# Evaluating GRU powered trading systems: the case of cryptocurrency markets

**Jose Marquez Jaramillo** JMARQU20@JH.EDU
*Miami, FL*

## Abstract

We study whether Gated Recurrent Unit (GRU) forecasts can translate into *investable* improvements for cryptocurrency portfolios under realistic constraints. Using a reproducible, leakage-free walk-forward pipeline (data → preprocessing → GRU training with a direction-aware loss → portfolio formation), we restrict the universe to the Top-5 coins by market capitalization and enforce a 35% single-asset cap with weekly rebalancing. We compare a GRU-driven allocator against (i) a naïve historical-mean allocator and (ii) a capped, market-weighted benchmark, under two objectives: *Maximize Expected Returns* and *Maximize Quadratic Utility*.

Out-of-sample evaluation (90 days following each training window) shows that, at a daily horizon, forecast accuracy is modest but non-trivial. Under Maximize Expected Returns, the GRU portfolio consistently improves upon the naïve baseline on level and risk metrics; however, mean daily outperformance versus the capped benchmark is not statistically significant. Under Quadratic Utility, GRU and naïve allocations are effectively indistinguishable—consistent with a small, concentration-capped universe where the risk penalty dominates modest differences in expected returns. All results are pre-cost.

This early release is limited in time and scope: a short evaluation window dominated by a bearish regime, 50 hyperparameter trials per model, and a Top-5 universe. We outline concrete extensions: more extensive tuning (200+ trials), longer horizons through the present, regime-conditioned analysis (volatility/bull–bear), broader universes (Top-20+), and architectural comparisons (GRU vs. LSTM vs. Transformer) within the same portfolio-centric evaluation framework.

## 1. Introduction

Cryptocurrency markets exhibit extreme volatility, heavy tails, and rapid regime shifts that complicate forecasting and portfolio construction. Deep learning sequence models have become effective tools for extracting predictive structure from non-stationary time series (Lim and Zohren, 2021). Among them, the Gated Recurrent Unit (GRU) provides a compact gated architecture that mitigates vanishing gradients and often matches LSTM performance with fewer parameters (Cho et al., 2014; Chung et al., 2014). In crypto specifically, comparative studies report that gated RNNs—including GRUs—often outperform naïve statistical baselines on out-of-sample error metrics (Bouteska et al., 2024; Seabe et al., 2023; Murray et al., 2023; Kaur et al., 2025; John et al., 2024), complementing early evidence that machine-learning signals can be predictive in liquid coins (Alessandretti et al., 2018).

**Objective.** We test whether GRU-based return forecasts translate into *portfolio* improvements in a concentrated, investable universe. We restrict the investable set to the *top five* cryptocurrencies by market capitalization (reselected at each rebalance), and we compare a GRU-driven allocator against two concrete baselines:

- **Capped market-cap benchmark:** a market-cap-weighted index of the same top-five universe with a 35% single-asset cap to limit concentration.
- **Naïve expected-return estimator:** portfolio construction driven by rolling historical-mean returns (a standard, low-assumption baseline) (Hyndman and Athanasopoulos, 2021).

**Method.** We use a two-stage, walk-forward pipeline: (i) a GRU predicts one-step-ahead asset returns from OHLCV and technical features; (ii) forecast vectors and an empirical covariance feed a long-only optimizer with a 35% per-asset cap to produce portfolio weights. We evaluate economic performance (cumulative return, Sharpe/Sortino, max drawdown, tracking error and information ratio versus the capped benchmark) and forecast diagnostics (RMSE/MAE and directional accuracy). In this early release, we do not debit transaction costs or trade frictions; all results are thus *pre-cost*.

**Contributions.** Relative to broader architecture comparisons, we offer: *(1)* a focused GRU-only study that measures *portfolio* value-add against two strong baselines (capped market index; naïve estimator); *(2)* a concentration-aware benchmark that mirrors investability constraints (top-five with a 35% cap); and *(3)* a clean, walk-forward experimental design that aligns forecasting and portfolio formation to reduce look-ahead and selection bias (Bouteska et al., 2024; Murray et al., 2023; Seabe et al., 2023; John et al., 2024).

## 2. Related Work

### 2.1. GRUs and deep sequence models for time series

The Gated Recurrent Unit (GRU) introduced compact gating and state updates that mitigate vanishing gradients while reducing parameter count relative to LSTMs (Cho et al., 2014). Early empirical comparisons reported that GRUs often match LSTM performance across language and sequence benchmarks with fewer parameters and faster training (Chung et al., 2014). More broadly, deep learning for time series has progressed from classical RNNs to gated RNNs, temporal CNNs, and attention-based models, with consistent guidance on regularization, normalization, and walk-forward validation for non-stationary data (Lim and Zohren, 2021). We leverage these design lessons but focus deliberately on GRUs to isolate whether a lightweight gated architecture can translate predictive structure into *portfolio* value.

### 2.2. Cryptocurrency return predictability and ML baselines

A growing body of evidence suggests that machine-learning features can capture short-horizon structure in major cryptocurrencies, though effect sizes are unstable and regime-dependent. Early work documented that feature-based models—momentum, liquidity, and market-microstructure covariates—can improve directional accuracy relative to naïve baselines on liquid coins (Alessandretti et al., 2018). More recent crypto-specific comparisons report gated RNNs (GRU/LSTM) outperform simple statistical baselines on out-of-sample prediction metrics, especially when models are trained with careful walk-forward protocols (Bouteska et al., 2024; Seabe et al., 2023). In our study, we retain a *naïve expected-return*

comparator (rolling historical mean), consistent with standard forecast-baseline practice in time-series analysis (Hyndman and Athanasopoulos, 2021).

### 2.3. From forecasts to portfolios: economic value of predictions

The crucial step—often underemphasized—is converting forecast improvements into economically meaningful portfolio outcomes. Studies that connect crypto forecasts to trading rules generally find that risk-adjusted gains hinge on disciplined rebalancing, position caps, and robust evaluation (e.g., walk-forward splits and pre- versus post-cost reporting) (Murray et al., 2023; John et al., 2024). Within deep-learning architectures, GRU- and LSTM-based signals have shown potential when paired with conservative position sizing and risk controls, though performance can be sensitive to universe definition and feature leakage (Kaur et al., 2025; Bouteska et al., 2024). Our design follows this line by (i) separating forecasting from allocation, (ii) imposing long-only and single-asset caps, and (iii) benchmarking against a concentration-controlled market-cap index built on the same investable set.

### 2.4. Capped market-cap indices and concentration control

Market-cap weighting concentrates risk in the largest asset(s), especially in small universes such as the top-five cryptocurrencies. Practical index construction therefore employs single-constituent caps and periodic rebalancing to balance representativeness and diversification. Our benchmark mirrors this practice via a 35% per-asset cap and the same selection/rebalance schedule used by the strategy, ensuring that performance comparisons reflect *forecast value* rather than differences in investability or concentration.

### 2.5. Broader deep-learning approaches to portfolio construction

Beyond GRU-based forecasting, several adjacent lines inform how deep learning can drive portfolios:

**Attention and Transformer-style models.** Advances in attention mechanisms and sequence modeling have broadened the toolbox for time-series representation, often improving long-horizon dependency modeling and feature selection (Lim and Zohren, 2021). While our study deliberately focuses on GRUs, the same two-stage design (forecast $\rightarrow$ allocate) extends to attention-based encoders.

**Policy learning and allocation rules.** An orthogonal strand learns *policies* (trading rules) directly rather than forecasting returns first. Empirical results suggest that risk-adjusted outcomes depend more on evaluation discipline—walk-forward splits, position caps, and pre/post-cost reporting—than on any single model family (Murray et al., 2023; John et al., 2024). We adopt these evaluation principles here but stay with a forecast-first pipeline.

**Robust optimization and ensembling.** Studies that connect ML signals to portfolios often stabilize weights via constraints, shrinkage, and simple ensembling; conservative position sizing is a recurring ingredient (Kaur et al., 2025; Bouteska et al., 2024; Seabe et al., 2023). Our long-only, 35% cap and naïve-mean comparator (Hyndman and Athanasopoulos, 2021) serve this stabilizing role and keep the focus on the incremental value of GRU forecasts.

## 3. Experimental Approach

### 3.1. Data

**Provenance and tooling.** All data engineering is performed with our in-house `Kallos` suite developed by the author: `kallos-data`[1] for ingestion, storage, feature computation, and index construction. The pipeline retrieves daily OHLCV and market capitalization from the `CoinGecko API`[2], persists the raw and processed tables in PostgreSQL, and exposes modular "processors" for indicators, signals, and index calculation.

**What is extracted.** For each asset in the investable universe we store: coin metadata (identifier, symbol), and a time series of open, high, low, close, volume, and market capitalization at daily frequency. The canonical fact table is keyed by `(asset_id, date)` and enforces type-stable numeric columns to avoid floating-point drift. Where the source provides splits/renames or chain migrations, we apply the repository's built-in normalization rules to maintain a continuous series.

**Derived computations (features).** The data project computes a library of technical features used by the forecaster in §2.2, including: (i) moving averages (SMA/EMA across short, medium, and long windows), (ii) momentum/oscillators (rate of change, RSI), (iii) trend/volatility pairs (MACD, Bollinger bands), (iv) volume-scaled indicators (volume EMA, money-flow style ratios). Exact parameter grids (e.g., SMA $\in \{10, 20, 50, 100, 200\}$; RSI $\in \{7, 14, 21\}$) are specified in the repository's processor configs and fixed prior to model tuning to avoid leakage.

**Universe definition (Top-5).** At each rebalance boundary we form the investable universe as the Top-5 cryptocurrencies by market capitalization computed from the same daily store. (If a stablecoin enters the Top-5, we follow the rule defined in our index methodology—exclude it or keep it—so that the benchmark and strategies use the identical rule set.) Constituents are selected on the final trading day of month $m$ and become effective for month $m{+}1$.

**Capped market-cap benchmark (implementation-equivalent to our trading constraint).** To ensure a fair comparison against a passive alternative that shares our investability and concentration constraints, we construct a capped market-cap index on the Top-5 universe using the index calculator shipped in `kallos-data` (see `crypto_index_calculator_docs.md`[3] for full details).

1. **Initial weights (cap-weighted).** On the rebalance date, compute
$$w_i^{(0)} = \frac{\text{mcap}_i}{\sum_{j \in \mathcal{U}} \text{mcap}_j}, \quad i \in \mathcal{U} \text{ (Top-5)}.$$
2. **35% cap via iterative redistribution.** Apply a single-name cap $c = 0.35$. Set $w_i \leftarrow \min\{w_i^{(0)}, c\}$, sum the excess $E = \sum_i (w_i^{(0)} - w_i)_+$, and redistribute $E$ proportionally

1. https://github.com/josemarquezjaramillo/kallos-data
2. https://www.coingecko.com/en/api
3. https://github.com/josemarquezjaramillo/kallos-data/blob/main/kallos/documentation/data_processing/crypto_index_calculator_docs.md

over the uncapped set $\mathcal{U}_{\text{free}} = \{k : w_k^{(0)} < c\}$:

$$w_k \leftarrow w_k + E \cdot \frac{w_k}{\sum_{\ell \in \mathcal{U}_{\text{free}}} w_\ell}.$$

Repeat until $w_i \leq c$ for all $i$, then renormalize so $\sum_i w_i = 1$.

3. **Between rebalances (buy-and-hold drift).** On day $t$ between rebalances, weights evolve with relative returns and are renormalized:

$$\tilde{w}_{i,t} = w_{i,t-1} \cdot \frac{P_{i,t}}{P_{i,t-1}}, \qquad w_{i,t} = \frac{\tilde{w}_{i,t}}{\sum_j \tilde{w}_{j,t}}.$$

4. **Index level.** With base value $\text{Index}_0 = 1000$, the daily return is

$$r_t = \sum_{i \in \mathcal{U}} w_{i,t-1} \left( \frac{P_{i,t}}{P_{i,t-1}} - 1 \right), \tag{1}$$

$$\text{Index}_t = \text{Index}_{t-1} \, (1 + r_t). \tag{2}$$

5. **Data quality and maintenance.** If an asset lacks a price on day $t$, its weight is temporarily redistributed proportionally across available names for that day; delisted names are removed at the next rebalance with proportional reallocation.

**Reproducibility notes.** All steps above are parameterized in the author's repository (e.g., `top_n_constituents=5`, `max_constituent_weight=0.35`, rebalance cadence). We freeze these parameters for the entire study and log the exact configuration files and database snapshot timestamps used for the experiments.

### 3.2. Modeling: preprocessing, tuning, training, and evaluation (via `kallos_models`)

**Tooling stack and package layout.** All modeling is implemented in the author's package `kallos_models`[4], with Darts for time-series containers and GRU wrappers (Herzen et al., 2021), Optuna for Hyper-Parameter Optimization (HPO) (Akiba et al., 2019), and PyTorch backend (Paszke et al., 2019).

**Preprocessing (feature transformations and leakage control).** We construct supervised samples from daily OHLCV-derived features (§3.1), applying group-specific transformations and enforcing leakage-safe fitting (scalers fit on training folds only; frozen on validation/test). As summarized in Table 1, we group transforms by feature family.

**Dimensionality reduction (price block).** Because price-derived features (OHLC, returns, price/MA ratios) are highly collinear, we apply PCA to the *standardized* price-feature block and retain the top $k$=2 components per asset. PCA is fit only on each step's training portion and applied unchanged to that step's validation and 90-day OOS periods (**?**Pedregosa et al., 2011).

4. https://github.com/josemarquezjaramillo/kallos_models

Table 1: Feature normalization strategy used in this study. Transformations are fit on training folds only and applied to validation/test with frozen parameters (no look-ahead).

| Feature Group | Example Features | Normalization Method | Key Advantages / Rationale |
|---|---|---|---|
| Price-based (OHLC) | Open, High, Low, Close; price/MA ratios | Log returns or price-to-MA ratios; per-asset standardization; **PCA on price block** ($k$=2, train-only fit) | Stabilizes variance; removes multicollinearity in the price block via PCA, improving conditioning for the learner (Siami-Namini et al., 2018). |
| Volume / Market-Cap | Trading volume; Market cap | $\log(1+x)$ transform + per-asset standardization | Compresses right-skewed, heavy-tailed distributions; improves numerical stability for learning. |
| Bounded oscillators | RSI, Stochastic, MFI | Rolling min–max to $[0, 1]$ (windowed; past-only) | Preserves natural bounds and comparability across assets/horizons; avoids leakage via trailing-window min–max (implementation detail of this project). |
| Unbounded "difference" indicators | EMA diff, MACD diff, Price–BB diff | Robust scaling via quantiles (per asset) | Reduces outlier leverage while keeping data centered—useful in crypto's heavy-tailed regimes (Alessandretti et al., 2018). |
| Rate of Change (ROC) | Price ROC; Volatility ratio | Signed log transform + per-asset standardization | Dampens extremes yet preserves directionality for sign-aware objectives; improves stability. |

**Target and data splits.** The target is one-day-ahead arithmetic return, $y_{t+1} = P_{t+1}/P_t - 1$. We adopt **walk-forward optimization (rolling origin)** with an **expanding static start**: train to a step end, validate on the tail, then deploy the tuned model for the subsequent **90-day OOS** block. This matches standard practice (Hyndman and Athanasopoulos, 2021) and the operational use in §3.3.

**Model: GRU.** We use a Darts GRU forecaster. Tunables include hidden size, layers, dropout, input chunk $\ell_{\text{in}}$, $\ell_{\text{out}}$=1, learning rate, weight decay, gradient clip, and optimizer. We disable teacher forcing for single-step forecasting, apply early stopping, and clip gradients.

**Custom loss function (and related literature).** We train with a *direction-selective MSE* that up-weights errors when the predicted and true returns have opposite signs. Let $d_t = \mathbf{1}\{\hat{y}_t\, y_t > 0\}$ (1 if the signs agree; 0 otherwise, with ties treated as wrong) and let $\lambda > 1$ be the direction penalty. The per-sample weight is $w_t = d_t + \lambda(1 - d_t)$, and the loss is

$$\mathcal{L} = \text{mean}_t\big[w_t\,(\hat{y}_t - y_t)^2\big] \tag{3}$$

Implemented as a `torch.nn.Module`, it integrates with Darts' PyTorch models. This aligns with recent work on *custom/asymmetric losses* for trading-oriented forecasting—e.g., MADL (Michańków et al., 2022), GMADL (Michańków et al., 2024), Dessain's down-

side/asymmetric objectives (Dessain, 2023), and DI-MSE (Yin, 2023) — which overweight misdirection, downside, or tails to better reflect economic use.

**Hyperparameter optimization (walk-forward).** We tune the GRU with Optuna (Akiba et al., 2019) using a static start and expanding-window protocol (walk-forward). Each trial is fit on $K$ folds (validation at each fold's tail), then deployed for the subsequent 90-day OOS block.

**Search space (per trial).** We optimize

$$\begin{aligned}\Theta = \{&\lambda,\\ &\texttt{hidden\_dim},\\ &\texttt{n\_rnn\_layers},\\ &\texttt{dropout},\\ &\texttt{batch\_size},\\ &\texttt{input\_chunk\_length},\\ &\texttt{learning\_rate}\}\end{aligned}$$

with:

- $\lambda \in \{1, 2, 3, 4, 5\}$ (directional penalty in the custom loss);
- $\texttt{hidden\_dim} \in \{32, 64, 128, 256\}$;
- $\texttt{n\_rnn\_layers} \in \{1, 2, 3, 4\}$;
- $\texttt{dropout} \in [0.0, 0.7]$;
- $\texttt{batch\_size} \in \{32, 64, 128\}$;
- $\texttt{input\_chunk\_length} \in [30, 70]$ (days);
- $\texttt{learning\_rate} \in [10^{-5}, 10^{-2}]$ (log-uniform).

The forecast horizon is fixed to $\texttt{output\_chunk\_length} = 1$.

**Objective across folds (compact form).** Let $\mathrm{RMSE}_k(\Theta)$ and $\mathrm{DA}_k(\Theta)$ be fold-$k$ metrics. We optimize two aggregated objectives:

$$J_1(\Theta) = K^{-1} \sum_{k=1}^{K} \mathrm{RMSE}_k(\Theta) \quad \text{(minimize)}, \tag{4}$$

$$J_2(\Theta) = K^{-1} \sum_{k=1}^{K} \mathrm{DA}_k(\Theta) \qquad \text{(maximize)}. \tag{5}$$

We use a multi-objective sampler (NSGA-II) and persist each study for auditability (Akiba et al., 2019).

**Trials and accounting.** We run **50 trials per model** and tune **102 models** total ($\approx$ 5,100 trials). Completed trials are not rerun when resuming studies.

**Early stopping.** Each fold trains with early stopping: *max epochs* = 1000, *patience* = 50, *min* $\Delta = 5 \times 10^{-4}$ on validation loss. Validation series/covariates are passed explicitly so the monitored quantity is correct.

**Deployment.** For each asset/step we select $\Theta^\star$ from the study, retrain on the full in-sample window, and generate daily predictions for the next **90 days** OOS (used in §3.3).

**Final training and evaluation.** For each asset/step we train a production GRU on the entire in-sample window up to the step end date and persist the artifacts needed for the 90-day OOS deployment:

1. **Data load.** Pull all rows up to the step end date from the feature store (same feature groups as in tuning).
2. **Fit transformer.** Create a *ColumnTransformer* and fit it on the full in-sample feature set; transform features to a normalized dataframe (train-only fit at this stage, no OOS rows).
3. **TimeSeries conversion.** Build Darts `TimeSeries` for the target and past covariates (same frequency as the dataframe index).
4. **Trainer config.** Use a PyTorch-Lightning trainer with **max_epochs** = 1000, **early stopping** (*patience* = 50, *min* $\Delta = 5 \times 10^{-4}$), and **monitor**=`train_loss`.
5. **Instantiate model.** Create the GRU with the selected hyperparameters $\Theta^\star$ and pass the trainer kwargs.
6. **Fit.** Train on the full in-sample `TimeSeries` with past covariates; early stopping monitors `train_loss`.
7. **Persist artifacts.** Save the trained model to {`study_name`}`.pt` and the fitted scaler to {`study_name`}`_scaler.pkl`. These artifacts are loaded verbatim in the inference/evaluation stage.
8. **OOS predictions.** The resulting model is then used to generate daily predictions for the subsequent **90 calendar days** (handled in our evaluation pipeline) which feed the weekly-rebalanced portfolio in §3.3.

*Notes.* (i) Unlike tuning—where scalers are refit per fold—production training fits the transformer once on the full in-sample set to match final deployment. (ii) Early stopping monitors `train_loss` because no validation slice is supplied in this stage; this accelerates convergence while preserving the "train-on-all in-sample" objective. (iii) Using the `study_name` prefix guarantees the correct model–scaler pairing at inference time.

### 3.3. Portfolio construction and backtesting (via `kallos_portfolios`)

**Framework overview (third-party libraries).** Portfolio experiments are implemented in the author's package `kallos_portfolios`[5], which follows a layered design: data/storage

5. https://github.com/josemarquezjaramillo/kallos_portfolios

→ predictors/models → optimisers → simulators → evaluation/analysis. We rely on three external libraries in this stage: *PyPortfolioOpt* for mean–variance optimization and linear constraints (Martin, 2021; **?**), *vectorbt* for high-performance, NumPy/Numba-accelerated backtesting (Polakow, 2025), and *QuantStats* for performance analytics and reporting (Aroussi, 2021).

**Strategies compared.** We evaluate three strategies under identical calendars and constraints:

1. **GRU-driven portfolio.** Expected returns $\hat{\boldsymbol{\mu}}_t$ come from §3.2 (7-day-ahead forecasts), aligned to the next weekly rebalance by the GRU simulator.
2. **Naïve estimator portfolio.** Rolling historical-mean returns provide $\hat{\boldsymbol{\mu}}_t$.
3. **Capped market-cap benchmark.** The passive comparator from §3.1 with a 35% single-asset cap (same universe and cap as the optimised strategies).

**Optimization details (PyPortfolioOpt).** At each weekly rebalance we optimize portfolio weights over the current Top-5 universe subject to long-only and concentration limits,

$$\mathbf{1}^\top \mathbf{w}_t = 1, \qquad 0 \le w_{i,t} \le 0.35 \;\; \forall i,$$

using `pypfopt.EfficientFrontier` with linear constraints (`weight_bounds`, `add_constraint`). We evaluate two objectives:

- **Max expected return:** $\max_{\mathbf{w}} \; \hat{\boldsymbol{\mu}}_t^\top \mathbf{w}$.
- **Quadratic utility:** $\max_{\mathbf{w}} \; U(\mathbf{w};\delta) = \hat{\boldsymbol{\mu}}_t^\top \mathbf{w} - \frac{\delta}{2}\,\mathbf{w}^\top \Sigma_t \mathbf{w}, \quad \delta > 0$, where $\delta$ is the risk-aversion parameter. We use the standard of $\delta$=1.

Here $\hat{\boldsymbol{\mu}}_t$ are expected returns from §3.2 (or the naïve estimator), and $\Sigma_t$ is the sample covariance computed on the investable set using a rolling **365-day** look-back. We use the SLSQP solver and clean weights after optimization (PyPortfolioOpt API).

**Backtesting engine (vectorbt).** Execution uses a weekly schedule (every 7 days); weights apply at each rebalance and drift between rebalances. The simulation is performed with `vectorbt`, which operates on pandas/NumPy arrays and is accelerated by Numba—enabling fast, vectorized portfolio modeling and analysis (Polakow, 2025).

**Performance analytics (QuantStats).** For each strategy–objective pair we generate tear sheets and a consolidated comparison report using `QuantStats` (`stats`, `plots`, `reports`) to compute standard metrics (total/annualized return, volatility, Sharpe/Sortino, max drawdown, rolling stats) (Aroussi, 2021).

**Evaluation protocol.** *Calendars.* Rebalances occur every 7 days (e.g., Mondays). The investable universe each day is the *current month's* constituents of the capped market-cap benchmark (§3.1); the simulator updates the universe on month-change. *Forecast alignment.* GRU predictions are mapped to the next rebalance date; if an asset lacks a prediction on a given date, documented fallbacks are applied before optimization. *Consistency across objectives.* We run `min_volatility`, `max_sharpe`, and `max_expected_returns` under identical constraints (long-only, $\sum w = 1$, $w_i \le 0.35$) using the same universe and rebalance schedule.

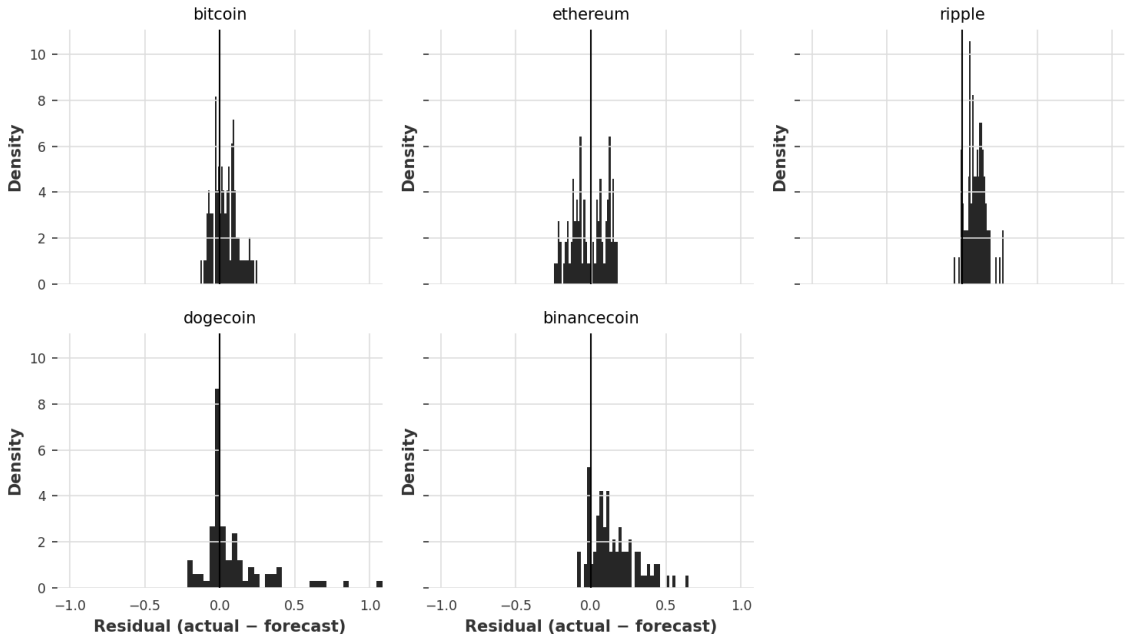


Figure 1: Residual distributions (actual − forecast) per coin over the 90-day OOS window. Top row: BTC, ETH, XRP; bottom row: DOGE, BNB.

## 4. Results

**At a glance.** (1) *Forecasts:* GRU achieves modest directional accuracy at daily frequency; error shapes differ by coin (Fig. 1). (2) *MER portfolios:* GRU narrows the gap to the capped benchmark and clearly outperforms the Naïve allocator on level/risk metrics, but daily mean outperformance is not statistically significant (Tables 2–6). (3) *QU portfolios:* GRU and Naïve behave nearly identically under quadratic utility; both track the benchmark with slightly milder tails (Tables 3–9).

### 4.1. Forecast evaluation (five case studies)

We evaluate five representative models—BTC, ETH, XRP, DOGE, BNB—trained through **2023-12-31** and deployed for the next **90 calendar days** (Q1 2024). Metrics include RMSE, MAE, directional accuracy (DA), rank correlations, and trading diagnostics; see §3.2 for the walk-forward protocol and §3.3 for how forecasts feed portfolios.

**Residual diagnostics.** We define residuals as $r_t = y_t - \hat{y}_t$ (positive = under-forecast). Common bins (Freedman–Diaconis) and symmetric axes enable shape comparison across assets.

**Takeaway.** BTC shows the tightest spread; XRP skews slightly positive (mild under-forecasting); DOGE/BNB exhibit heavier right tails, consistent with jumpier dynamics. These asymmetries matter once forecasts are mapped into constrained portfolios.

### 4.2. Portfolio results

We compare three strategies under two objectives: GRU-driven, Naïve (historical mean), and the capped market-cap benchmark. Rebalancing is **weekly**, the universe is the **Top-5** with a **35%** per-asset cap (§3.3). We report cumulative returns (Fig. 2), core performance (CAGR, Vol, Sharpe, Max DD, Calmar), additional stats (Total return, Win rate, VaR/cVaR), and daily-return hypothesis tests.

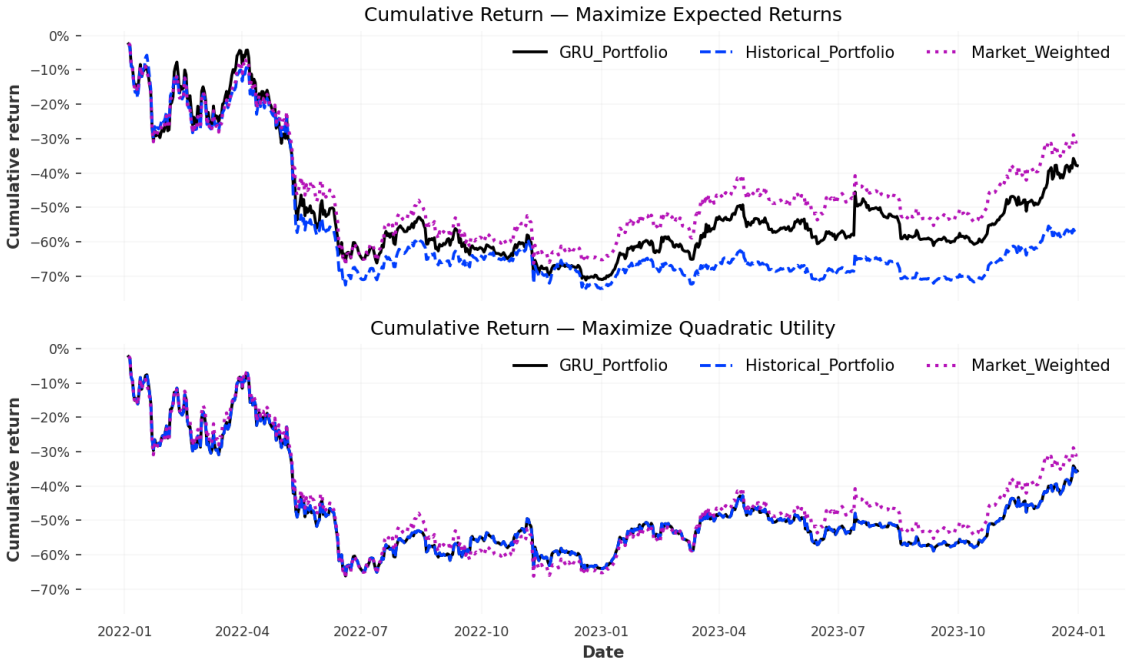


Figure 2: Cumulative returns by objective. **Top:** Maximize Expected Returns (MER): GRU, Naïve, Benchmark. **Bottom:** Maximize Quadratic Utility (QU): same three strategies. Axes shared across rows.

#### 4.2.1. Maximize Expected Returns (MER)

**Core performance.** Under MER GRU improves materially over the Naïve allocator on level metrics (Total return and CAGR) and is *less negative* on Sharpe/Sortino (Tables 2–4), but daily mean outperformance vs. the benchmark is not significant (Table 5); variance is higher than the benchmark (Table 6), consistent with deeper drawdowns.

Table 2: Core performance under *MER*. $r_f$=0.

| Strategy | CAGR | Ann. Vol | Sharpe | Max DD | Calmar |
|---|---|---|---|---|---|
| GRU (forecast-driven) | −15.35% | 53.75% | −0.29 | −71.15% | −0.22 |
| Naïve (historical mean) | −25.63% | 50.74% | −0.51 | −73.66% | −0.35 |
| Capped benchmark | −12.01% | 48.78% | −0.25 | −66.82% | −0.18 |

#### 4.2.2. Maximize Quadratic Utility (QU)

**Core performance.** Under QU, the risk penalty dominates modest differences in expected returns in a small, capped universe: GRU and Naïve converge (identical metrics; Table 3) and neither shows significant daily mean outperformance versus the benchmark (Table 8); tails are slightly milder than the benchmark (Table 7).

Table 3: Core performance under *QU*. $r_f$=0.

| Strategy | CAGR | Ann. Vol | Sharpe | Max DD | Calmar |
|---|---|---|---|---|---|
| GRU (forecast-driven) | −14.26% | 46.98% | −0.30 | −66.19% | −0.22 |
| Naïve (historical mean) | −14.26% | 46.98% | −0.30 | −66.19% | −0.22 |
| Capped benchmark | −12.01% | 48.78% | −0.25 | −66.82% | −0.18 |

## 5. Concluding Remarks

**Summary.** We studied whether GRU forecasts can translate into portfolio value on a constrained, investable crypto universe. Under *Maximize Expected Returns* (MER), the GRU allocator consistently improved upon a naïve historical-mean strategy on level and risk metrics, yet did not deliver statistically significant daily mean outperformance versus the capped market benchmark. Under *Maximize Quadratic Utility* (QU), the GRU and naïve allocations converged—consistent with a small, concentration-capped Top-5 universe where the risk penalty dominates modest differences in expected returns. Forecast diagnostics suggest usable but modest signal at daily frequency; residual shapes vary meaningfully by coin and likely interact with portfolio constraints.

**Limitations.** This is an early, pre-cost study with deliberate scope limits: (i) a short out-of-sample window dominated by a bearish regime; (ii) a concentrated Top-5 universe with a 35% cap and weekly rebalancing; (iii) GRU tuning at *50 trials per model*. These choices reduce computation and look-ahead risk but may understate the model's ceiling.

**Priority extensions.**

1. **Scale the search and horizon.** Increase hyperparameter trials (e.g., $\geq 200$/model) and extend the walk-forward to the present, so the evaluation spans multiple volatility and market-direction regimes.

2. **Regime awareness.** Re-estimate and test by volatility/market regimes (e.g., low/high realized vol; bull/bear). A simple approach is to stratify walk-forward blocks; a richer one is a regime-switching or mixture-of-experts GRU with regime-conditioned loss.

3. **Broaden the universe.** Move from Top-5 to at least Top-20 by market cap to reduce concentration, increase cross-sectional degrees of freedom, and give the optimizer room to express views subject to caps.

4. **Utility-aligned training.** Explore losses closer to the portfolio objective (e.g., asymmetric/DA-weighted MSE already used here; quantile or utility-weighted losses), and calibrate risk-aversion via cross-validated utility rather than fixed parameters.

5. **Robust risk and constraints.** Test alternative covariance estimators and regularization (e.g., Ledoit–Wolf, shrinkage targets), turnover penalties, tighter weight caps, and rebalancing schedules.

6. **Costs and frictions.** Incorporate transaction costs, slippage, and borrow/venue effects into backtests; report both pre- and post-cost results.

7. **Architectural comparison.** Train *GRU, LSTM, and Transformer* variants under identical features, splits, loss, and HPO budget; enforce parameter/compute parity; and compare both forecast and portfolio metrics with formal tests. For Transformers, use finance-friendly encoders (causal/sliding-window or temporal-fusion style), regularize (dropout/weight decay), and tune sequence length/attention window.

**Closing.** Within the constraints studied, GRU forecasts improved upon a naïve baseline but did not consistently beat a concentration-aware market benchmark. The roadmap above—more trials, longer horizon, regime conditioning, a broader universe, and utility-aligned training—should clarify whether GRUs can deliver durable, investable edge in crypto portfolios.

Table 4: Additional statistics under *MER*. VaR/cVaR are daily.

| **Strategy** | **Total Ret** | **Win Rate** | **VaR (95%)** | **cVaR (95%)** |
|---|---|---|---|---|
| GRU (forecast-driven) | −38.54% | 50.41% | −5.52% | −7.61% |
| Naïve (historical mean) | −57.89% | 50.82% | −5.07% | −7.12% |
| Capped benchmark | −30.89% | 49.86% | −4.87% | −7.61% |

Table 5: Mean-difference tests under *MER*. Mean Diff = A−B in bp/day (two-sided).

| **Comparison** | **N** | **Mean Diff** | $t$ | $p$ | **95% CI (bp/day)** |
|---|---|---|---|---|---|
| GRU vs. Naïve | 736 | 5.6 | 1.45 | 0.148 | [–2.0, 13.3] |
| GRU vs. Benchmark | 728 | –0.5 | –0.12 | 0.907 | [–8.3, 7.4] |
| Naïve vs. Benchmark | 728 | –6.2 | –1.98 | 0.049 | [–12.4, –0.0] |

Table 6: Diagnostics for *MER*: variance equality (F), distribution (KS), first-order stochastic dominance (FSD).

| **Comparison** | $F$ **(p)** | **KS (p)** | **FSD** |
|---|---|---|---|
| GRU vs. Naïve | 1.12 (0.118) | 0.020 (0.998) | None |
| GRU vs. Benchmark | 1.23 (0.006) | 0.038 (0.655) | None |
| Naïve vs. Benchmark | 1.09 (0.228) | 0.038 (0.655) | None |

Table 7: Additional statistics under *QU*. VaR/cVaR are daily.

| **Strategy** | **Total Ret** | **Win Rate** | **VaR (95%)** | **cVaR (95%)** |
|---|---|---|---|---|
| GRU (forecast-driven) | −36.21% | 51.09% | −4.65% | −7.46% |
| Naïve (historical mean) | −36.21% | 51.09% | −4.65% | −7.46% |
| Capped benchmark | −30.89% | 49.86% | −4.87% | −7.61% |

Table 8: Mean-difference tests under *QU*. Mean Diff = A−B in bp/day (two-sided).

| **Comparison** | **N** | **Mean Diff** | $t$ | $p$ | **95% CI (bp/day)** |
|---|---|---|---|---|---|
| GRU vs. Naïve | 736 | 0.0 | – | – | [0.0, 0.0] |
| GRU vs. Benchmark | 728 | –1.3 | -0.51 | 0.610 | [–6.2, 3.6] |
| Naïve vs. Benchmark | 728 | –1.3 | -0.51 | 0.610 | [–6.2, 3.6] |

Table 9: Diagnostics for *QU*: variance equality (F), distribution (KS), first-order stochastic dominance (FSD).

| **Comparison** | $F$ **(p)** | **KS (p)** | **FSD** |
|---|---|---|---|
| GRU vs. Naïve | 1.00 (1.000) | 0.000 (1.000) | Identical |
| GRU vs. Benchmark | 1.07 (0.386) | 0.022 (0.995) | None |
| Naïve vs. Benchmark | 1.07 (0.386) | 0.022 (0.995) | None |

## References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

Laura Alessandretti, Abeer ElBahrawy, Luca Maria Aiello, and Andrea Baronchelli. Anticipating cryptocurrency prices using machine learning. *Complexity*, 2018(1), January 2018. ISSN 1099-0526. doi: 10.1155/2018/8983590.

Ran Aroussi. Quantstats: Portfolio analytics for quants. https://github.com/ranaroussi/quantstats, 2021. Accessed: 2025-08-10.

Ahmed Bouteska, Mohammad Zoynul Abedin, Petr Hajek, and Kunpeng Yuan. Cryptocurrency price forecasting – a comparative analysis of ensemble learning and deep learning methods. *International Review of Financial Analysis*, 92:103055, March 2024. ISSN 1057-5219. doi: 10.1016/j.irfa.2023.103055.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

Jean Dessain. Improving the prediction of asset returns with machine learning by using a custom loss function. *Advances in Artificial Intelligence and Machine Learning*, 03(04): 1640–1653, 2023. ISSN 2582-9793. doi: 10.54364/aaiml.2023.1193.

Julien Herzen, Francesco Lässig, Samuele Giuliano Piazzetta, Thomas Neuer, Léo Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan Kościsz, Dennis Bader, Frédérick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and Gaël Grosch. Darts: User-friendly modern machine learning for time series. 2021. doi: 10.48550/ARXIV.2110.03224.

R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2021. ISBN 9780987507136.

David L. John, Sebastian Binnewies, and Bela Stantic. Cryptocurrency price prediction algorithms: A survey and future directions. *Forecasting*, 6(3):637–671, August 2024. ISSN 2571-9394. doi: 10.3390/forecast6030034.

Ramneet Kaur, Mudita Uppal, Deepali Gupta, Sapna Juneja, Syed Yasser Arafat, Junaid Rashid, Jungeun Kim, and Roobaea Alroobaea. Development of a cryptocurrency price prediction model: leveraging gru and lstm for bitcoin, litecoin and ethereum. *PeerJ Computer Science*, 11:e2675, March 2025. ISSN 2376-5992. doi: 10.7717/peerj-cs.2675.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, February 2021. ISSN 1471-2962. doi: 10.1098/rsta.2020.0209.

Robert Martin. Pyportfolioopt: portfolio optimization in python. *Journal of Open Source Software*, 6(61):3066, May 2021. ISSN 2475-9066. doi: 10.21105/joss.03066.

Jakub Michańków, Paweł Sakowski, and Robert Ślepaczuk. Lstm in algorithmic investment strategies on btc and samp;p500 index. *Sensors*, 22(3):917, January 2022. ISSN 1424-8220. doi: 10.3390/s22030917.

Jakub Michańków, Paweł Sakowski, and Robert Ślepaczuk. Generalized mean absolute directional loss as a solution to overfitting and high transaction costs in machine learning models used in high-frequency algorithmic investment strategies, 2024.

Kate Murray, Andrea Rossi, Diego Carraro, and Andrea Visentin. On forecasting cryptocurrency prices: A comparison of machine learning, deep learning, and ensembles. *Forecasting*, 5(1):196–209, January 2023. ISSN 2571-9394. doi: 10.3390/forecast5010010.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, November 2011. ISSN 1532-4435.

Oleg Polakow. vectorbt: Find your trading edge, using the fastest backtesting engine, 2025. URL https://github.com/polakowo/vectorbt.

Phumudzo Lloyd Seabe, Claude Rodrigue Bambe Moutsinga, and Edson Pindza. Forecasting cryptocurrency prices using lstm, gru, and bi-directional lstm: A deep learning approach. *Fractal and Fractional*, 7(2):203, February 2023. ISSN 2504-3110. doi: 10.3390/fractalfract7020203.

Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401. IEEE, December 2018. doi: 10.1109/icmla.2018.00227.

Haojie Yin. Enhancing directional accuracy in stock closing price value prediction using a direction-integrated mse loss function. In *Proceedings of the 1st International Conference on Data Analysis and Machine Learning*, pages 119–126. SCITEPRESS - Science and Technology Publications, 2023. doi: 10.5220/0012810200003885.