ML-based Alpha Ranking and Portfolio Optimization

François Goybet
francois.goybet@epfl.ch

Gavriil Kharkhordine
gavriil.kharkhordine@gmail.com

Haocong Li
haocong.li@epfl.ch

***Abstract*—We study equity portfolio construction through cross-sectional learning-to-rank. Instead of predicting returns directly, we train models to recover the relative ordering of stocks within each month, using firm-level characteristics from the Open Source Asset Pricing dataset. Our final XGBoost specification delivers statistically significant long–short spreads, monotonic decile behavior, and strong out-of-sample portfolio performance. We also compare these ranking-based portfolios with a reinforcement learning baseline that incorporates transaction costs and liquidity constraints.**

I. Introduction

Portfolio construction is often more naturally framed as a ranking problem than as a return prediction problem. In practice, investors care less about the exact return of each stock than about identifying the best assets at the top of the cross-section.

This paper applies learning-to-rank methods to monthly equity selection. Using firm-level characteristics from the Open Source Asset Pricing dataset, we train boosted ranking models to order stocks by expected future performance and then convert these rankings into portfolios. We evaluate the models both with ranking metrics such as NDCG and Hit@K, and with economic measures such as long–short spreads, Sharpe ratio, and cumulative return.

Our main finding is that ranking-aware models produce economically meaningful signals that remain effective after portfolio construction. The resulting top-ranked portfolios exhibit strong out-of-sample performance, supporting the use of learning-to-rank as a practical framework for equity selection.

II. Related Work

A. Learning-to-Rank

Learning-to-rank (LTR) methods optimize the *ordering* of items rather than the accuracy of individual predictions. Three paradigms are standard in the literature (Liu 2009): pointwise, pairwise, and listwise.

a) Pointwise methods: cast ranking as regression or classification on each item independently, then sort by the predicted score. Because the loss ignores cross-item structure, the model is implicitly asked to recover absolute return levels, a setting in which financial data is notoriously noisy and signal-to-noise ratios are low. Pointwise objectives therefore tend to over-fit idiosyncratic noise rather than the relative ordering that portfolio construction actually requires.

b) Pairwise methods: reformulate ranking as binary classification over item pairs: for each pair (i, j) with $y_i > y_j$, the model is penalized when $s_i \leq s_j$. RankNet (C. Burges et al. 2005) introduced the canonical pairwise loss based on a logistic transformation of score differences,

$$\mathcal{L}_{\text{pair}}(i, j) = \log(1 + \exp(-\sigma(s_i - s_j))), \tag{1}$$

where s_i, s_j are the model scores and σ controls the sharpness of the sigmoid. The intuition is that if every pair is ordered correctly, the global ranking follows by transitivity. In practice, optimizing all $O(n^2)$ pairs uniformly is both expensive and statistically wasteful, since most pairs are easy.

c) Listwise methods: extend this idea to entire lists, defining a loss over the full ordering at each query (in our setting, each cross-section at date t). ListNet (Cao et al. 2007) models the probability of each item being ranked first via a softmax over scores and minimizes the cross-entropy against the ground-truth distribution. Empirical results across information retrieval benchmarks show that listwise objectives outperform pairwise objectives on long lists, since the latter weights all pairs equally and ignores absolute position in the ranking.

A central limitation of pure pairwise and listwise losses is that they treat all errors symmetrically, whereas ranking metrics such as NDCG concentrate weight at the top of the list. LambdaRank (C. J. C. Burges, Ragno, and Le 2006) addresses this by reweighting pairwise gradients by the change in the target metric induced by swapping the pair:

$$\lambda_{ij} = \frac{-\sigma}{1 + \exp(\sigma(s_i - s_j))} \cdot |\Delta\text{NDCG}_{ij}|. \tag{2}$$

LambdaMART (C. J. C. Burges 2010) integrates this gradient into a gradient-boosted regression tree framework and remains a strong baseline; it is the conceptual basis for XGBoost's `rank:pairwise` and `rank:ndcg` objectives (T. Chen and Guestrin 2016), both of which we evaluate.

d) Evaluation metrics.: We adopt two complementary metrics. *Normalized Discounted Cumulative Gain at K* (Järvelin and Kekäläinen 2002) discounts gains by log-position so that errors near the top of the list dominate:

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{y_i}{\log_2(i+1)}, \quad \text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}, \tag{3}$$

where IDCG@K is the DCG of the ideal ordering and the normalization makes scores comparable across queries of different sizes. *Hit@K* (analogous to top-K accuracy) reports the fraction of true top-K items recovered in the predicted top-K and offers a more intuitive read for non-technical readers. Both metrics align with the economic reality that a long-short portfolio derives its return from the extremes of the ranking, not the middle (Lin, Su, and Zhu 2026).

B. Position of This Work

We extend this literature by coupling LTR objectives with systematic portfolio construction. Our central hypothesis is that, given firm-level fundamental and market features from the Open Source Asset Pricing dataset (A. Y. Chen and Zimmermann 2022), a gradient-boosted ranker trained with a rank-aware loss can identify cross-sectional return ordering accurately enough at the extremes to support a profitable long strategy. We evaluate this end-to-end: from ranking quality (NDCG@30, Hit@30) through to portfolio outcomes (Sharpe ratio, cumulative return, maximum drawdown), allowing us to test whether improvements in ranking metrics actually translate into economic value.

III. Universe and Data

A. Investment Universe

We use a dataset based on Zimmerman (2022) ((A. Y. Chen and Zimmermann 2022)), which is commonly used in the asset pricing literature. The paper introduces a large set of firm-level characteristics that are used to study and predict stock returns using machine learning methods.

Our investment universe consists of U.S. equities, and we restrict the sample to the period starting in 1990. In order to ensure sufficient liquidity and more realistic trading conditions, we further filter the dataset by excluding stocks with a market capitalization below $10 million. This filtering step helps reduce issues related to illiquidity and potential difficulties in executing trades (e.g., filling orders).

To facilitate data access and feature construction, we rely on the Python API *openassetpricing*, which implements the framework proposed by Zimmerman (2022). This tool automatically downloads the required data sources (e.g., from WRDS) and constructs the corresponding firm-level features. The resulting dataset is restricted to U.S. equities.

B. Data Construction

The data is organized in a cross-sectional format: at each date, we observe a set of firms described by a common feature vector. These features include firm characteristics such as financial ratios, past returns, and other market-based indicators.

Our dataset is built with the `openassetpricing` framework (A. Y. Chen and Zimmermann 2022), which provides a large collection of firm-level characteristics widely used in the asset pricing literature. These variables cover, among others, accruals, profitability, investment, momentum, volatility, liquidity, and valuation signals. The framework also provides detailed documentation for each feature, including its original source, year of publication, and precise definition.

We use these characteristics as the raw input to our models and apply a preprocessing pipeline before training. In addition, we augment the dataset with three meta-features obtained from Ridge regressions trained on the training set to predict 1-month, 3-month, and 6-month forward returns. The out-of-sample predictions of these models are appended as additional columns. The final feature set contains 207 columns.

a) Preprocessing pipeline.: Raw firm-month observations are passed through a stateful pipeline before reaching the ranker. Metadata columns (`permno`, `yyyymm`, `sector`, and forward returns) are separated from the feature matrix so that transformations are applied only to the explanatory variables. All cross-sectional operations are performed within each month independently, and every statistic used for preprocessing is fit on the training set and then reused unchanged on validation and test sets to avoid look-ahead bias.

The pipeline supports the following steps, applied in order:

- **Cross-sectional winsorisation.** We clip each feature within a given month to the $[q,\ 1-q]$ quantile range, with $q = 0.01$. This limits the influence of extreme outliers while preserving the relative structure of the data.
- **Cross-sectional rank normalisation.** We replace each feature value by its within-month rank percentile in $[0, 1]$. This removes scale differences and makes the representation more robust to heavy tails and extreme values.
- **Standardisation.** We experiment with global z-score scaling, global min-max scaling, sector-wise z-score scaling, and sector-wise min-max scaling. In practice, we found this not to be a useful. Cross-sector standardisation is a natural extension we plan to explore further.
- **Median imputation.** Missing values are imputed using the training-set median of each feature. This is preferred over zero-imputation because zero can be a meaningful value for many financial ratios.
- **Ridge meta-features.** As described above, three Ridge predictions are appended to the feature set. These additional signals often rank among the most useful features and provide a strong linear baseline that the gradient-boosted ranker can refine.
- **Centroid distance** (optional): we also consider the L2 distance from each stock's feature vector to the monthly cross-sectional centroid. This feature is intended to capture how atypical a firm is relative to the rest of the universe.
- **Low-variance feature removal.** Features with near-zero variance on the training set are removed, as they contribute little signal and can introduce numerical noise.

We also experimented with an autoencoder-based representation learning step, but in practice this was only a variation of a multilayer perceptron and did not bring meaningful gains. A tabular transformer is a more promising direction for future work, but we have not implemented it yet.

In the configurations reported in Section VII, we enable winsorisation, sector standardisation, median imputation, and

the Ridge meta-features. PCA, the autoencoder, and the centroid feature are disabled.

IV. Models

A. Problem Formulation

We frame portfolio construction as a cross-sectional ranking task. Let $t \in \{1, \dots, T\}$ index rebalance dates (months) and let $\mathcal{S}_t$ denote the set of stocks in the investable universe at date t. For each stock $i \in \mathcal{S}_t$ we observe a feature vector $x_{i,t} \in \mathbb{R}^d$ and a forward return $y_{i,t}^{(h)}$ over horizon $h \in \{1, 3, 6\}$ months. The model learns a scoring function $f_\theta : \mathbb{R}^d \to \mathbb{R}$ such that, within each month t, the induced ordering of $\{f_\theta(x_{i,t})\}_{i \in \mathcal{S}_t}$ approximates the true ordering of $\{y_{i,t}^{(h)}\}_{i \in \mathcal{S}_t}$. Crucially, the loss is computed *within* each monthly group rather than across the pooled dataset: only relative ordering of contemporaneous stocks carries economic meaning, since absolute return levels are dominated by time-varying market factors.

B. Base Rankers

We evaluate two gradient-boosted tree backends, both of which natively support group-wise ranking objectives:

a) XGBoost: (T. Chen and Guestrin 2016) with the `rank:pairwise` and `rank:ndcg` objectives. The former implements a RankNet-style logistic pair loss; the latter is a LambdaRank-style variant that reweights pair gradients by $|\Delta\text{NDCG}|$, concentrating learning effort on errors at the top of the ranking.

b) LightGBM: with the `lambdarank` objective and configurable truncation level K (we use $K = 10$), which restricts gradient computation to the top-K portion of each list.

Both backends require integer relevance grades rather than continuous returns, so a label-encoding step is necessary.

C. Label Encoding

We compared three encoding schemes, all applied within each monthly group so that the per-month relevance distribution is identical across periods:

- **Argsort:** assign each stock its within-month rank in $\{0, 1, \dots, |\mathcal{S}_t| - 1\}$, where $|\mathcal{S}_t| - 1$ corresponds to the highest realized return.
- **Decile / Quintile:** bucket stocks into 10 (resp. 5) equal-frequency groups by realized return, assigning grades $\{0, \dots, 9\}$ or $\{0, \dots, 4\}$.
- **Top-k only:** a recent proposal (Lin, Su, and Zhu 2026) suggests labelling only the top assets with their rank and assigning grade 0 to all others, on the hypothesis that the model should focus exclusively on identifying winners.

Empirically, argsort labelling produced the best downstream portfolio metrics in our experiments. Quintile and decile encodings degraded both NDCG@30 and Sharpe, likely because collapsing many stocks into the same grade discards the fine-grained information that the gradient-boosted ranker can otherwise exploit. The top-k-only scheme also underperformed argsort in our setup; we conjecture that zeroing out the bottom and middle of the cross-section deprives the long-short construction stage of the short-side signal.

D. Multi-Horizon Modelling

Forward returns at different horizons (1, 3, and 6 months) carry different signal-to-noise profiles and may favour different feature subsets. We therefore train one independent ranker per horizon, sharing only the backend and base hyperparameters. We then evaluated two strategies for combining the resulting score vectors:

- **Mean score:** weighted average of raw model scores across horizons.
- **Mean rank (Borda count):** convert each horizon's scores to within-month ranks and average those ranks. More robust to scale differences across horizons.

Counterintuitively, neither aggregation outperformed the single-horizon ranker matched to the rebalance frequency of interest. Using the 1-month model alone for monthly rebalancing yielded the highest Sharpe in our backtests. We attribute this to two factors: (i) the longer-horizon labels are more strongly contaminated by intervening events that are irrelevant to a 1-month decision, and (ii) the mean-score aggregation can be dominated by the noisier long-horizon ranker even when its individual ranking quality is weaker. We therefore retain only the horizon-matched model in our reported configurations, while keeping the multi-horizon code path available for longer rebalance frequencies in future work.

E. Ensembling

We additionally implemented a randomised ensemble in which n XGBoost rankers are each trained on (i) a random subset of features, (ii) a random contiguous sub-interval of training months, and (iii) randomly sampled `max_depth` and `learning_rate`. Final scores are the mean of the base-ranker outputs. Despite this diversity, the ensemble did not consistently beat a single well-tuned XGBoost ranker on either NDCG@30 or Sharpe in our current experiments. We suspect that the gains from variance reduction are offset by the loss in signal when individual base rankers see only a fraction of the features; tuning the sampling ranges more aggressively, or moving to per-month rather than per-window subsampling, are natural next steps.

V. Portfolio Construction

A. From Ranking to Portfolio

Our model does not aim to directly predict future returns, but instead learns to produce a ranking of assets. Predicting returns is known to be extremely challenging. For this reason, we do not rely on the predicted values themselves, but rather on the relative ordering of assets. This ranking allows us to identify which stocks are expected to outperform others the next month.

In traditional portfolio construction, one typically estimates expected returns together with a covariance matrix of asset returns, and then solves an optimization problem. However, such approaches require accurately estimating the covariance

structure, which is difficult in practice. Instead, we adopt simpler and more robust strategies based directly on the predicted rankings.

Our approach is also consistent with the evaluation metric used during training. We rely on the Normalized Discounted Cumulative Gain (NDCG), which places more weight on the top of the ranking. As a result, misranking highly ranked assets is penalized more heavily than errors at the bottom of the ranking. This property aligns well with portfolio construction, where the top-ranked assets typically drive most of the performance.

Finally, we focus exclusively on long-only strategies. In practice, short selling can be difficult to implement for many investors due to constraints imposed by brokers, such as limited availability of securities to borrow or additional costs. For this reason, and given our emphasis on ranking the best-performing assets, we restrict our analysis to long positions.

In the next subsection, we present the different portfolio construction strategies derived from these rankings.

B. Equal-Weighted Portfolio

We consider a simple portfolio construction strategy based on equal weights. At each rebalancing date, we select the top (N) assets according to the predicted ranking and assign each of them an equal weight of (1/N) in the portfolio.

Despite its simplicity, this approach is widely used in practice and can deliver strong performance, especially when the ranking model is effective at identifying the best-performing assets. It also avoids additional estimation errors that may arise from more complex weighting schemes.

In this project, we evaluate several configurations of this strategy, in particular portfolios constructed using the top 1 and top 10 ranked assets.

C. Market Capitalization Weighted Portfolio

Another simple portfolio construction strategy consists in selecting the top (N) assets according to the predicted ranking and assigning weights proportional to their market capitalization. In this setting, larger firms receive higher weights within the portfolio.

We choose to rely on market capitalization as a weighting scheme because it naturally emphasizes more liquid and established companies, which are generally easier to trade and involve lower transaction costs.

In this project, we evaluate several configurations of this strategy, using (N = 1) (which is equivalent to the equal-weighted case with a single asset), as well as (N = 10, 20, 50 and 100).

D. Reinforcement Learning for Portfolio Optimization

While discrete, hand-crafted portfolio sorting rules (e.g., decile or quantile portfolios) provide robust baseline strategies, they often struggle to dynamically balance alpha extraction against real-world execution frictions. To overcome this limitation, we formulate portfolio construction as a continuous, end-to-end optimization problem solved via Deep Reinforcement Learning (RL) (Jiang, Xu, and Liang 2017). Our RL pipeline ingests cross-sectional signals, explicitly models market microstructure frictions (slippage and liquidity constraints), and continuously outputs an optimal weight vector.

1) Upstream Alpha Integration via Walk-Forward Ensembling: To prevent look-ahead bias and simulate a realistic production environment, the upstream predictive signal (an XGBoost-derived rank (T. Chen and Guestrin 2016)) is generated using a walk-forward expanding window methodology (Lopez de Prado 2018). Specifically, the model is trained iteratively (e.g., training on 1990–1993, validating on 1994, and predicting out-of-sample for 1995), sequentially expanding the training set year-by-year. This out-of-sample prediction (`xgboost_rank_oos`) is merged with traditional state features to form a comprehensive predictive state vector for the RL agent.

2) State Representation and Feature Engineering: At each monthly timestep t, the RL agent observes a state matrix encompassing a variable universe of N_t assets. The raw feature set comprises:

- **Alpha Signal:** The out-of-sample walk-forward XGBoost rank.
- **Liquidity & Cost Proxies:** Log dollar volume (`DolVol`), quoted bid-ask spread, and stock-level realized volatility.
- **Macro & Style Factors:** Market-wide volatility (`VolMkt`), macro trend factors, market capitalization (`Size`), Book-to-Market (`BM`), and 12-month momentum (`Mom12m`).

To ensure distributional stability across time, all predictive state features are strictly cross-sectionally z-scored per month. Crucially, raw financial metrics required for market mechanics (e.g., raw dollar volume for impact cost, unscaled bid-ask spread) bypass the normalization step and are passed directly to the simulation engine.

3) Permutation-Invariant Architecture and Action Space: Financial markets exhibit a dynamically changing universe of investable assets due to IPOs, bankruptcies, and liquidity thresholding. Standard Multi-Layer Perceptrons (MLPs) require fixed-size inputs and depend on specific positional ordering. To resolve this, our policy network employs a "Set Transformer Lite" architecture (Lee et al. 2019; Zaheer et al. 2017) (*Shared-Weight Extractor*). The network applies a shared-weight MLP independently to each stock's feature vector, extracting a dense embedding h_i. A global market state is then constructed by concatenating the mean and max aggregations of all active h_i, alongside the current cash weight.

The Actor head outputs raw continuous logits $a_i \in \mathbb{R}$ for up to 2,200 potential equity slots plus one cash slot. A dynamic action mask $-\infty$ is applied to inactive or delisted assets prior to a Softmax activation (Jiang, Xu, and Liang 2017), mathematically guaranteeing that the final portfolio weights w_i strictly satisfy $w_i \geq 0$ and $\sum w_i = 1$.

4) Market Simulation Constraints and Reward Setup: Our custom Gymnasium environment rigorously simulates

portfolio drag to prevent the RL agent from learning spurious high-turnover strategies. The step logic enforces:

1) **Transaction Costs:** Formulated as a combination of a linear half-spread and a non-linear square-root market impact law (Almgren et al. 2005).
2) **Orphan Resolution (Delisting):** Stocks that disappear from the CRSP dataset are automatically swept to cash. Genuine performance-based delistings are heavily penalized (incorporating actual CRSP delisting returns or assuming a 100% capital loss), training the agent to avoid distress risk.
3) **Data Augmentation via Stock Dropout:** During training, we apply a 20% random dropout rate to the valid universe at the start of each episode. This acts as a regularizer (Srivastava et al. 2014), preventing the agent from overfitting to a specific subset of dominant mega-cap stocks and forcing it to learn generalized ranking dynamics.

The agent optimizes for the *Active Return* reward, defined as the net portfolio return (post-execution costs) minus the S&P 500 benchmark return.

In the subsequent results section, we contrast the performance of this RL formulation against several static, hand-crafted ranking baselines (e.g., market-cap weighted Top/Bottom deciles). We demonstrate how the RL agent learns a non-trivial weighting scheme that dynamically adapts to liquidity regimes, navigating the trade-off between maximizing raw alpha and surviving execution friction.

E. Other Strategies

At this stage, our portfolio construction methods do not rely on any information related to industry or sector classifications. As a result, we do not apply sector-neutral constraints or adjustments, which could otherwise help reduce unintended concentration risks and improve diversification.

Incorporating sector information to build sector-neutral portfolios or control for industry exposures is a well-established approach in asset management. However, this requires additional data and modeling choices that are beyond the scope of this project.

We leave this as a potential extension for future work.

VI. Experiments

A. Experimental Pipeline

This section describes the end-to-end experimental pipeline used in this study, from raw data extraction to portfolio evaluation.

We start by retrieving the dataset from the *openassetpricing* framework, applying an initial filter to exclude stocks with a market capitalization below \$10 million. This step ensures a minimum level of liquidity and removes illiquid securities that could distort empirical results.

The processed data is then passed through a dedicated `DataManager` and `FeaturePipeline`. This module is responsible for selecting the investment universe based on the top (N) stocks by market capitalization and applying a sequence of feature engineering and preprocessing steps, including cross-sectional transformations, centroid-based feature construction, removal of low-variance features, missing value imputation, and feature scaling. When available, sector information is used during scaling. In addition, ridge regression models are fitted on past returns and incorporated into the feature set as additional predictive signals.

Once the dataset is fully processed, a `RankingAnalyzer` is used to compute and summarize the ranking performance statistics, which are detailed in the following subsection.

The portfolio construction is then handled by a dedicated `PortfolioConstructor`. For each rebalancing period (monthly), the model generates predictions and constructs portfolio weights according to the selected strategy. These predictions are then used to form the portfolio positions for the subsequent period.

Finally, a `PortfolioAnalyzer` evaluates the resulting strategies by computing performance metrics and risk statistics, allowing for a systematic comparison of the different portfolio construction approaches.

B. Ranking Analyzer

For each model, we evaluate the quality of the out-of-sample ranking produced at each rebalancing period. To do so, we rely on standard ranking-based metrics, namely NDCG@30 and Hit@30. These metrics assess how well the model identifies the most relevant assets, with a particular focus on the top of the ranking.

In addition to these ranking metrics, we also evaluate the economic significance of the predictions using a decile-based analysis. For each out-of-sample month, assets are sorted according to the predicted ranking and assigned to deciles. A well-performing model should produce monotonically increasing average returns across these deciles.

We further focus on the extreme portfolios by constructing a long-short strategy based on the top and bottom deciles. Specifically, we compute the return difference between the top decile and the bottom decile and test the null hypothesis:

$$H_0 : \mathbb{E}[R_{\text{top decile}} - R_{\text{bottom decile}}] = 0.$$

To account for time-series dependence in returns, we use a Newey–West adjusted t-test rather than a standard t-test, which would incorrectly assume independence over time. This provides robust inference for autocorrelated return series.

The resulting NDCG@30, Hit@30, as well as the t-statistic and p-value from this long-short test, are reported in the experimental results section. In addition, we also report the spread, defined as the difference in returns between the top and bottom deciles:

$$\text{Spread} = \mathbb{E}[R_{\text{top decile}}] - \mathbb{E}[R_{\text{bottom decile}}].$$

C. Portfolio Analyzer

For each portfolio construction strategy, we evaluate performance in an out-of-sample setting. The strategies are backtested over time, and we compute the resulting PnL (profit

and loss) series to assess their economic performance under realistic trading conditions.

Transaction costs are taken into account using a cost model expressed in basis points (bps), where 1 basis point corresponds to 0.01% of the traded amount. This allows us to incorporate trading frictions and obtain more realistic net returns.

To compare strategies, we report a standard set of performance metrics computed on the out-of-sample PnL. These include the annualized Sharpe ratio, annualized return (in percentage), maximum drawdown, as well as the average winning trade return and average losing trade return.

VII. Results

A. Results and Final Model Selection

Table I summarizes the out-of-sample performance of the different ranking configurations. Overall, the results show that the learning-to-rank framework is able to generate economically meaningful signals, with several specifications producing statistically significant long–short spreads.

Although configuration **C8** achieves the highest spread of 1.478% and the strongest statistical significance with a **t-statistic of** 2.688 and a **p-value of** 0.0072, we do not retain it as our final model for portfolio construction.

Instead, we select **C7**, based on `XGBoost` with the `rank:ndcg` objective, 100 boosting rounds, and a **top-300 market-cap universe**. This model achieves a strong **NDCG@30** of 0.5396, a **Hit@30** of 0.1469, and a statistically significant spread of 1.242% with a **p-value below** 5%. Importantly, it also provides the best ranking quality among all tested specifications.

The decision not to retain C8 is primarily motivated by the choice of investment universe. Since C8 is trained and evaluated on the **top 500 stocks by market capitalization**, it naturally includes a larger proportion of smaller-cap names compared to the top-300 universe. While this broader universe can mechanically increase the achievable spread, it may also introduce additional volatility, lower liquidity, and less stable ranking behavior. In particular, part of the excess spread may come from exposure to smaller-cap stocks rather than purely better ranking performance.

By contrast, the top-300 universe used in C7 provides a more conservative and robust setting, focused on larger and more liquid firms. This makes the resulting strategy easier to interpret and more realistic from an implementation perspective, especially in the context of portfolio construction.

In addition, the **Hit@30** of C7 remains close to the results reported in the 2026 reference paper, which further supports its selection as the final model.

For these reasons, we use configuration **C7** as the final ranking model to build the long–short portfolios presented in the next section.

B. Decile Analysis and Portfolio Performance

Figure 1 and Figure 2 summarize the two main diagnostics of the strategy. The first plot shows the average realized return by predicted decile, while the second compares the cumulative performance of the different long-only portfolios against the benchmark.

Overall, the results are encouraging. The cumulative PnL increases steadily over time despite a few temporary downturns, which suggests that the signal remains economically meaningful over the full sample rather than being concentrated in a short subperiod. The strategy also appears to recover after drawdowns, which is a positive sign for robustness.

More importantly, the decile curve is close to monotonic in the upper part of the ranking. The best predicted deciles systematically deliver higher average returns than the lower ones, which is exactly the behavior we want from a ranking model. This is especially relevant for the long-only setting, where we care most about identifying the strongest assets at the top of the ranking rather than only separating winners from losers.

In the cumulative performance plot, the portfolios built from the top-ranked stocks clearly outperform the benchmark over most of the sample. Among the tested variants, the more concentrated portfolios tend to generate the strongest final performance, although they also come with greater volatility. This trade-off is consistent with the intuition that stronger selection improves expected return but may increase exposure to estimation noise.

These figures support the choice of the final model: the ranking signal is not only statistically significant, but it also translates into a coherent portfolio ordering and strong out-of-sample cumulative performance.

References

Almgren, Robert et al. (2005). "Direct estimation of equity market impact". In: *Risk* 18.7, pp. 58–62.

Burges, Chris et al. (2005). "Learning to Rank Using Gradient Descent". In: *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. ACM, pp. 89–96. DOI: 10.1145/1102351.1102363.

Burges, Christopher J. C. (2010). *From RankNet to LambdaRank to LambdaMART: An Overview*. Tech. rep. MSR-TR-2010-82. Microsoft Research.

Burges, Christopher J. C., Robert Ragno, and Quoc V. Le (2006). "Learning to Rank with Nonsmooth Cost Functions". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 19. MIT Press, pp. 193–200.

Cao, Zhe et al. (2007). "Learning to Rank: From Pairwise Approach to Listwise Approach". In: *Proceedings of the 24th International Conference on Machine Learning (ICML)*. ACM, pp. 129–136. DOI: 10.1145/1273496.1273513.

Chen, Andrew Y. and Tom Zimmermann (2022). "Open Source Cross-Sectional Asset Pricing". In: *Critical Finance Review* 27.2, pp. 207–264.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 785–794. DOI: 10.1145/2939672.2939785.

TABLE I
RANKING PERFORMANCE ACROSS MODEL CONFIGURATIONS (XGBOOST)

Model Configuration					Model Results				
Config ID	Model	Loss Function	Rounds	Top N Market Cap	NDCG@30	Hit@30	Spread (%)	t-stat	p-value
C1	XGBoost	rank:pairwise	10	300	0.517	0.1323	0.6321	2.211	0.02701
C2	XGBoost	rank:pairwise	10	500	0.5142	0.137	1.013	2.486	0.0129
C3	XGBoost	rank:pairwise	100	300	0.5332	0.1526	1.005	2.266	0.02346
C4	XGBoost	rank:pairwise	100	500	0.5128	0.1505	1.016	1.78	0.075
C5	XGBoost	rank:ndcg	10	300	0.5343	0.1391	0.6982	0.961	0.3366
C6	XGBoost	rank:ndcg	10	500	0.5258	0.1411	1.105	2.045	0.04087
C7	XGBoost	rank:ndcg	100	300	0.5396	0.1469	1.242	2.346	0.04087
C8	XGBoost	rank:ndcg	100	500	0.5224	0.15	1.478	2.688	0.007189

TABLE II
PORTFOLIO PERFORMANCE ACROSS MODEL CONFIGURATIONS

Config ID	Strategy Name	Sharpe Ratio (ann)	Return (% ann)	Max Drawdown (%)	Avg Win (%)	Avg Loss (%)
C7	top_1	0.4047	8.699	-69.27	10.37	-6.615
C7	top_10_equal_weights	1.04	24.9	-29.95	7.268	-4.528
C7	top_10_market_cap	1.12	30.84	-40.03	7.653	-5.37
C7	top_10_market_cap	1.12	30.84	-40.03	7.653	-5.37
C7	top_20_market_cap	1.181	30.72	-33.85	7.26	-5.37
C7	top_50_market_cap	1.106	25.19	-29.76	6.48	-4.73
C7	top_100_market_cap	1.040	21.26	-27.68	5.56	-4.49
RL	RL_strat	0.85	13.6	-19.9	4.23	-3.86

Järvelin, Kalervo and Jaana Kekäläinen (2002). "Cumulated Gain-Based Evaluation of IR Techniques". In: *ACM Transactions on Information Systems* 20.4, pp. 422–446. DOI: 10.1145/582415.582418.

Jiang, Zhengyao, Dixing Xu, and Jinjun Liang (2017). "A deep reinforcement learning framework for the financial portfolio management problem". In: *arXiv preprint arXiv:1706.10059*.

Lee, Juho et al. (2019). "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International conference on machine learning*. PMLR, pp. 3744–3753.

Lin, Yan, Yihong Su, and Zhaobo Zhu (2026). "Empirical Asset Pricing via Learning-to-Rank". In: *SSRN Working Paper*. DOI: 10.2139/ssrn.6348379. URL: https://ssrn.com/abstract=6348379.

Liu, Tie-Yan (2009). "Learning to Rank for Information Retrieval". In: *Foundations and Trends in Information Retrieval* 3.3, pp. 225–331. DOI: 10.1561/1500000016.

Lopez de Prado, Marcos (2018). *Advances in financial machine learning*. John Wiley & Sons.

Schulman, John et al. (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Zaheer, Manzil et al. (2017). "Deep sets". In: *Advances in neural information processing systems*. Vol. 30.

APPENDIX: EXTENDED REINFORCEMENT LEARNING PIPELINE CONFIGURATION

This appendix provides granular details on the architectural parameters, simulation mechanics, and Proximal Policy Optimization (PPO) (Schulman et al. 2017) training schedule used to generate our RL portfolio policy. The pipeline is designed around a fully vectorized NumPy backend, processing ∼420 months of data across 2,200 slots with zero Python loop overhead.

A.1 Network Architecture

The policy network relies on a custom Shared-Weight Extractor designed for continuous, permutation-invariant action spaces (Zaheer et al. 2017):

- **Stock Encoder:** A 2-layer shared MLP with dimensions $[64, 64]$ and ReLU activations. This encoder transforms the k-dimensional feature vector of each stock i into a 64-dimensional latent representation.
- **Aggregator:** Masked global mean pooling and max pooling over all active stock embeddings, concatenated with the current cash weight.
- **Actor/Critic Heads:** Both the Actor (policy) and Critic (value function) utilize independent MLPs with hidden dimensions $[128, 64]$. The Actor outputs unnormalized logits mapped to a valid probability distribution (long-only weights) via a masked Softmax layer (Jiang, Xu, and Liang 2017).

A.2 Execution Cost Formulation

To maintain realism, the environment deducts transaction costs C_t at every rebalance, modeling both fixed frictional costs and scale-dependent market impact (Almgren et al.

2005). Let Δw_i be the change in target weight for asset i, and K_t be the total portfolio gross capital (initialized at \$10M USD to simulate institutional scale):

$$C_{spread} = \sum_i \frac{1}{2} \cdot |\Delta w_i| \cdot \text{BidAskSpread}_i \tag{4}$$

$$C_{impact} = \sum_i \eta \cdot \frac{\text{VolMkt}}{\sqrt{12}} \cdot \sqrt{\max\left(0, \frac{|\Delta w_i| \cdot K_t}{\text{DolVol}_i}\right)} \cdot |\Delta w_i| \tag{5}$$

Where $\eta = 1.0$ is the impact scaling coefficient, $\text{VolMkt}/\sqrt{12}$ translates annualized volatility to a monthly horizon, and DolVol_i is the raw traded dollar volume of the asset. The gross return is subsequently adjusted by $C_t = C_{spread} + C_{impact}$ alongside the risk-free rate accrued on unallocated cash.

A.3 Proximal Policy Optimization (PPO) Hyperparameters

The agent is trained using the Stable-Baselines3 implementation of PPO (Schulman et al. 2017). We substantially tightened standard clipping and entropy parameters to stabilize learning in a low signal-to-noise financial environment:

TABLE III
PPO TRAINING CONFIGURATION

Parameter	**Value**
Total Timesteps	$1,000,000$
Parallel Environments (N_{envs})	8
Episode Length	36 Months
Batch Size	288
Epochs per Rollout (K)	10
Learning Rate	3.0×10^{-5}
Discount Factor (γ)	0.999
GAE Parameter (λ)	0.95
Clipping Range (ϵ)	0.1 (Tightened)
Target KL Divergence	0.02 (Early stopping triggered if exceeded)
Entropy Coefficient	0.005
Value Function Coefficient	0.5
Max Gradient Norm	0.5

The reduction of the learning rate to 3×10^{-5} and the introduction of a Target KL early-stopping threshold (KL > 0.02) prevents catastrophic policy updates driven by highly volatile monthly market anomalies.

A.4 Environment Regularization: Stock Dropout

To mitigate the risk of the RL agent memorizing specific historical trajectories of dominant mega-cap stocks (survivorship and look-back bias), we introduce a novel environment-level augmentation technique termed *Stock Dropout* (Srivastava et al. 2014).

Unlike standard neural network dropout applied to latent feature representations, Stock Dropout operates directly on the environment's investable universe. At the initiation of each 36-month training episode, the environment evaluates the full cross-section of valid assets, N_t. A configured fraction p (in our pipeline, $p = 0.20$) of these assets is uniformly sampled and explicitly removed from the `active_universe`.

Crucially, this exclusion mask is locked for the entirety of the episode. Because the policy network utilizes a permutation-invariant Set Transformer architecture, it natively accepts this dynamically reduced state space without any structural modifications.

By forcing the agent to construct optimal portfolios and navigate execution costs without relying on historically obvious "winners," the policy is coerced into learning the true generalized relationships between the upstream XGBoost alpha, liquidity constraints, and expected returns. During out-of-sample evaluation and live testing, the dropout rate is disabled ($p = 0$), allowing the trained agent to allocate across the complete asset universe.

Fig. 1. Mean realized return across predicted deciles. The upper deciles exhibit an approximately monotonic increase, confirming that the ranking model successfully identifies the best-performing assets.

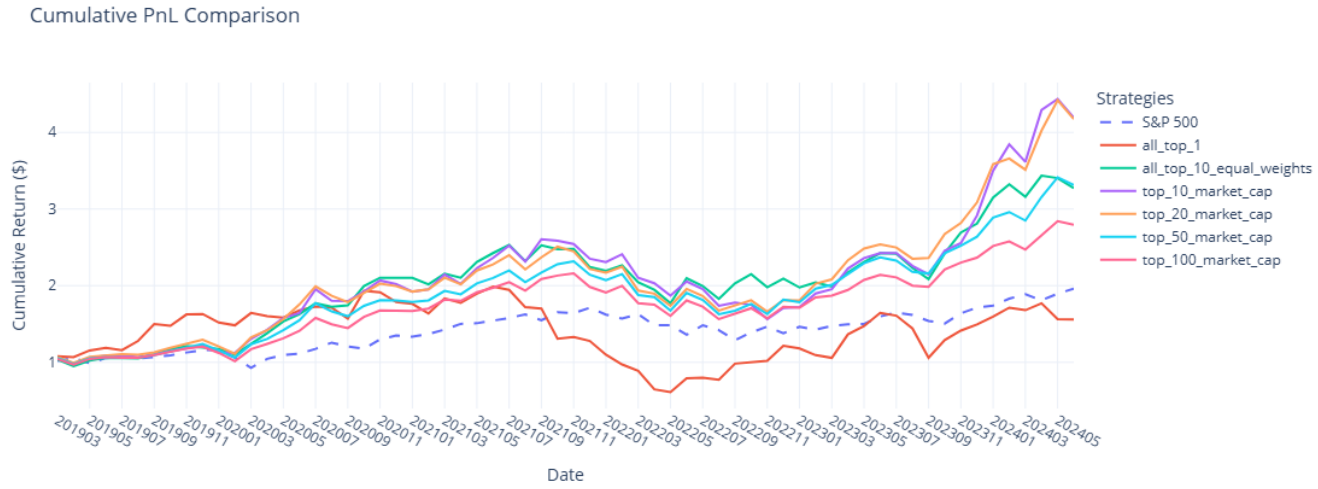

Fig. 2. Cumulative PnL comparison between the benchmark and the long-only portfolios constructed from the highest-ranked assets. Despite periods of drawdown, the overall trend remains clearly upward.