol assets are scaled *up*, high-vol assets scaled *down*, so each contributes comparable risk to the book.

From weights to portfolio returns

The daily gross return of asset k, realized at $t+1$, is the position held at the end of day t times the next-day asset return:

$$R_{t+1,k} = w_{t,k} \cdot r_{t+1,k}.$$

Assuming equal risk-capital allocation across the K active assets, the aggregate daily gross portfolio return is the cross-sectional average:

$$R_{t+1}^{\text{port}} = \frac{1}{K} \sum_{k=1}^{K} R_{t+1,k} = \frac{1}{K} \sum_{k=1}^{K} w_{t,k}\, r_{t+1,k}.$$

This single number per day is the entire input to the loss. There is no supervised target: the network is trained by optimizing a function of $\{R_t^{\text{port}}\}$ directly, as the next slide shows.

Deriving the loss: maximize the Sharpe ratio

Unlike forecasting models that minimize a prediction error, here we optimize risk-adjusted economic performance directly. Over a training sequence of length T, with $R^{\text{port}} = \{R_1^{\text{port}}, \ldots, R_T^{\text{port}}\}$, define the sample moments

$$\hat{\mathbb{E}}[R] = \frac{1}{T}\sum_{t=1}^{T} R_t^{\text{port}}, \qquad \hat{\mathrm{Var}}[R] = \frac{1}{T}\sum_{t=1}^{T} \left(R_t^{\text{port}} - \hat{\mathbb{E}}[R]\right)^2.$$

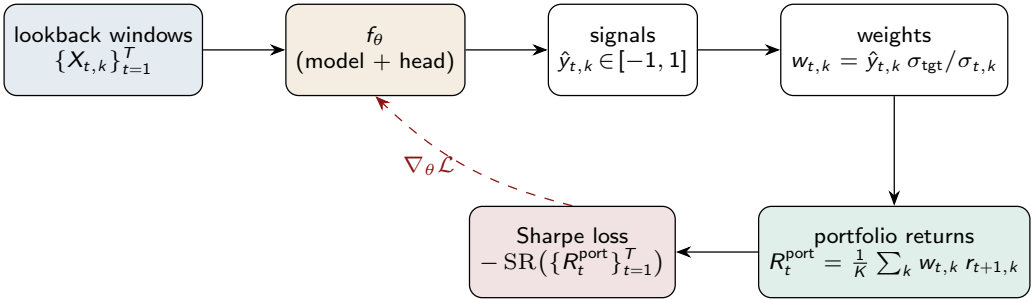
The loss is the negative differentiable annualized Sharpe ratio:

$$\boxed{\mathcal{L}(\theta) = -\,\mathrm{SR} = -\,\frac{\hat{\mathbb{E}}[R]}{\sqrt{\hat{\mathrm{Var}}[R] + \epsilon}}\,\sqrt{252}}$$

with 252 trading days per year and ϵ for numerical stability. Minimizing $\mathcal{L}$ maximizes expected return while penalizing variance.

How the training data is built over a window of length T

A "training example" is not one (window $\to$ label) pair. The loss is a function of the whole return path over the window T: roll f_θ across every day, collect the portfolio returns, then score them with a single Sharpe ratio.



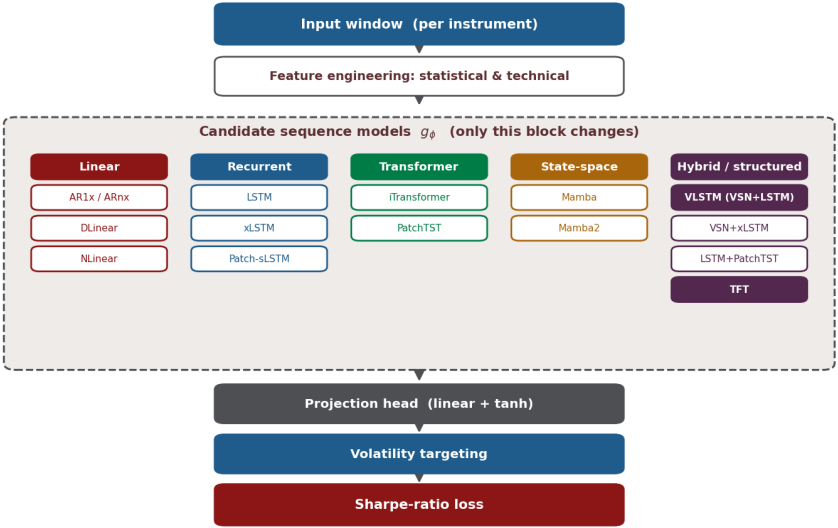
One forward pass produces a return for *every* day in T; the Sharpe is computed once on the pooled series, so gradients flow from the whole path back into f_θ.

The end-to-end recipe, in steps

For a chosen sequence model g_ϕ, the whole pipeline is:

1. Build causal features $x_{t,k}$ and stack a lookback window $X_{t,k} \in \mathbb{R}^{L \times d}$.
2. Estimate ex-ante volatility $\sigma_{t,k}$ by EWMA.
3. Forward pass: $h_{t,k} = g_\phi(X_{t,k})$, then $\hat{y}_{t,k} = \tanh(w_{\text{lin}}^\top h_{t,k} + b_{\text{lin}}) \in [-1, 1]$.
4. Size by volatility targeting: $w_{t,k} = \hat{y}_{t,k}\, \sigma_{\text{tgt}} / \sigma_{t,k}$.
5. Aggregate: $R_t^{\text{port}} = \frac{1}{K} \sum_k w_{t,k}\, r_{t+1,k}$ for every day in T.
6. Score the pooled path with one Sharpe ratio and backpropagate $\nabla_\theta(-\mathrm{SR})$.

The architectures benchmarked



The TFT: covered in our course

The Temporal Fusion Transformer combines a recurrent encoder for local dynamics with interpretable multi-head attention across time, plus variable selection networks.

It is a strong, structured benchmark for financial forecasting and one of the top performers in the study.

We covered the TFT in detail in the course:

- Lecture 7: *click here for the slides*
- Course materials: *BUSI70575 GitHub page*

Outline

The Coursework

Hyperparameter Optimization

The Meta-Model

Calibration and Position Sizing

Portfolio Construction with Neural Networks

Bridging Both: Meta-Signals as Features

The combined feature vector

Part 5's recipe was deliberately agnostic to the features. Here we make them concrete: feed it the outputs of our meta-labeling pipeline. For each instrument k and time t, the per-step input becomes

$$x_{t,k} = \Big[\underbrace{\text{technical / statistical features}}_{d \text{ channels}},\ \underbrace{s_{t,k}}_{\text{primary side}},\ \underbrace{\hat{p}_{t,k}}_{\text{meta prob.}}\Big] \in \mathbb{R}^{d+2}.$$

Two separate channels, deliberately:

- $s_{t,k} \in \{-1, 0, +1\}$ carries direction.
- $\hat{p}_{t,k} \in [0, 1]$ is direction-agnostic by construction. Use the out-of-fold CPCV probabilities from Part 2: in-sample $\hat{p}$ is over-confidently correct and poisons the net with lookahead.
- Feeding them separately lets the network learn how trust should modulate direction, rather than forcing a fixed product $s \cdot \hat{p}$.

Everything downstream is exactly the Part 5 pipeline, now reading $d + 2$ features.

Volatility estimation (EWMA)

The pipeline needs a causal, ex-ante volatility per asset, computed before any weight. We reuse the same EWMA recurrence from Part 5, here written out in full.

Algorithm EWMA conditional mean and volatility

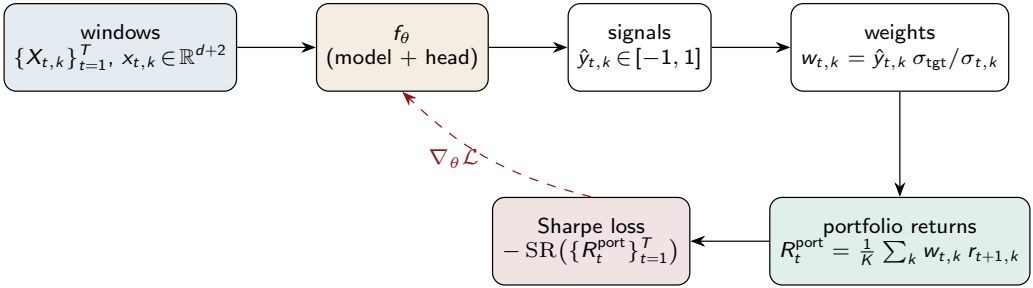
Require: Daily returns $\{r_t\}$, span (e.g. 60)
Ensure: Conditional volatility $\{\sigma_t\}$

$\lambda \leftarrow \dfrac{2}{\text{span} + 1}$
Initialize $\mu_0 \leftarrow r_0, \quad \sigma_0^2 \leftarrow \text{Var}(r_{0:21})$
for $t = 1, 2, \ldots$ **do**
    $\mu_t \leftarrow \lambda\, r_t + (1 - \lambda)\, \mu_{t-1}$ ▷ conditional mean
    $\sigma_t^2 \leftarrow \lambda\, (r_t - \mu_t)^2 + (1 - \lambda)\, \sigma_{t-1}^2$ ▷ conditional variance
end for
return $\sigma_t \leftarrow \sqrt{\sigma_t^2}$ for all t

σ_t is reused throughout: it scales the features and sizes the position.

Forward propagation, recap

Training rolls the combined-feature windows through the network and scores the whole path. This is the Part 5 flow, now with $d + 2$ input channels.



The pooled Sharpe over the window T is the only objective; the training algorithm follows.

Training by maximizing the Sharpe ratio

The objective comes first: we train end to end so the portfolio Sharpe is maximized.

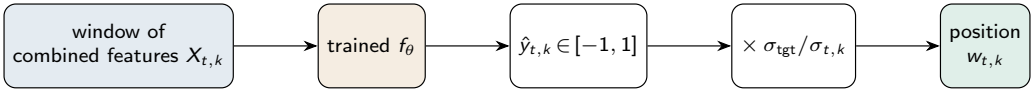
Algorithm End-to-end Sharpe optimization (DeePM pooling)

Require: Windows $\{X_{t,k}\}$, returns $\{r_{t+1,k}\}$, epochs
Ensure: Trained parameters $\theta = (\phi, w_{\text{lin}}, b_{\text{lin}})$

for epoch $= 1, \ldots,$ epochs **do**
  for each minibatch of sequences **do**
    Forward pass $\rightarrow$ weights $\{w_{t,k}\}$, then $R^{\text{port}}_{t+1} = \frac{1}{K}\sum_k w_{t,k}\, r_{t+1,k}$
    $\hat{\mathbb{E}}[R] \leftarrow \frac{1}{T}\sum_t R^{\text{port}}_t, \quad \hat{\mathrm{Var}}[R] \leftarrow \frac{1}{T}\sum_t (R^{\text{port}}_t - \hat{\mathbb{E}}[R])^2$
    $\mathcal{L}(\theta) \leftarrow -\dfrac{\hat{\mathbb{E}}[R]}{\sqrt{\hat{\mathrm{Var}}[R] + \epsilon}}\sqrt{252}$ ▷ neg. annualized Sharpe
    $\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}(\theta)$ ▷ Adam, gradient clipping
  end for
  Early stop on validation Sharpe (patience 20)
end for
return θ

Inference: from signal to volatility-targeted weights

At deployment there is no loss and no backprop. The trained network maps each window to a conviction, which volatility targeting turns into a tradable weight.



The output is a vector of positions $\{w_{t,k}\}_{k=1}^{K}$ across the universe, ready to trade. The algorithm below makes the loop explicit.

Inference algorithm: position sizing via volatility targeting

Algorithm Signal to volatility-targeted weights (inference)

Require: Window $X_{t,k}$, sequence model g_ϕ, projection $(w_{\text{lin}}, b_{\text{lin}})$, target vol σ_{tgt}
Ensure: Portfolio weights $\{w_{t,k}\}$ for each instrument k

1: **for** each instrument k among the K active **do**
2: $\quad h_{t,k} \leftarrow g_\phi(X_{t,k})$ ▷ temporal state
3: $\quad \hat{y}_{t,k} \leftarrow \tanh\left(w_{\text{lin}}^\top h_{t,k} + b_{\text{lin}}\right) \in [-1, 1]$ ▷ conviction
4: $\quad \text{vs_factor}_{t,k} \leftarrow \dfrac{1}{\sigma_{t,k}}$
5: $\quad w_{t,k} \leftarrow \hat{y}_{t,k} \cdot \sigma_{\text{tgt}} \cdot \text{vs_factor}_{t,k}$ ▷ scale to target risk
6: **end for**
7: **return** $\{w_{t,k}\}_{k=1}^{K}$

The weight at t uses only information up to t. Realized returns $r_{t+1,k}$ are never seen at inference; they enter only the training loss.

Recap

1. **Coursework.** Build a meta-model over a primary signal for 11 instruments; deliver calibrated probabilities, marked on methodology.
2. **HP optimization.** Tune with CPCV: purge, embargo, rank by AUC, break ties on the Sharpe distribution.
3. **Meta-model.** A binary classifier; label with triple-barrier, threshold via $L/(G+L)$ to filter by adding zeros, evaluate against the blind primary.
4. **Calibration.** Calibrate on out-of-fold pairs, then size continuously; smoother methods and calibration improve risk-adjusted outcomes.
5. **Neural portfolio.** An end-to-end recipe learns positions directly by maximizing Sharpe; structured recurrent architectures win.
6. **Bridge.** Feed the primary side and out-of-fold meta-probability as two channels, then learn sizing and allocation end-to-end.

Thank you for your attention