Production-grade Probabilistic Quantitative + ML Trading System for an Illiquid Emerging Market (NEPSE)

Audience: Quantitative researcher with strong math and Python skills. Institutional depth; assume no prior conversation context.

Table of contents

1. Problem Definition

What is being predicted.

Define the *random variable* explicitly. Let P_t be the mid-price at time t. For a fixed horizon h (e.g., 1 day, 5 days), define the log-return:

$$R_{t,h} \equiv \log \frac{P_{t+h}}{P_t}.$$

The system must produce a **probabilistic forecast** of $R_{t,h}$ conditional on the filtration $\mathcal{F}_t$ (all information available at time t).

$$\text{Goal: } \hat{F}_{t,h}(x) \approx \mathbb{P}(R_{t,h} \le x \mid \mathcal{F}_t),$$

or equivalently provide the conditional density $f_{t,h}(x)$ and moments (mean, variance, higher moments or quantiles).

Why price prediction is wrong; why returns are used.

- Price is a nonstationary process with multiplicative scale and trending level shifts. Predicting price level is dominated by drift and scale effects; such predictions are poor for execution. Returns (log-returns) remove multiplicative scale and allow modeling of stationary increments (or detect when stationarity fails).
- A point prediction $\mathbb{E}[R_{t,h}]$ is insufficient; execution decisions require the *distribution* of outcomes (tail risk, skewness, kurtosis) and probability mass around the bid-ask and slippage thresholds.

Mathematical definition of the target.

Full target: conditional distribution $F_{t,h}$ or a parametric family $\mathcal{D}(\theta_{t,h})$ where parameters $\theta_{t,h}$ are predicted from features. Example: $R_{t,h} \mid \mathcal{F}_t \sim t(\mu_{t,h}, \sigma_{t,h}, \nu_{t,h})$ where μ, σ, ν are time-varying.

---

2. Data Preparation & Stationarity

Log returns.

Definition (single-step):

$$r_t = \log P_t - \log P_{t-1} = \log\left(\frac{P_t}{P_{t-1}}\right).$$

Multi-step horizon h:

$$R_{t,h} = \log\left(\frac{P_{t+h}}{P_t}\right) = \sum_{i=1}^{h} r_{t+i}.$$

Why required.

- Additivity across horizons.
- Stabilizes multiplicative effects; aligns with continuous-time returns $d\log P$.
- Enables use of standard time-series tools that assume stationarity (or piecewise stationarity).

Stationarity assumptions and consequences of violating them.

Assume: r_t is weakly stationary conditional on regimes: constant mean and autocovariance structure within a regime. In emerging markets, stationarity often fails globally due to regime shifts (policy, market microstructure events) and illiquidity spikes; consequences:

- Model coefficients become time-varying—necessitating regime modeling or robust online updating.
- Estimators become biased; standard errors incorrect.
- Backtest Sharpe and VaR estimates are unreliable.

Python: compute returns and quick stationarity checks

```python
import numpy as np
import pandas as pd
from statsmodels.tsa.stattools import adfuller

# price series: pandas Series indexed by timestamp
def log_returns(price: pd.Series) -> pd.Series:
    return np.log(price).diff().dropna()

# ADF test for stationarity
def adf_test(series: pd.Series):
    stat, pvalue, _, _, crit_vals, _ = adfuller(series.dropna())
    return dict(stat=stat, pvalue=pvalue, crit=crit_vals)

# Example
# prices = pd.read_csv('prices.csv', index_col=0, parse_dates=True)['mid']
# r = log_returns(prices)
# print(adf_test(r))
```

Notes: do not blindly accept ADF in illiquid data—thin trading produces zero returns and spurious unit roots.

---

3. Market Regime Modeling

Model: Markov Regime-Switching Autoregressive (MS-AR)

Full (econometric) equation for r_t with K regimes and AR(p) dynamics within each regime:

Let $s_t \in \{1, \dots, K\}$ be an unobserved Markov chain with transition matrix Q where $q_{ij} = \mathbb{P}(s_t = j \mid s_{t-1} = i)$. Conditional on regime $s_t = k$:

$$r_t = \mu^{(k)} + \sum_{j=1}^{p} \phi_j^{(k)} r_{t-j} + \sigma^{(k)} \varepsilon_t^{(k)}, \qquad \varepsilon_t^{(k)} \overset{iid}{\sim} D(0,1),$$

where D is a standardized distribution (Gaussian or t). Transition dynamics:

$$\mathbb{P}(s_t = j \mid s_{t-1} = i) = q_{ij}.$$

Explanation of regimes

- *Trend* regime: $\phi_1^{(k)} > 0$, positive autocorrelation; persistent directional moves.
- *Mean-reversion* regime: negative autocorrelation, $\sum \phi_j^{(k)}$ pulling back to mean.
- *Stress* regime: high $\sigma^{(k)}$, fat tails, potentially skewed disturbances.

Why regime-awareness is mandatory in emerging markets

- Nonstationarity is dominated by infrequent but structural regime switches. Liquidity, policy moves, and concentrated ownership create abrupt transitions.

- Trading signals that ignore regimes will be miscalibrated for volatility and autocorrelation, causing large P&L drawdowns when regime flips.

Python implementation using `statsmodels`

```
import numpy as np
import pandas as pd
from statsmodels.tsa.regime_switching.markov_autoregression import
MarkovAutoregression

# r: pandas Series of returns
# Fit MS-AR with K regimes and order p
def fit_ms_ar(r: pd.Series, k_regimes=3, order=1, trend='c'):
    model = MarkovAutoregression(r, k_regimes=k_regimes, order=order,
trend=trend)
    res = model.fit(em_iter=50, search_reps=5, method='nm')
    # res.summary() gives coefficients and smoothed probabilities
    smooth_probs = res.smoothed_marginal_probabilities
    return res, smooth_probs

# Example
# res, probs = fit_ms_ar(r, k_regimes=3, order=1)
# probs.tail()
```

Interpretation: use `res.predict()` and `res.smoothed_marginal_probabilities` to obtain regime probabilities $\mathbb{P}(s_t = k \mid \mathcal{F}_t)$. These probabilities feed downstream features and dynamic weighting.

---

4. Volatility Modeling

Choice: EGARCH (asymmetric, models log-variance) or GJR-GARCH (captures leverage effect). Use heavy-tailed innovations (Student-t) to model fat tails.

EGARCH(1,1) formulation. Let σ_t^2 be conditional variance. Using EGARCH(p,q) with p=q=1:

$$\log \sigma_t^2 = \omega + \beta \log \sigma_{t-1}^2 + \alpha \left(\frac{|\varepsilon_{t-1}|}{\sigma_{t-1}} - E\left[\frac{|\varepsilon_{t-1}|}{\sigma_{t-1}} \right] \right) + \gamma \frac{\varepsilon_{t-1}}{\sigma_{t-1}},$$

where $\varepsilon_t = r_t - \mu_t$ and γ captures asymmetry (leverage).

Why volatility is a forecast, not noise.

- Volatility clusters and is predictable conditional on past innovation magnitudes, liquidity, and regime probabilities; treating it as noise ignores predictable risk and mis-scales position size.

Fat tails and leverage.

- Use Student-t or generalized error distribution for ε_t to capture tails.

- Include asymmetric terms (EGARCH / GJR) to let negative shocks impact future variance more than positive shocks.

Python implementation using `arch`

```
from arch import arch_model

# r: pandas Series of returns
# Fit EGARCH(1,1) with t-distribution
def fit_egarch(r: pd.Series):
    am = arch_model(r*100, mean='Constant', vol='EGARCH', p=1, q=1,
dist='StudentsT')
    res = am.fit(disp='off')
    return res

# Fit GJR-GARCH(1,1)
def fit_gjr(r: pd.Series):
    am = arch_model(r*100, mean='Constant', vol='GARCH', p=1, o=1, q=1,
dist='StudentsT')
    res = am.fit(disp='off')
    return res

# Example
# eg_res = fit_egarch(r)
# print(eg_res.summary())
```

Scale returns (multiply by 100) for numerical stability in arch package. Use the forecasted $\hat{\sigma}_{t+h}$ to standardize residuals and compute probabilistic forecasts.

---

5. Liquidity & Market Impact

Amihud illiquidity measure. For asset i, over a window of d days:

$$\mathrm{ILLIQ}_{i,t} = \frac{1}{d} \sum_{\tau=t-d+1}^{t} \frac{|R_\tau|}{\mathrm{Volume}_\tau},$$

units: return-per-unit-volume. Interpretation: higher values = more price move for given volume (illiquid).

Why ignoring liquidity makes backtests fake.

- Assumed execution at mid-price and unlimited size is unrealistic. In illiquid markets, market impact and crossing spreads dominate small predicted edges. Backtests that ignore liquidity overstate returns and understate drawdowns.

Liquidity as a penalty term.

In expected return calculation, subtract impact/cost:

$$\text{AdjustedEdge}_t = \mathbb{E}[R_{t,h} \mid \mathcal{F}_t] - c_0 - c_1 \cdot \text{ILLIQ}_t - c_2 \cdot \text{Spread}_t,$$

where c_1, c_2 are calibration parameters (estimated from microstructure or backtests). Alternatively, incorporate liquidity into position cap:

$$\text{TradeSize}_{\max} = \min\left(\text{VaR-based cap},\ \kappa \cdot \frac{\text{ADV}}{\text{market impact factor}}\right).$$

---

6. Decomposition: Linear Structure vs Nonlinear Residuals

Model decomposition. Decompose returns as:

$$r_t = L_t + \varepsilon_t, \qquad L_t = \mathbf{x}_t^\top \beta_t$$

where L_t is the structured (linear) component, e.g., AR terms, regime-dependent means, and observable exogenous linear predictors; and ε_t is the residual. The residual is what the ML layer should model.

Mathematical definition of residuals.

Given estimated linear predictor $\hat{L}_t$, residual:

$$\hat{\varepsilon}_t = r_t - \hat{L}_t.$$

Standardize residuals by forecasted volatility $\hat{\sigma}_t$:

$$\tilde{\varepsilon}_t = \frac{\hat{\varepsilon}_t}{\hat{\sigma}_t}.$$

Why ML should only model residuals.

- Linear structure captures predictable, interpretable dynamics with low variance of estimation. If ML models raw returns, it risks re-learning linear relations and overfitting noise.
- ML focuses capacity on nonlinear interactions in standardized residuals where extra explanatory power might exist.

---

7. Machine Learning Layer

Model choice: LightGBM or XGBoost for gradient-boosted trees as primary; neural nets only as an extension with strict regularization and temporal CV.

Feature set (example):

- Lagged returns: $r_{t-1}, \ldots, r_{t-p}$
- Lagged absolute returns: $|r_{t-1}|, \ldots$

- Volatility forecasts: $\hat{\sigma}_t, \hat{\sigma}_{t+1}$
- Regime probabilities: $\mathbb{P}(s_t = k)$ for each k
- Liquidity measures: Amihud, ADV, spread
- Orderbook features (if available): top-of-book imbalance, depth ratios
- Calendar features: day-of-week, month
- Cross-sectional features (when moving to multiple stocks): market cap, float, concentration

Training target.

Train on standardized residuals or directly on tail-probabilities. Preferred target:

$$Y_t = \mathbb{E}\left[\frac{R_{t,h}}{\hat{\sigma}_{t,h}} \mid \mathcal{F}_t\right] \approx \mathbb{E}[\tilde{\varepsilon}_{t,h} \mid \mathcal{F}_t].$$

For probabilistic outputs, train models to predict multiple quantiles or distribution parameters (e.g., gradient boosting for quantile regression or use NGBoost for probabilistic boosting).

Avoiding leakage.

- Use strictly causal features (no future data).
- Train/validate via expanding window forward chaining (no random shuffle).
- When features use rolling statistics, ensure only past window is used.
- Use purging to remove overlapping labels if horizons overlap.

Python example (LightGBM) — training on residuals

```
import lightgbm as lgb
from sklearn.model_selection import TimeSeriesSplit

# X: DataFrame of features indexed by time. y: target = standardized
residuals for horizon h

def train_lgb(X, y, n_splits=5):
    tss = TimeSeriesSplit(n_splits=n_splits)
    models = []
    for train_idx, val_idx in tss.split(X):
        X_tr, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_tr, y_val = y.iloc[train_idx], y.iloc[val_idx]
        dtrain = lgb.Dataset(X_tr, y_tr)
        dval = lgb.Dataset(X_val, y_val)
        params = {
            'objective': 'regression',
            'metric': 'l2',
            'learning_rate': 0.01,
            'num_leaves': 31,
            'min_data_in_leaf': 100,
            'lambda_l1': 1.0,
            'lambda_l2': 1.0,
            'verbose': -1
        }
        bst = lgb.train(params, dtrain, valid_sets=[dval],
early_stopping_rounds=100, num_boost_round=2000)
        models.append(bst)
    return models

# Predict: ensemble average of models
```

Notes: use strong regularization (min_data_in_leaf, lambda L1/L2) and low learning rate. Use feature importance diagnostics and SHAP for interpretability.

---

8. Forecast Combination

Combine linear forecast and ML residual forecast.

Let:\
- $L_{t,h}$: linear forecast for horizon h (from MS-AR predicted mean) - $\hat{\varepsilon}^{ML}_{t,h}$: ML forecast for standardized residual - $\hat{\sigma}_{t,h}$: volatility forecast

Construct conditional mean forecast:

$$\hat{\mu}_{t,h} = L_{t,h} + \hat{\sigma}_{t,h} \cdot \hat{\varepsilon}^{ML}_{t,h}.$$

For full predictive distribution, combine parametric residual distribution (e.g., Student-t with fitted $\hat{\nu}$) with estimated mean and scale:

$$R_{t,h} \mid \mathcal{F}_t \sim t\left(\mu = \hat{\mu}_{t,h},\ \sigma = \hat{\sigma}_{t,h},\ \nu = \hat{\nu}_{t,h}\right).$$

Weighting logic.

- Fixed weighting: choose scalar $\alpha \in [0,1]$ and set final mean $= (1-\alpha)L + \alpha(L + \sigma\varepsilon^{ML})$ — equivalently trust ML fraction α.
- Dynamic weighting: let α_t be inverse of expected conditional MSE estimated from CV; compute rolling out-of-sample RMSEs for linear and ML components and set weights proportional to inverse RMSE.

Formally, let $\mathrm{MSE}^{lin}_t, \mathrm{MSE}^{ml}_t$ be forward-looking CV errors; set

$$\alpha_t = \frac{\mathrm{MSE}^{lin}_t}{\mathrm{MSE}^{lin}_t + \mathrm{MSE}^{ml}_t}.$$

Final predictive distribution. Use mixture or single parametric with fat tails. If residuals non-Gaussian, use a parametric family for tails or a mixture-of-experts combining regime-specific distributions.

---

9. Signal Extraction (MOST IMPORTANT)

Predictions → Signals (not trades).

Define the **edge** as expected net return per unit of traded risk after costs and impact.

Mathematical edge for a proposed trade of sign $s \in \{+1, -1\}$ and intended notional fraction x of portfolio:

$$\mathrm{Edge}(x) = x \cdot \hat{\mu}_{t,h} - \mathrm{Cost}(x) - \mathrm{Impact}(x) - \lambda_{\mathrm{liq}} \cdot f(\mathrm{ILLIQ}_t, x),$$

where $\hat{\mu}_{t,h}$ is forecasted return, Cost(x) includes explicit fees and half-spread, Impact(x) is expected adverse price movement from executing size x, and λ_{liq} penalizes illiquidity.

Why accuracy % is meaningless.

- Classification accuracy ignores calibration and economic significance. A 60% accuracy with tiny edges may be useless; a 40% accuracy with large positive expectancy per trade (asymmetric payoff) could be profitable. Expected value per trade and risk per trade matter.

Threshold logic.

Set a threshold τ on $\mathrm{Edge}/\mathrm{TradeRisk}$ where TradeRisk is e.g., forecast volatility scaled by size.

Formulate z-score of edge:

$$Z_t = \frac{\hat{\mu}_{t,h} - \mathrm{costs}}{\hat{\sigma}_{t,h}}.$$

Trade only if $|Z_t| > \tau$. Choose τ to control expected turnover and ensure positive expected P&L after costs. Example heuristics:

- Calibrate τ via rolling OOS backtest to maximize realized IR under transaction cost budget.
- Use separate thresholds per regime and liquidity bucket.

Mathematical threshold selection (example).

Maximize expected utility under linear utility with risk-aversion parameter γ: choose trade size x solving

$$\max_x \; x\hat{\mu} - x^2 \frac{\hat{\sigma}^2}{2\gamma} - \mathrm{Cost}(x).$$

First-order condition yields threshold on $\hat{\mu}/\hat{\sigma}$. Solve for minimal $\hat{\mu}$ that leads to nonzero x.

Python: turning predictive distribution into signals

```
import numpy as np

# inputs per asset/time
mu_hat = ...        # forecasted mean return (horizon h)
sigma_hat = ...     # forecasted vol
illiq = ...         # Amihud
spread = ...        # half-spread
fee = 0.0005

# cost model approx: linear + quadratic impact
def cost_estimate(x, adv, illiq, spread, fee, k1=0.1, k2=0.3):
    # x: fraction of capital to trade
    # adv: average daily volume (fraction of float)
    market_impact = k1 * (x/adv) + k2 * (x/adv)**2
    explicit = spread + fee
    liq_penalty = illiq * 1e3
    return explicit + market_impact + liq_penalty

# Signal generation
def make_signal(mu_hat, sigma_hat, illiq, spread, adv, fee=0.0005, tau=0.5):
    # z-score
    z = mu_hat / sigma_hat
    # naive desired fraction from Kelly-like
    f_kelly = mu_hat / (sigma_hat**2 + 1e-9)
    # compute costs at candidate f
    cost = cost_estimate(abs(f_kelly), adv, illiq, spread, fee)
    adjusted_edge = mu_hat - cost
    z_adj = adjusted_edge / sigma_hat
    if abs(z_adj) > tau:
        sign = np.sign(z_adj)
        size = np.clip(0.5 * f_kelly, -1.0, 1.0)  # conservative scale
        return dict(signal=sign, size=size, z=z_adj)
    return dict(signal=0, size=0, z=z_adj)

# Example usage in vectorized backtest loop
```

Notes: signals must be stored as time-stamped records, with regime, liquidity bucket, and expected cost attached for post-trade analysis.

---

10. Position Sizing & Risk Control

Volatility targeting and Kelly-style sizing.

- *Volatility targeting:* choose position size such that portfolio ex-ante volatility equals target σ^*:

$$w_t = \frac{\sigma^*}{\hat{\sigma}_t} \cdot \frac{\hat{\mu}_t}{|\hat{\mu}_t|},$$

but volatility targeting alone ignores expected return magnitude. Better: fractional Kelly:

$$f_t = \eta \cdot \frac{\hat{\mu}_t}{\hat{\sigma}_t^2}, \qquad \eta \in (0,1) \text{ for shrinkage}$$

cap and liquidity adjustment:

$$f_t^{adj} = f_t \cdot \frac{1}{1 + \zeta \cdot \mathrm{ILLIQ}_t}, \qquad |f_t^{adj}| \le f_{\max}$$

Drawdown containment logic.

- Stop-loss by portfolio drawdown: reduce leverage stepwise after drawdown thresholds (e.g., reduce by 25% after 5% drawdown, 50% after 10%).
- Volatility / tail risk triggers: if realized vol > k * forecast vol or stress regime probability > 0.6, move to cash or reduce size.
- Hard capital allocation caps by single-asset exposure and daily turnover limits.

Risk overlays. Implement delta limits, stop-losses, sector-level caps, correlation stress tests, and daily max loss per trader/strategy.

---

11. Backtesting & Evaluation

Metrics that matter.

- Information Ratio / Sharpe (with robust standard error).
- Maximum drawdown and duration.
- Turnover and transaction costs.
- Liquidity consumption and market impact realized vs modeled.
- Tail risk metrics: 95% VaR, expected shortfall.
- Profit per trade, win size vs loss size distributions.

Why win-rate is misleading.

Win-rate ignores magnitude — large losses can wipe out many small wins. Expected return per trade and distributional skewness/kurtosis matter more.

Common backtest lies and mitigations.

- *Lookahead / Label leakage:* avoid by strictly causal features and time-series CV. Use purging when labels overlap.
- *Survivorship bias:* use full universe including delisted stocks.
- *Transaction cost underestimation:* model spread, fees, and price impact calibrated to real fills.
- *Ignoring execution latency and queueing:* in illiquid markets, model order matching and partial fills.

Backtest skeleton (walk-forward)

```
# Pseudocode: expanding-window walk-forward
window_train = 252*3  # 3 years
step = 21  # re-train monthly
for start in range(0, n - window_train - test_size, step):
    train_slice = slice(start, start+window_train)
    test_slice = slice(start+window_train, start+window_train+step)
    # fit linear / regime / vol / ml on train
    # produce forecasts for test
    # simulate execution with cost model and sizing
    # record PnL, turnover, drawdown
# aggregate OOS metrics
```

---

12. Profitability Reality Check

What determines profitability.

- True, persistent information content in features that is not already arbitraged away.
- Low friction or predictable liquidity windows to execute without large impact.
- Robust risk management and regime-aware reduction of exposure during structural changes.

Expected statistical characteristics (ranges, not promises).

For an alpha on an illiquid emerging index:

- Annualized expected return *alpha* may be small (1–10% excess depending on capital and costs).
- Expect high turnover and variable realized Sharpe; realistic target IR often 0.3–1.0 for mid-frequency strategies after costs in such markets.

Why most such systems fail.

- Overfitting to noise, failure to model costs and liquidity, structural breaks, insufficient capacity, and organizational execution weaknesses.

---

13. Scalability & Extensions

From index to stocks.

- Move from single-series modeling to panel models. Use cross-sectional ML that borrows strength across assets (multi-task learning) but account for differing liquidity and idiosyncratic microstructure.
- Add hierarchical risk models: factor exposures, sector constraints.

Increasing ML complexity safely.

- Use holdout sets, nested CV, and model monitoring. Increase complexity only if OOS performance improves in a stable way across regimes.
- Regularize heavily and use calibration targets (probabilistic scoring rules like CRPS or negative log-likelihood) rather than raw RMSE.

What breaks first as capital increases.

- Liquidity: realized market impact grows nonlinearly with size.
- Slippage and opportunity cost from inability to deploy full size.
- Strategy crowding leading to alpha decay.

Appendix: Compact formulas and code snippets

1) Target & density

$$R_{t,h} = \sum_{i=1}^{h} r_{t+i}, \quad r_t = \log P_t - \log P_{t-1}.$$

Parametric forecast: $R_{t,h} \mid \mathcal{F}_t \sim t(\mu_{t,h}, \sigma_{t,h}, \nu_{t,h})$.

2) Amihud

$$\text{ILLIQ}_t = \frac{1}{d} \sum_{\tau=t-d+1}^{t} \frac{|R_\tau|}{\text{Volume}_\tau}.$$

3) Kelly fraction (fractional)

$$f_t = \eta \cdot \frac{\mu_t}{\sigma_t^2}, \qquad 0 < \eta < 1.$$

4) Simple cost model

$$\text{Cost}(x) = \text{spread} + c_1 \frac{x}{\text{ADV}} + c_2 \left(\frac{x}{\text{ADV}}\right)^2.$$

Final notes on productionization

- Implement modular architecture: data ingestion layer (resilient to missing ticks), feature store, model training pipelines (airflow or prefect), model registry, real-time scoring service, and execution engine with simulator and real broker connectors.
- Monitoring: model drift detection, PnL attribution, regime-probability monitoring, and automated rollback.
- Governance: documented assumptions, parameter provenance, and independent validation of cost/impact models.

End of document.