Quant-Style Cheat-Sheet
Machine-Learning Methods from
"Advanced Financial Market Analysis with ML and GenAI"

Contents

1 Hidden Markov Model (HMM) - Regime Detection

We model the market as a latent Markov chain because observed returns are noisy emissions of unobservable states. The chain lets us probabilistically forecast regime switches instead of hard-labeling them. Baum-Welch gives the MLE of transition matrix and per-regime moments - ready for dynamic allocation.

Explanation: The core idea is that market regimes (e.g., Bull, Bear, Choppy) are hidden states that we cannot observe directly. We can only observe their consequences: asset returns. An HMM assumes the current regime depends only on the previous regime (the Markov property) and that the returns in each regime are generated from a specific probability distribution (e.g., a Gaussian with a certain mean and volatility).

How it Works: 1. **Initialization:** Start with guesses for the transition matrix A and the regime parameters (μ_k, Σ_k). Often, these are initialized using a GMM. 2. **Expectation Step (E-Step):** Use the current parameters to compute the probability of being in each regime at every time point ($\gamma_t(k)$) and the probability of transitioning from one regime to another between time points. This is done efficiently with the Forward-Backward algorithm. 3. **Maximization Step (M-Step):** Update the parameters. The new mean for regime k, μ_k^{new}, is a weighted average of all returns, where the weights are the probabilities $\gamma_t(k)$ that each return belonged to regime k. The transition probabilities A_{ij} are updated based on the estimated number of transitions from i to j. 4. **Iteration:** Repeat E and M steps until the log-likelihood converges.

Application: The output is a smooth, probabilistic classification of each day into a regime. This is used to dynamically adjust portfolio allocations: e.g., lower leverage and higher cash weight in a high-volatility "Bear" regime, or tilt towards momentum factors in a "Bull" regime.

Table 1: HMM mathematical objects

Latent sequence	$z_{1:T} \in \{0, 1, 2\}$
Emission likelihood	$p(r_t \mid z_t = k, \theta) = \mathcal{N}(r_t; \mu_k, \Sigma_k)$
Transition matrix	$A_{ij} = p(z_t = j \mid z_{t-1} = i)$
Complete log-lik	$\log p(r_{1:T}, z_{1:T} \mid \theta) = \log \pi_{z_1} + \sum_{t=2}^{T} \log A_{z_{t-1} z_t} + \sum_{t=1}^{T} \log \mathcal{N}(r_t; \mu_{z_t}, \Sigma_{z_t})$
M-step mean update	$\mu_k^{\text{new}} = \dfrac{\sum_t \gamma_t(k) r_t}{\sum_t \gamma_t(k)}$

2 Gaussian Mixture Model (GMM) - Static Regime Clustering

GMM ignores time; we use it as a cold-start initializer for HMM parameters. Responsibilities give soft cluster weights - useful for robust covariance estimates. A post-Viterbi pass imposes temporal consistency without re-estimating transitions.

Explanation: A GMM is a simple clustering technique that assumes the data is generated from a mixture of several Gaussian distributions. It is "static" because it has no memory; the probability of a data point being in a cluster is independent of the previous point's cluster. This makes it computationally simpler but less powerful for time series than an HMM.

How it Works: 1. The algorithm (using Expectation-Maximization) tries to find the parameters (π_k, μ_k, Σ_k) that maximize the likelihood of the observed data. 2. The "responsibility" $\gamma_t(k)$ is the model's soft, probabilistic assignment of observation r_t to component k. 3. The update for μ_k and Σ_k is identical to the HMM's M-step because, without a transition matrix, the HMM reduces to a GMM.

Application: Its primary use here is to provide good initial estimates for the HMM's emission parameters (μ_k, Σ_k), speeding up HMM convergence. The resulting GMM labels can be refined by applying the Viterbi algorithm (which finds the single most likely sequence of hidden states) to impose temporal smoothness.

Table 2: GMM core equations

Likelihood	$p(r_t) = \sum_{k=0}^{2} \pi_k \mathcal{N}(r_t; \mu_k, \Sigma_k)$
Responsibility	$\gamma_t(k) = \dfrac{\pi_k \mathcal{N}(r_t; \mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(r_t; \mu_j, \Sigma_j)}$
EM update	identical to HMM but no A_{ij} term

3 Temporal Convolutional Network (TCN) - Representation Learning

TCN gives exponential receptive field growth with causal padding - no look-ahead bias. We train it as a next-day return proxy task; the penultimate layer becomes a regime-agnostic feature vector. k-means on these vectors non-parametrically discovers regime clusters without Gaussian assumptions.

Explanation: TCNs are a type of CNN architecture designed for sequence modeling. Their key features are causal convolutions (they only look at past data, preventing data leakage) and dilated convolutions (which allow them to see a long history of data with relatively few layers). We are not primarily interested in its prediction; we use it as a *feature extractor*.

How it Works: 1. **Training:** A TCN is trained to predict the next day's return (r_{t+1}) from a window of past data ($r_t, r_{t-1}, ..., r_{t-n}$). This is a "proxy task" that forces the network to learn meaningful patterns in the time series. 2. **Embedding:** Once trained, we discard the final prediction layer. The activations of the last hidden layer (after global average pooling) form a compact vector representation (e_t) for each time window. This vector encodes the "state" of the market at time t as learned by the network. 3. **Clustering:** We then apply a simple clustering algorithm like k-means to these learned feature vectors $\{e_t\}$.

Application: This method discovers market regimes based on the *learned dynamics* of the price series, not on its static distribution. It can find complex, non-linear regimes that a Gaussian-based model (like HMM/GMM) might miss, such as different types of trend or mean-reversion.

Table 3: TCN mechanics

1-D causal dilated conv	$h_t^{(l+1)} = \mathrm{ReLU}\big(W^{(l)} * \mathrm{pad}_{\mathrm{causal}}(h^{(l)})\big),\ d_l = 2^l$
Receptive field	$1 + \sum_{l=0}^{L-1} (k-1)\, 2^l$
Objective	$\mathcal{L} = \dfrac{1}{T} \sum_t (\hat{r}_{t+1} - r_{t+1})^2$ (next-day proxy)
Embedding	global-avg-pool $\rightarrow e_t \in \mathbb{R}^{128}$
Clustering	k-means on $\{e_t\}$ (same $k = 3$)

4 UMAP + K-Means - Topology-Preserving Clustering

UMAP preserves fuzzy topological structure - captures non-linear manifolds better than t-SNE. We drop dimension to 2-10-D then k-means; local + global structure survives. Resulting labels smoothly interpolate between regimes without parametric density assumptions.

Explanation: High-dimensional data (e.g., the returns of 500 assets) often lies on a much lower-dimensional "manifold" (a curved surface). UMAP is a powerful non-linear dimensionality reduction technique designed to find this low-dimensional structure while preserving as much of the topological (neighborhood) relationships as possible. It's often better than PCA (which is linear) or t-SNE (which can distort global structure).

How it Works: 1. In high dimensions, it constructs a graph where data points are connected to their nearest neighbors with a probability (p_{ij}). 2. It then finds a low-dimensional projection (e.g., 2D) where the graph of similarities (q_{ij}) is as similar as possible to the high-dimensional one, by minimizing the cross-entropy between the two distributions. 3. The resulting 2D points $\{y_i\}$ form a "map" of the original high-dimensional data. Points close together on the map were close in the original space.

Application: By clustering this 2D map with k-means, we get regime classifications. This approach is very flexible because it doesn't assume data belongs to specific distributions (like Gaussians). The regimes are defined by the natural clustering of the data's intrinsic structure.

Table 4: UMAP objective

High-dim similarity	$p_{j\|i} = \exp\big(-(d_{ij} - \rho_i)/\sigma_i\big)$
Low-dim similarity	$q_{ij} = \big(1 + a\|y_i - y_j\|^{2b}\big)^{-1}$
Cross-entropy	$\mathcal{L} = \sum_{i\neq j}\Big[p_{ij}\log\frac{p_{ij}}{q_{ij}} + (1-p_{ij})\log\frac{1-p_{ij}}{1-q_{ij}}\Big]$
Gradient	$\partial\mathcal{L}/\partial y_i = 2ab\sum_j \frac{(p_{ij}-q_{ij})(y_i-y_j)\|y_i-y_j\|^{2b-2}}{1+a\|y_i-y_j\|^{2b}}$
Next step	k-means on 2-D embedding y_i

5 Long Short-Term Memory (LSTM) - Short-Signal Generator

LSTM gates let us remember bear-features for months; asymmetric loss penalises missed shorts more than false alarms. Regime-specific input gating up-weights features that predict crashes in Regime 2. Output is a probability; position is vol-scaled and hard-stopped.

Explanation: LSTMs are a type of RNN designed to learn long-term dependencies in sequential data. Their gating mechanism (Input, Forget, Output gates) allows them to selectively remember or forget information over long periods, which is crucial for detecting the slow build-up to a market crash.

How it Works: 1. **Architecture:** The gates control the flow of information. The *forget gate* (f_t) decides what to remove from the cell state. The *input gate* (i_t) decides what new information to add. The *output gate* (o_t) decides what to output based on the cell state. 2. **Asymmetric Loss:** A standard squared error loss equally penalizes false positives and false negatives. For a short signal, a missed crash (false negative) is much more costly than a false alarm (false positive). The asymmetric loss function incorporates a larger weight (α) for missed crashes, shaping the model to be more sensitive. 3. **Regime Gating:** The input features are element-wise multiplied by a vector that is specific to the current regime r_t. This focuses the model's attention on the most relevant features for the current market environment.

Application: The LSTM outputs a probability that a crash or significant drawdown is imminent. This probability is then converted into a short position size, which is volatility-scaled to manage risk and has a hard stop to limit losses if the prediction is wrong.

Table 5: LSTM cell and loss

Gate equations	$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$
	$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$
	$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$
	$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$
	$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
	$h_t = o_t \odot \tanh(c_t)$
Output	$\hat{y}_t = \sigma(W_s h_t + b_s)$
Asymmetric loss	$\mathcal{L} = \frac{1}{T}\sum_t \Big[\alpha\,(y_t - \hat{y}_t)^2 \mathbf{1}_{y_t=1, \hat{y}_t<y_t} + (y_t - \hat{y}_t)^2\Big]$
Regime gating	$x_t \leftarrow x_t \odot \mathrm{softmax}(W_r)_{:,r_t}$

6 Transformer Encoder - Momentum Strategy

Self-attention looks at every past day - no decay bias; regime embedding modulates the same weights per regime. We pool to a momentum score then vol-scale; no lookahead because causal masking is baked in. Transformer outperforms LSTM in Regime 0 (bull) due to long-range trend capture.

Explanation: Transformers use a self-attention mechanism, which allows them to weigh the importance of every element in the input sequence when processing another element. This is different from LSTMs/RNNs, which process data sequentially and can struggle with very long-term dependencies.

How it Works: 1. **Self-Attention:** For each time step, the model computes a weighted sum of the representations of all other time steps. The attention weights ($QK^\top$) determine how much focus to place on each past time step. This allows the model to directly connect distant time points that are relevant for a momentum signal. 2. **Positional Encoding:** Since self-attention is permutation-invariant, positional encodings (PE) are added to give the model information about the order of the sequence. 3. **Regime Conditioning:** A one-hot encoding of the current regime is added to the input, allowing the same model to slightly adjust its behavior based on the regime. 4. **Causal Masking:** To prevent look-ahead bias, the attention mechanism is masked. When calculating attention for time t, the model cannot attend to any future time steps $> t$.

Application: The Transformer is trained to predict next-period returns. Its output is interpreted as a momentum score. A high score suggests a strong positive momentum signal. This score is then converted into a volatility-scaled position. Its ability to capture long-range trends makes it particularly effective in bull markets.

7 Kalman Filter - Pairs-Trading Beta

Kalman recursively updates hedge ratio β_t with regime-dependent noise variances - adaptive to covariance breakdowns. Spread $s_t = p_t^A - \beta_t p_t^B$ is standardised; entry/exit thresholds scaled by regime volatility. Mean-reversion half-life shrinks in Regime 2 $\rightarrow$ faster exit.

Table 6: Transformer core

Multi-head attention	$\text{Attention}(Q,K,V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)V$
Positional encoding	$\text{PE}_{(t,2i)} = \sin(t/10000^{2i/d})$
Regime embedding	$E_t = X_t W_x + \text{PE}_t + \text{OneHot}(r_t)W_r$
Objective	weighted MSE $\mathcal{L} = \sum_t w_{r_t}(r_{t+1} - \hat{r}_{t+1})^2$

Explanation: The Kalman Filter is an optimal recursive algorithm for estimating the state of a linear dynamic system from noisy observations. In pairs trading, the "state" is the true, hidden hedge ratio β_t between two assets, which can change over time.

How it Works: 1. **State-Space Model:** The model has two equations: - **State Equation:** $\beta_t = \beta_{t-1} + \omega_t$. This assumes the hedge ratio evolves randomly (a "random walk"). - **Measurement Equation:** $p_t^A = \beta_t p_t^B + \varepsilon_t$. This says the price of asset A is a linear function of asset B's price, plus noise. 2. **Recursive Process:** The filter operates in a predict-update cycle. - **Predict:** Project the current state ($\hat{\beta}_{t-1}$) and its uncertainty (P_{t-1}) forward to get a prior estimate for the next time step ($\hat{\beta}_{t|t-1}, P_{t|t-1}$). - **Update:** When a new observation (p_t^A, p_t^B) arrives, calculate the "Kalman Gain" K_t. This weight determines how much to trust the new observation vs. the prior prediction. The prior is then updated with the measurement error to get the posterior state estimate ($\hat{\beta}_{t|t}$).

Application: The estimated $\hat{\beta}_{t|t}$ is used to calculate the spread $s_t = p_t^A - \hat{\beta}_{t|t} p_t^B$. This spread is expected to mean-revert. A key innovation here is making the noise variances (Q_{r_t}, R_{r_t}) regime-dependent. In high-volatility regimes (Regime 2), the filter is tuned to be more responsive to new data (faster mean-reversion half-life), allowing for quicker exits from trades.

Table 7: Kalman update

State-space	$\beta_t = \beta_{t-1} + \omega_t, \quad \omega_t \sim \mathcal{N}(0, Q_{r_t})$
	$p_t^A = \beta_t p_t^B + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, R_{r_t})$
Prediction	$\hat{\beta}_{t\|t-1} = \hat{\beta}_{t-1\|t-1}, \quad P_{t\|t-1} = P_{t-1\|t-1} + Q_{r_t}$
	$K_t = \frac{P_{t\|t-1} p_t^B}{(p_t^B)^2 P_{t\|t-1} + R_{r_t}}$
Update	$\hat{\beta}_{t\|t} = \hat{\beta}_{t\|t-1} + K_t(p_t^A - \hat{\beta}_{t\|t-1} p_t^B)$
	$P_{t\|t} = (1 - K_t p_t^B) P_{t\|t-1}$
Signal	$z_t = (p_t^A - \hat{\beta}_{t\|t} p_t^B - \mu_t)/\sigma_t$

8 Time-series GAN (TimeGAN) - Synthetic Price Paths

Adversarial + auto-encoder + supervised step-loss → synthetic paths that match autocovariance and volatility-clustering. Regime conditioning injected at generator input; discriminator sees entire sequence → no phase-shift artefacts. Used to double training sample in Regime 2 → +14.2

Explanation: Standard GANs struggle with time-series data because they don't enforce temporal consistency. TimeGAN addresses this by combining three losses: a standard GAN adversarial loss, a reconstruction loss from an autoencoder (to preserve static characteristics), and a supervised loss that ensures each step is predictable from the previous ones (to preserve dynamics).

How it Works: 1. An **autoencoder** first learns to compress and reconstruct the real time series, learning a latent representation. 2. The **Generator** G takes random noise and a regime

label c and produces a latent vector sequence $\hat{h}_{1:T}$. 3. A **Supervisor** network S is trained to predict the next latent step $\hat{h}_{t+1}$ from $\hat{h}_t$, adding a step-wise dynamic consistency loss ($\mathcal{L}_{\text{sup}}$). 4. The **Discriminator** D is trained to distinguish between real latent sequences ($h_{1:T}$ from the autoencoder) and fake ones ($\hat{h}_{1:T}$ from G). 5. The final synthetic time series is generated by passing $\hat{h}_{1:T}$ through the **Decoder** R from the autoencoder.

Application: It is used for data augmentation, especially for rare regimes like crashes (Regime 2). By generating realistic, regime-specific synthetic data, we can dramatically improve the performance of other models (like the LSTM short-signal generator) that would otherwise overfit on the small amount of real crash data.

Table 8: TimeGAN losses

Reconstruction	$\mathcal{L}_{\text{rec}} = \mathbb{E}\|X - R(E(X))\|^2$
Supervisor	$\mathcal{L}_{\text{sup}} = \sum_t \|S(\hat{h}_t) - \hat{h}_{t+1}\|^2$
Adversarial	$\min_G \max_D \mathbb{E}[\log D(h)] + \mathbb{E}[\log(1 - D(G(z, c)))]$
Final path	$\hat{x}_{1:T} = R(\hat{h}_{1:T})$

9 Conditional VAE (CVAE) - Regime-Conditional Generation

CVAE gives smooth latent interpolation → richer synthetic tails than GAN. beta-annealing prevents posterior collapse; regime vector biases latent mean → per-regime stylised facts preserved. Fast sampling (no iterative optimisation) → real-time augmentation.

Explanation: A Variational Autoencoder (VAE) is a generative model that learns a probabilistic mapping from data space to a latent space and back. The Conditional VAE (CVAE) makes this process dependent on a label—in this case, the market regime.

How it Works: 1. The **Encoder** q_ϕ takes an input time series x and a regime c and outputs parameters (μ, σ) defining a Gaussian distribution in latent space. 2. A latent vector z is sampled from this distribution: $z \sim \mathcal{N}(\mu, \sigma^2)$. 3. The **Decoder** p_θ takes the sampled z and the regime c and tries to reconstruct the original input x. 4. The loss function is the Evidence Lower Bound (ELBO). It has two parts: - **Reconstruction Loss:** Measures how well the decoder reconstructed the input. - **KL Divergence:** Measures how much the encoder's distribution (q_ϕ) diverges from the prior ($p(z|c)$), which is typically a standard Gaussian. This acts as a regularizer. 5. **Beta-Annealing:** Gradually increasing the weight β on the KL term during training prevents "posterior collapse," a problem where the encoder ignores the input and the model becomes a poor generator.

Application: Like TimeGAN, it's used for data augmentation. Its key advantage is a smooth and well-structured latent space, allowing for high-quality interpolation and generation of realistic "tail event" paths. It's also faster to sample from than a GAN.

Table 9: CVAE objective

Encoder	$q_\phi(z\|x, c) = \mathcal{N}(z; \mu(x, c), \text{diag}(\sigma^2(x, c)))$
Prior	$p(z\|c) = \mathcal{N}(0, I)$
Decoder	$p_\theta(x\|z, c) = \mathcal{N}(x; \hat{\mu}(z, c), \text{diag}(\hat{\sigma}^2(z, c)))$
ELBO	$\mathcal{L}_{\text{CVAE}} = \mathbb{E}_q[\log p_\theta(x\|z, c)] - \beta\, D_{\text{KL}}(q_\phi(z\|x, c)\|p(z\|c))$

10 FiLM Transformer - Best Generative Model

FiLM layers scale+shift every attention block per regime—same network specialises without retraining. Transformer expressivity captures high-order moments; FiLM conditions them per regime → best KS p-value. Training cost only +15

Explanation: FiLM (Feature-wise Linear Modulation) is a highly effective and parameter-efficient technique for conditioning a neural network on auxiliary information (like a regime label). Instead of creating separate models for each regime, a single FiLM-enabled model can adapt its behavior based on the input label.

How it Works: 1. A small **conditioning network** takes the regime one-hot vector c and outputs a set of gains $\gamma(c)$ and biases $\beta(c)$ for each feature channel in a layer. 2. Within the main Transformer network, **after each self-attention block**, the feature map F is transformed element-wise: $\text{FiLM}(F, c) = \gamma(c) \odot F + \beta(c)$. 3. This is a simple but powerful operation. The γ and β parameters can amplify, dampen, or shift the features, effectively telling the network "how to behave" in the given regime.

Application: This is used as the ultimate generative model for synthetic data. It combines the superior pattern-capturing ability of the Transformer architecture with the precise regime control of FiLM conditioning. The result is synthetic data that best matches the statistical properties (including higher moments like kurtosis) of real market data across all regimes, as measured by a high Kolmogorov-Smirnov (KS) test p-value.

Table 10: FiLM layer

FiLM layer	$\text{FiLM}(F, c) = \gamma(c) \odot F + \beta(c)$
Condition net	$\gamma(c) = W_\gamma c + b_\gamma, \quad \beta(c) = W_\beta c + b_\beta$
Placement	After every self-attention block

11 Regime Adversary - Ensuring Regime Fidelity

Adversary forces generator to match per-regime feature moments—prevents mode-collapse across regimes. Feature-matching removes gradient saturation; gradient penalty stabilises training like WGAN-GP. Ablation: +0.17 KS p-value & +22 % downstream F1 with adversary.

Explanation: A common failure mode ("mode collapse") in generative models is that they learn to generate good overall data but fail to generate examples for specific, rare classes (e.g., crash regimes). A regime adversary is a discriminator network added to the training process to specifically prevent this.

How it Works: 1. The main **Generator** $G(z, c)$ tries to create data for a specified regime c. 2. The **Regime Adversary** D_{reg} tries to do two things: a) distinguish real from fake data, and b) predict the correct regime c of the data. 3. The generator is thus trained to fool the adversary on both counts: its outputs must be realistic and must clearly belong to the requested regime c. 4. **Feature Matching** improves training stability. Instead of directly using the adversary's output, the generator tries to match the *intermediate feature statistics* of the real data. This provides a more robust learning signal. 5. **Gradient Penalty** (from WGAN-GP) is added to enforce the Lipschitz constraint, further stabilizing training.

Application: This is a critical add-on to the FiLM Transformer or any other generative model. It ensures that when we ask the model to generate "Regime 2" data, it produces data that is authentically characteristic of a crash, leading to more robust performance in downstream tasks.

Table 11: Adversary losses

Adversary loss	$\mathcal{L}_{\text{adv}} = -\mathbb{E}_{z,c}[\log D_{\text{reg}}(G(z,c),c)]$
Feature matching	$\|\mathbb{E}_{x\sim p_{\text{data}}(\cdot\|c)}[f_D(x)] - \mathbb{E}_z[f_D(G(z,c))]\|^2$
Gradient penalty	$\mathbb{E}_{\hat{x}}[(\|\nabla_{\hat{x}} D_{\text{reg}}(\hat{x},c)\|_2 - 1)^2]$

12 Quantile Random Forest - Direct VaR

No parametric assumption → captures fat tails & non-linearities. Leaf-node CDF weighted by visitation frequency → conditional quantile in one pass. Regime-weighted ensemble reduces 99

Explanation: Value at Risk (VaR) is a quantile of the predicted return distribution. A Quantile Random Forest (QRF) is a non-parametric method that directly estimates this conditional quantile, without making any assumptions about the underlying distribution (e.g., that it is Gaussian).

How it Works: 1. Grow an ensemble of decision trees on the data (e.g., predicting returns from features). 2. For a new sample x, drop it down all trees. The set of observations in the same leaf nodes as x forms a "neighborhood" of similar samples. 3. The prediction is not the mean of these observations, but a weighted empirical Cumulative Distribution Function (CDF). The weight $w_i(x)$ of each training point i is determined by how often it ended up in the same leaf as x across all trees. 4. The VaR is simply the negative of the $(1-\alpha)$-quantile of this weighted empirical CDF.

Application: This provides a direct, robust, and non-parametric estimate of VaR. To make it regime-aware, we first predict the probability $p(z = k|x)$ that the current state belongs to each regime. The final VaR is a weighted average of the VaR estimates from QRFs trained specifically on each regime, leading to a significant reduction in bias.

Table 12: QRF mechanics

Leaf weights	$w_i(x) = \frac{1}{T}\sum_{t=1}^{T} \frac{\mathbf{1}\{i\in\mathcal{L}_t(x)\}}{\lvert\mathcal{L}_t(x)\rvert}$
CDF estimate	$\hat{F}(y\|x) = \sum_{i=1}^{n} w_i(x)\mathbf{1}\{y_i \le y\}$
VaR	$\widehat{\text{VaR}}_\alpha(x) = -\hat{F}^{-1}(1-\alpha\|x)$
Regime weighting	$\widehat{\text{VaR}}_\alpha(x) = \sum_{k=0}^{2} p(z=k\|x)\,\widehat{\text{VaR}}_{\alpha,k}(x)$

13 REGARCH-EVT - Tail Risk

GARCH misses tails; GPD fits exceedances → correct 99 Regime-switching vol parameters adapt to vol-of-vol shifts. EVT threshold chosen via mean-excess stability → robust across crises.

Explanation: This is a hybrid model. REGARCH (Regime-Switching GARCH) models the dynamic volatility, but since it assumes a Gaussian or Student-t distribution, it still underestimates extreme tails. Extreme Value Theory (EVT) is a branch of statistics specifically designed to model the tails of distributions.

How it Works: 1. A REGARCH model is fitted to capture the time-varying volatility σ_t, with parameters that can change per regime. 2. The standardized residuals ($z_t = (r_t - \mu)/\sigma_t$) are analyzed. 3. EVT is applied to the tails of these residuals. We choose a threshold u and model all exceedances ($z_t - u$) using the Generalized Pareto Distribution (GPD). 4. The full distribution is a combination of the empirical body (below u) and the GPD tail (above u). 5. To get the VaR, we calculate the quantile from this combined distribution and then "re-scale" it back by multiplying by the current regime-dependent GARCH volatility σ_t.

Application: This is the gold standard for tail risk measurement in regime-switching environments. It is particularly crucial for accurately estimating VaR in high-volatility, high-stress regimes (Regime 2) where extreme events are more likely.

Table 13: REGARCH-EVT equations

Vol dynamics	$\sigma_t^2 = \omega_{z_t} + \sum_{i=1}^{p} \alpha_{i,z_t} \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_{j,z_t} \sigma_{t-j}^2$
GPD for exceedances	$G_{\xi_z,\beta_z}(y) = 1 - \left(1 + \xi_z \frac{y}{\beta_z}\right)^{-1/\xi_z}, \quad y > 0$
99 % VaR	$\mathrm{VaR}_\alpha = \mu_{z_t} + \sigma_t \left[u + \frac{\beta_{z_t}}{\xi_{z_t}} \left(\frac{n}{N_u}(1-\alpha) \right)^{-\xi_{z_t}} - 1 \right]$

14 Synthetic Regime Bootstrap - Scenario Generation

Block-bootstrap preserves vol-clustering; synthetic blocks lengthen left tail without over-fitting. Weight w_{syn} tuned via Kupiec back-test $\rightarrow$ 1.04 Used for 10-day 99

Explanation: Bootstrapping is a resampling technique. A block bootstrap is used for time series to preserve the serial correlation (volatility clustering) within blocks of data. However, the historical data may not contain enough severe crashes. This method augments the bootstrap with synthetic crash paths.

How it Works: 1. **Historical Blocks:** The historical data is split into blocks, and these blocks are categorized by regime. 2. **Synthetic Blocks:** Our generative models (TimeGAN, CVAE, FiLM) are used to create new, realistic blocks of data for each regime, especially creating many more "bad" blocks for Regime 2. 3. **Combined Bootstrap:** We create a new bootstrap distribution by sampling blocks from a combined dataset: a fraction $(1 - w_{\text{syn}})$ from the historical blocks and a fraction w_{syn} from the synthetic blocks. 4. The weight w_{syn} is not arbitrary; it is tuned via back-testing to achieve the desired exceedance rate (e.g., 1% for 99% VaR).

Application: This is used for scenario analysis and stress testing. By carefully blending real and synthetic bad scenarios, we can generate a more realistic and severe distribution of potential future losses than history alone provides, leading to more robust portfolio construction.

Table 14: Bootstrap recipe

Block definition	$B_{k,j} = \{r_i, \ldots, r_{i+L-1} \mid z_s = k \ \forall s\}$
Bootstrap CDF	$\hat{F}_R(x) = \frac{1}{N_{\text{blocks}}} \sum_b \mathbf{1}\{\text{ret}_b \le x\}$
Synthetic weight	$w_{\text{syn}} \in (0,1)$ controls tail mass
Final VaR	$\mathrm{VaR}_\alpha = -\hat{F}_R^{-1}(1-\alpha)$

15 Ensemble Risk Forecast - Model-Agnostic Tail

Log-score weights push capital to the best-calibrated model each month - no manual tuning. Regime-weighted update reacts within 63 days to model decay. Final 99

Explanation: Instead of relying on a single VaR model, we combine several (e.g., QRF, REGARCH-EVT, Synthetic Bootstrap) into an ensemble. The ensemble forecast is more robust than any single model.

How it Works: 1. **Scoring:** Each model m is scored based on its recent calibration. A common score is the Kupiec P-value, which measures how well the model's stated VaR confidence level (e.g., 99%) matches the actual historical exceedance rate. A higher P-value means better calibration. 2. **Weighting:** Model weights w_m are calculated using the softmax function on these scores. A higher score leads to a higher weight. The parameter λ controls how concentrated the weights are on the best model. 3. **Regime-Weighted Update:** The scores are calculated not over a simple rolling window, but over a regime-weighted window. Recent performance in the *current* regime is given more importance, allowing the ensemble to quickly adapt if a model starts failing in a new environment.

Application: This creates a dynamic, self-correcting risk management system. The ensemble VaR consistently meets regulatory back-testing requirements (e.g., the "green zone" in the Basel framework) by automatically shifting trust to the currently best-performing model.

Table 15: Ensemble weights

Weight construction	$w_m = \exp(\lambda S_m) \big/ \sum_{m'} \exp(\lambda S_{m'}), \quad S_m = \text{Kupiec p-value}_m$
Ensemble VaR	$\text{VaR}_{\text{ens}} = \sum_{m=1}^{4} w_m \, \text{VaR}_m$
Outcome	Exceedance rate 1.02 % (target 1 %) with lowest bias –0.9 %

16 Strategy-Decay Detection - Alpha Fading

Chow test detects structural breaks in factor loadings - early warning before Sharpe halves. Regime-weighted re-estimation focuses on recent relevant regime - faster convergence. Cool-off prevents over-fitting to noise; rolling Sharpe stays ¿ 1.5 for 5 years.

Explanation: Trading strategies (alphas) often decay over time as markets adapt. The Chow test is a statistical test for a structural break in a regression model—a sudden change in the relationship between variables.

How it Works: 1. The performance of a strategy is modeled as a linear function of various factors and its own past returns. 2. The Chow test is applied at various points in time. It splits the data into two periods and tests whether the regression coefficients are significantly different between the two periods. A significant F-statistic indicates a structural break. 3. **Regime-Weighting:** The test and subsequent model re-estimation are not performed on raw recent data, but on data weighted by its similarity to the current regime. This filters out noise from irrelevant past regimes. 4. **Cool-Off:** After a re-optimization, a minimum period is enforced before the next test to prevent overfitting to recent noise.

Application: This provides an early warning system for strategy decay. Detecting a break prompts re-optimization of the strategy's parameters using only the most relevant data (post-break and regime-matched), helping to restore performance and extend the strategy's life.

Table 16: Chow test

Chow statistic	$F = \dfrac{[\text{RSS}_{\text{pool}} - (\text{RSS}_1 + \text{RSS}_2)]/k}{(\text{RSS}_1 + \text{RSS}_2)/(n_1 + n_2 - 2k)}$
Re-optimisation window	Last 252 days regime-weighted
Cool-off	21-day minimum live period before next test

17 Portfolio Construction - Regime-Weighted Risk-Parity

Risk-parity equalises vol contributions; regime tilt over-weights the strategy that owns that regime. Turn-over cap at 20 Result: Sharpe 2.37 with –6.8

Explanation: Risk Parity is a portfolio construction method that allocates capital so that each asset (or strategy) contributes equally to the total portfolio risk. This is often more robust than capital-weighted allocation (like 60/40).

How it Works: 1. The goal is to solve for weights w_i such that $\mathrm{RC}_i = w_i(\Sigma w)_i$ is equal for all components i. This typically involves an optimization process. 2. **Regime Tilt:** The pure risk-parity solution is then tilted. The weight for a strategy is increased if it has a high Sharpe ratio in the *current* regime z_t and/or has a low market beta ($\beta^{\mathrm{mkt}}_{i,z_t}$), meaning it provides diversification. 3. **Turnover Control:** A constraint is added to limit the total weekly change in weights. This is crucial for controlling transaction costs (TC) and making the strategy practical to implement.

Application: This is the final step that combines all alpha signals (LSTM short, Transformer momentum, etc.) and risk models into a single portfolio. It dynamically balances the portfolio based on which strategies are expected to perform best in the current regime, while rigorously managing risk and costs.

Table 17: Weight recipe

Vol target	$\sigma_{\mathrm{target}} = 15\%$ ann.
Risk-contrib	$\mathrm{RC}_i = w_i\,(\Sigma \mathbf{w})_i$
Regime tilt	$w_i \propto \frac{\mathrm{Sharpe}_{i,z_t}}{\sigma_i} \cdot (1 - \beta^{\mathrm{mkt}}_{i,z_t})$
Turn-over cap	$\lVert \mathbf{w}_t - \mathbf{w}_{t-1} \rVert_1 \le 20\%$ per week

18 Performance Attribution - Where Does the Edge Come From?

Brinson decomposition extended for regime-switching coefficients - quantifies synergy term. Interaction +2.4 Risk drag only –1.2

Explanation: Performance attribution breaks down the total portfolio return into its sources to understand what drove performance.

How it Works: 1. An extended Brinson model attributes return to: - **Alpha:** The return from the individual strategies' stock-picking/skill. - **Allocation:** The return from over/under-weighting certain regimes. - **Interaction:** The return from the synergy between allocation and alpha (e.g., being overweight a regime AND the strategy working well there). 2. A key addition here is quantifying the boost from using **synthetic data** (+1.73. The **Risk Drag** (–1.2

Application: This analysis is crucial for justifying the complex ML system to stakeholders. It proves that the edge comes not from a single "magic" model, but from the sophisticated integration of multiple components.

19 Ethical & Market-Impact Check - Does Our Success Create Systemic Risk?

Crowding shock scenario shows VaR increases +18 % if everyone runs our Regime-2 short - internal limit kicks in. Model card disclosed to regulators - full transparency + stress results. Caps short notional $\le$ 30 % NAV when AggShort ¿ 2σ median - systemic risk mitigated.

Table 18: Attribution table

Alpha strategies	+7.6 %
Regime timing	+2.3 %
Synthetic data	+1.7 %
Risk drag	–1.2 %
Interaction (ensemble synergy)	+2.4 %

Explanation: A successful strategy can become a risk to itself and the market if too many participants copy it ("crowding"). This section analyzes the ethical implications and systemic risk of the strategy.

How it Works: 1. **Crowding Shock Analysis:** A scenario is simulated where the strategy becomes popular. The "Aggressive Short" (AggShort) position in the market is estimated. The model calculates how this crowded trade would amplify losses for everyone (including this portfolio) if it unwound, leading to a "systemic VaR shock." 2. **Internal Limits:** To mitigate this self-fulfilling prophecy, the strategy has a built-in rule: if the overall market's short positioning becomes extreme (e.g., ¿ 2 standard deviations above its median), the strategy will automatically reduce its own short exposure to a maximum of 30% of Net Asset Value (NAV). 3. **Transparency:** A "model card" documenting the strategy's purpose, performance, and limitations is prepared for regulators, promoting transparency and trust.

Application: This is a critical part of responsible quant finance. It ensures the strategy not only performs well but also operates within a framework that considers its external impact and potential to contribute to systemic events.

Table 19: Crowding shock

Systemic VaR shock	$\Delta \mathrm{VaR}_\alpha^{\mathrm{sys}} = \mathrm{VaR}_\alpha \cdot \left(1 + \phi \frac{\mathrm{AggShort}_t}{\text{Market-cap}_t}\right)$
Internal limit	Short exposure capped at 30 % of NAV when AggShort ¿ 2σ median
Transparency	Full model card released to regulators

20 Final 2-Second Quant Summary

"We treat the market as a latent-regime stochastic process, learn its non-linear generators, augment rare-sample space, adapt alphas and risk models per regime, and ensemble everything so that when the next once-in-a-decade storm hits, the book is already positioned for it."