A Human in the Loop Alpha Generation

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Fatima Rahimi

Annandale-on-Hudson, New York
May, 2026

Abstract

We extend Vuletic et al.'s FinGAN framework for forecasting ETF excess returns of S&P 500 stocks. The thesis contributes four elements: a modular TensorFlow reimplementation of the original PyTorch codebase; an architectural extension, FinTimeGAN, that introduces a TimeGAN style Embedder Recovery pair to run the adversarial game in a learned latent space; an independent branch loss training protocol that resolves a discrepancy between Vuletic's published algorithm and her reference implementation; and an empirical comparison of input return representations.

Models are evaluated on a 28 stock single name universe spanning nine GICS sectors, and on a Vuletic comparable universe of 22 stocks and 9 sector ETFs. The evaluation framework separates model selection (by validation Sharpe), statistical significance gating (a portfolio level 200 permutation test), and distributional sanity gating (collapse flags) into three independent axes, and reports per ticker results descriptively rather than as binary verdicts.

On the 28 stock universe, FinGAN achieves a portfolio weighted signal Sharpe of +2.022 at $p < 10^{-4}$, exceeding the ARIMA(2,0,2) baseline at +0.642 and the Ridge AR(10) baseline at +0.227. On the Vuletic comparable universe, FinGAN reaches a portfolio Sharpe of +3.17, surpassing the best result reported by Vuletic et al. on the same universe. Cross sectional analysis indicates that the gains from generative modeling come from amplifying existing signals in defensive and utility stocks rather than from extracting signals where linear baselines fail.

Dedication

As Ferdowsi writes,

توانا بُوَد هر که دانا بُوَد
ز دانش دل پیر بُرْنا بُوَد

Mighty is she who has knowledge;
from knowledge, old hearts become young.

I dedicate this senior project to all the immigrant children whose only way to shape their future, in the country they have come to call home, is through knowledge.

Acknowledgments

I am deeply grateful to my supervisor, Valerie Barr, for her unwavering support and guidance through the ups and downs of getting this project done. Beyond her feedback and encouragement, which shaped this work at every stage, she has been a source of inspiration and a role model to me. Watching her work long hours, and finding her replies to my emails arriving on weekend evenings, taught me what it means to never stop working hard for the things that matter.

I would also like to thank Emanuele Citera for supporting me in writing this project from a richer finance perspective.

I am thankful to Bard College for providing the opportunity to continue my education despite all immigration effects and the computational resources that made this work possible, and to Milena Vuletic and her co-authors for releasing the original FinGAN code, which served as the reference implementation throughout this senior project.

Finally, I owe more than I can express to my family. To my parents and sisters, whose belief in education carried me through every stage of my studies long before this project began, and whose support made it possible for me to be here at all. And to Milad Mehraban, who saw me through this in ways I will not try to put into words, and who reminded me I was capable of finishing it when I was most certain I was not. This senior project would not exist without them.

Contents

Chapter 1

Introduction

Investment plays a central role in the modern economy, and quantitative methods for identifying profitable trading opportunities have been an active area of research for decades. Researchers have developed mathematical and statistical models to capture patterns in financial data and to support decisions about asset allocation and trading[3][33][22][10]. More recently, advances in neural network architectures, together with increased computational power and access to large scale financial datasets, have accelerated interest in applying machine learning techniques to investment decision making. These approaches can model complex, high dimensional relationships in market data that are difficult to capture with traditional methods, and they have become an increasingly important tool in quantitative finance.

Despite their promise, financial markets pose unique challenges for data driven models. Financial time series are noisy, non stationary, and strongly influenced by market structure and economic constraints. As a result, effective financial modeling requires not only expressive learning architectures but also careful consideration of portfolio construction principles, risk measurement, and evaluation metrics that align with real world trading objectives.

This project addresses one specific problem within financial machine learning: forecasting the next half trading day excess return of individual US equities relative to their sector ETF, and doing so in a way that captures not only the most likely outcome but the full distribution of possible outcomes. Two scope choices are worth naming up front. First, this is a *per asset* forecasting problem, not a cross sectional ranking problem: we train and evaluate models that

predict each stock's return on its own, rather than ranking stocks against each other on a given day. Second, we work at *half trading day frequency* rather than the more common daily frequency: each trading day is split into an overnight *closetoopen* return and an intraday *opentoclose* return, which are interleaved into a single sequence so that the model can exploit the distinct statistical properties of each half session. Both choices are motivated and implemented in detail in Chapter 4.

The remainder of this chapter introduces the financial concepts needed to follow the later chapters and explains why single point forecasts are insufficient for financial decision making.

1.1 Finance Primer

This section covers the basic financial concepts used throughout the project. Readers from a finance background can safely skim it; it is written for the non finance reader.

1.1.1 What is finance?

Finance encompasses a wide range of economic activities, including money management, banking, pensions, mutual funds, hedge funds, and loans. A large portion of these activities centers on *investment*, which refers to taking an ownership position in an asset with the goal of earning a positive return. Financial products such as stocks, bonds, commodities, and foreign exchange are commonly treated as investable assets.

Financial products are often divided into two broad categories. The first is *cash products*, which include stocks, bonds, currencies, and commodities. The second is *derivatives*, which include forwards, futures, swaps, and options. A derivative obtains its value from an underlying asset, and its payoff is determined by the relationship between the derivative contract and that underlying instrument.

The concept of value in finance has two interpretations. The *market price* is determined by supply and demand in live trading. The *fair value* is the theoretically appropriate price based on expected future cash flows and discounting. Fair value is closely related to the time

value of money, discounted cash flow techniques, and arbitrage principles. Arbitrage refers to the possibility of earning a risk free profit with zero net investment by exploiting price inconsistencies across assets or markets.

Stocks, or equities, represent ownership in a company. Shareholders may receive dividends, although many companies, particularly American ones, no longer pay them. Shares may also provide voting rights depending on the company's charter. Stock trading occurs in private markets, where transactions are negotiated directly, and in public markets such as the New York Stock Exchange, NASDAQ, and Euronext. Public markets offer greater transparency and liquidity, and prices in these markets fluctuate due to company performance, macroeconomic conditions, and market sentiment. This project works exclusively with publicly traded US equities and sector level exchange traded funds, both of which are priced and traded on public exchanges.

1.1.2 What is Modern Porfolio Theory?

Modern Portfolio Theory (MPT) is a mathematical framework for constructing investment portfolios that maximize expected return for a given level of risk. The theory was developed by Harry Markowitz and first introduced in his 1952 paper "Portfolio Selection" published in the Journal of Finance.[26] His contributions on how he changed Portfolio selection structure to focus on risk return trade off, earned him the 1990 Nobel Prize in Economics.

A central principle of MPT is *diversification*. While individual investments exhibit a trade off between risk and return, with higher returns typically coming with higher risk, Markowitz[26] demonstrated that combining assets with different risk characteristics can improve overall portfolio performance. By selecting an optimal mix of assets aligned with an investor's risk tolerance, it is possible to achieve more efficient risk return outcomes than by holding individual securities alone [36].

The expected return of a portfolio is the weighted sum of the expected returns of its individual assets. For a portfolio of four equally weighted assets with expected returns of 4%, 6%, 10%,

and 14%, the portfolio expected return is

$$(0.25 \cdot 4\%) + (0.25 \cdot 6\%) + (0.25 \cdot 10\%) + (0.25 \cdot 14\%) = 8.5\%. \tag{1.1}$$

Portfolio risk, however, does not decompose so simply. It depends not only on the individual variances of each asset but also on the correlations between asset returns. Because returns are not perfectly correlated, the overall portfolio risk, measured by standard deviation, is typically lower than what a naive weighted average of individual risks would suggest. This is the mathematical foundation of diversification: combining imperfectly correlated assets reduces total risk without proportionally reducing expected return.

1.1.3 Alpha

Alpha is one of five pillars of investment analysis in modern portfolio management. The other four are beta, standard deviation, R squared, and the Sharpe ratio. Alpha measures the performance of a portfolio relative to a benchmark. According to Investopedia, "Alpha (α) is an investing term that measures an investment strategy's ability to outperform the market, often called its edge. It represents the excess or abnormal return of an investment compared to a benchmark index, adjusted for risk"[3]. A positive alpha means an investment outperformed its benchmark; a negative alpha means it underperformed. Alpha is usually considered together with beta(β), which captures systematic risk.

Alpha is often described as the "holy grail" of investing and attracts significant attention from both investors and advisors. In its simplest form, alpha is the difference between an investment's total return and the return of a benchmark within the same asset class. It is therefore meaningful only when the benchmark is appropriate and comparable: the alpha of an equity ETF cannot be directly compared to that of a fixed income ETF. More sophisticated formulations exist, including Jensen's alpha, which incorporates the Capital Asset Pricing Model (CAPM) by adjusting for systematic risk through the risk free rate and beta. When working with computed alpha, it is essential to understand the underlying methodology: alpha depends

on the choice of benchmark index within an asset class, and when no suitable index is available, practitioners may rely on algorithmically constructed benchmarks.

The models in this project forecast excess returns against a sector ETF benchmark. That is, they forecast the alpha component of a stock's return with respect to its sector. A successful forecast of excess returns translates directly into a trading strategy that targets alpha rather than broad market exposure.

1.2 Financial Data

Financial data are highly noisy, meaning that unpredictable fluctuations often overwhelm the underlying signal. Noise reduction and feature extraction therefore become essential components of financial modeling. While many early studies examined single asset prediction problems, real markets involve many interacting securities. As a result, researchers increasingly study multi asset datasets where models must capture cross sectional relationships and manage high dimensionality in a stable and scalable way. The Financial Data comes in different forms. Below we will discuss the different types of financial data.

1.2.1 Historical price Market Data (Day wise)

In general, historical data consists of open price, high price, low price, close price, and traded volume for each stock. Following is a sample of Yahoo Finance day wise data set. [1]

Date	Prev. Close	Open	High	Low	Close	Volume
1 December 2014	533.5	539.85	539.85	530.1	536	5093458
2 December 2014	536	535.7	535.8	528.1	528.95	3555543
3 December 2014	528.95	535.2	538.8	526.3	529	3997174
4 December 2014	529	534	537.25	522	527.75	3600835
... ...	... ...	... ...	... ...	... ...	... ...	... ...

Figure 1.1: A sample of the TATAMOTORS company's day-wise historical data.

[1]Here is Daily equity price data set of Yahoo on Kaggle dataset website `https://www.kaggle.com/datasets/suruchiarora/yahoo-finance-dataset-2018-2023/data`

1.2.2 Historical Price Market Tick Data (Minute wise)

Tick data generally contain the open price, high price, low price, close price, and trading volume of the stock, and they are used for prediction in intraday trading. Following is an example of the minute by minute stock data.

Stock Symbol	Time	Open	High	Low	Close	Volume
SBIN	09:08	492.65	492.65	492.65	492.65	53244
SBIN	09:16	492.8	494.6	491.85	494	159589
SBIN	09:17	493.9	494.15	493.75	494.15	162984
SBIN	09:18	494.3	495	493.5	494	123425
... ...	... ...	... ...	... ...	... ...	... ...	... ...

Figure 1.2: An example of the minute by minute historical data collected by the State Bank of India [30]

1.2.3 Sentiment Data

Tweets and comments on twitter is one of the often used social network data for sentiment research. These data are usually labeled as favorable, neutral, and negative.

1.2.4 Image Data

Image data are increasingly being used as input to convolutional neural networks (CNNs) for financial modeling and analysis. Candlestick charts are usually used as the input to CNNs. As seen in Figure1.3, these charts show the price variations through some time frame.

1.2.5 Fundamental Data

This type of data often focuses on underlying business of a company's stock. The company's performance is shown through financial statements such as annual report, balance sheets, profit loss statement, management information, competitors information, products, economic and political factors, and industrial relationship. Below is an example of Fundamental Data.

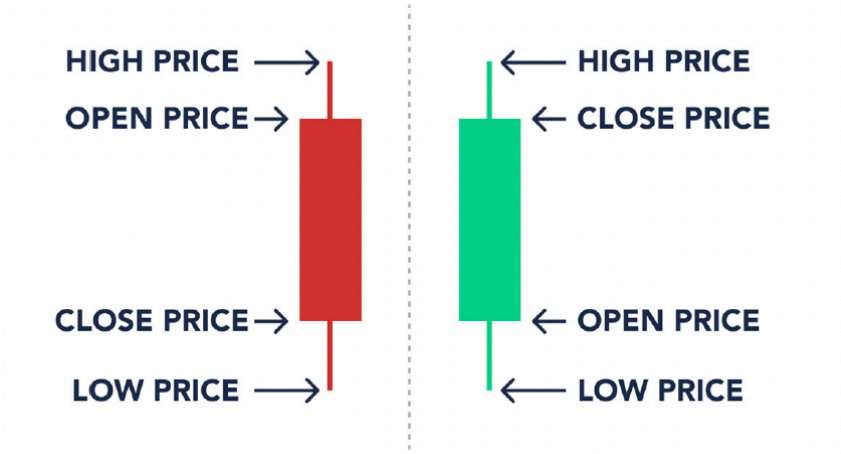


Figure 1.3: A candlestick pattern is a price chart pattern used in technical analysis of the financial markets.

	ticker	dimension	calendardate	datekey	reportperiod	fiscalperiod	lastupdated	accoci	assets	assetsavg	...	sharesbas	shareswa	shareswadil	sps	tangibles	taxassets	taxexp	taxliabilities	tbvps	workingcapital
None																					
0	AAPL	MRY	2025-12-31	2025-09-27	2025-09-27	2025-FY	2026-01-30	-5.571000e+09	3.592410e+11	3.415135e+11	...	1.484039e+10	1.494850e+10	1.500470e+10	27.840	3.592410e+11	0.0	2.071900e+10	0.0	24.032	-1.767400e+10
1	AAPL	MRY	2024-12-31	2024-09-28	2024-09-28	2024-FY	2026-01-30	-7.172000e+09	3.649800e+11	3.468792e+11	...	1.520414e+10	1.534378e+10	1.540810e+10	25.485	3.649800e+11	0.0	2.974900e+10	0.0	23.787	-2.340500e+10
2	AAPL	MRY	2023-12-31	2023-09-30	2023-09-30	2023-FY	2026-01-30	-1.145200e+10	3.525830e+11	3.416320e+11	...	1.563423e+10	1.574423e+10	1.581255e+10	24.344	3.525830e+11	0.0	1.674100e+10	0.0	22.394	-1.742000e+09
3	AAPL	MRY	2022-12-31	2022-09-24	2022-09-24	2022-FY	2026-01-30	-1.110900e+10	3.527550e+11	3.552292e+11	...	1.607075e+10	1.621596e+10	1.632582e+10	24.317	3.527550e+11	0.0	1.930000e+10	0.0	21.754	-1.857700e+10
4	AAPL	MRY	2021-12-31	2021-09-25	2021-09-25	2021-FY	2026-01-30	1.630000e+08	3.510020e+11	3.430135e+11	...	1.653017e+10	1.670127e+10	1.686492e+10	21.904	3.510020e+11	0.0	1.452700e+10	0.0	21.016	9.355000e+09
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
580	AAPL	ARQ	1995-03-31	1995-05-15	1995-03-31	1995-Q2	2026-01-30	4.400000e+07	6.050000e+09	NaN	...	1.356975e+10	1.373613e+10	1.373613e+10	0.193	6.050000e+09	350000000.0	4.300000e+07	789000000.0	0.440	2.918000e+09
581	AAPL	ARQ	1994-12-31	1995-02-09	1994-12-30	1995-Q1	2026-01-30	-1.700000e+07	5.533000e+09	NaN	...	1.352535e+10	1.361922e+10	1.361922e+10	0.208	5.533000e+09	269000000.0	1.110000e+08	732000000.0	0.406	2.783000e+09
582	AAPL	ARQ	1994-09-30	1994-12-13	1994-09-30	1994-Q4	2026-01-30	-1.083400e+07	5.302746e+09	NaN	...	1.342784e+10	1.329832e+10	NaN	0.187	5.302746e+09	293048000.0	7.027200e+07	670668000.0	0.399	2.532147e+09
583	AAPL	ARQ	1994-06-30	1994-08-12	1994-07-01	1994-Q3	2026-01-30	-1.547400e+07	5.123412e+09	NaN	...	1.332185e+10	1.332185e+10	NaN	0.161	5.123412e+09	256945000.0	8.464200e+07	655995000.0	0.385	2.342502e+09
584	AAPL	ARQ	1993-12-31	1994-01-26	1993-12-31	1994-Q1	2026-01-30	-2.650900e+07	5.042440e+09	NaN	...	1.313121e+10	1.309907e+10	1.309907e+10	0.188	5.042440e+09	279198000.0	2.452600e+07	663906000.0	0.385	1.879846e+09

Figure 1.4: A fundamental data table from Sharadar's fundamental table

The combination of noisy financial data, high dimensional asset spaces, and non stationary market dynamics has motivated a wide range of modeling approaches in quantitative finance. Researchers have explored traditional statistical models, machine learning techniques, and deep learning architectures, each with different assumptions and limitations. The following chapter reviews prior work that addresses these challenges, with particular emphasis on methods for alpha generation, representation learning, and multi asset modeling.

1.3 Why point predictions are not enough

Traditional supervised approaches to return forecasting, such as linear regression, gradient boosted trees, and LSTMs (Long short-term memory networks) produce a single point prediction for each future timestep. For financial decision making, a point prediction is not enough, because it says nothing about the uncertainty around it. A predicted return of 0.2% means something very different if the true return could plausibly range from -5% to $+5\%$ than if it could only range from -0.3% to $+0.7\%$.

Consider two stocks, both with an expected next day return of 0.2%.

- Stock A has a standard deviation of 3.5%: its realized return on any given day could easily be anywhere from a 3% loss to a 3% gain. The expected return of 0.2% is completely swamped by the day to day noise. A risk manager looking at this forecast would either avoid the trade entirely or take a very small position, because the prediction is indistinguishable from zero in practical terms.

- Stock B has a standard deviation of 0.5%: its realized return is much more tightly distributed around 0.2%, and an adverse move beyond about -0.8% would be unusual. The forecast is far more actionable, and the risk manager can size this position larger without taking on meaningful risk of catastrophic loss.

The two forecasts have identical expected values but the trading implications are entirely different. A point prediction model cannot distinguish Stock A from Stock B: both predictions are just "0.2%." A model that instead produces a full distribution can. The same lesson applies

to evaluation. Risk adjusted performance metrics like the Sharpe ratio depend on the full distribution of returns across many days, not on the accuracy of any single prediction. A model that happens to predict the right direction on average but with high variance will score poorly on Sharpe even if its point accuracy looks good.

Distributional forecasting can be approached in several ways. Parametric methods such as *Quantile regression* [2] or *Bayesian neural networks* produce distributional output by placing assumptions on the shape of the output distribution: normality, mixture of Gaussian, or a specified set of Quantile levels. There has been research done on application of *Generative Adversarial Networks*5.3 in the financial setting because they make no such parametric assumption. The distributional shape is learned from the data itself rather than imposed by the modeler, which matters for financial returns because they are famously not normally distributed: they exhibit heavy tails, asymmetry, and volatility clustering that any fixed parametric family handles only approximately.

What is needed, then, is a model that samples from the *conditional distribution* of future returns given past history. Given a conditioning window of recent returns, the model should produce not one forecast but many plausible forecasts, which together describe both the most likely outcome and the uncertainty around it. The models in this project therefore produce *distributional* forecasts. Rather than emitting one number per input, they sample many plausible next step returns from a learned conditional distribution, which together describe both the most likely outcome and the uncertainty around it.

This is one specific approach within a broader landscape. Researchers have tackled the limitations of traditional return forecasting in several ways: through hand engineered factor discovery, reinforcement learning over symbolic alphas, deep learning on order book microstructure, hybrid neural architectures, and generative models. The next chapter reviews five representative lines of work in this space, with particular attention to how each handles the joint problem of high

[2] Quantile regression models the relationship between a set of predictor (independent) variables and specific percentiles (or "quantiles") of a target (dependent) variable, most often the median.[?IBM]

dimensionality, noise, and distributional output. Chapter 3 then states the specific contributions this project makes within that landscape.

Chapter 2

Background

As argued in the introduction chapter, the financial decision making requires distributional forecasts rather than point predictions, and this requirement narrows the space of viable model classes. This chapter situates that requirement within the broader landscape of recent work in machine learning based alpha generation. Modern quantitative trading increasingly faces the challenge of modeling high dimensional, noisy, and non stationary financial time series across many assets and horizons. Traditional approaches such as linear factor models and time series models fail to capture non linearity, market regime shifts, and cross sectional dependencies.

A common problem in this area is that the number of assets under study is often much larger than the number of observations available for each. For example, a researcher may want to model 2,000 stocks using only 250 days of history. Downstream tasks such as covariance estimation, return distribution modeling, tail risk assessment, and stress scenario construction all require large amounts of data relative to the number of parameters, and having fewer observations than assets strongly increases the risk of overfitting, particularly for non linear models.

In response to these challenges, recent work has explored deep learning architectures for alpha generation, hybrid neural networks that combine multiple model families, and generative models that can produce realistic financial scenarios under limited data. At the same time, there is growing recognition that model architectures, feature engineering, and evaluation metrics must be designed with explicit attention to financial structure, economic interpretability, and trading feasibility.

We begin this chapter by reviewing the classical approaches that preceded the deep learning era, then turns to five representative strands of more recent machine learning work: a neural network designed for feature construction and alpha discovery on Chinese equities [12], a reinforcement learning framework that searches the space of symbolic alphas[35], a study of deep learning models for high frequency limit order book forecasting on US stocks [22], a hybrid CNN-LSTM-GNN architecture for cross sectional stock selection [11], and implementation of generative models that embeds the financial structure directly into model for U.S. equity stocks [10][33]. It also reviews the evaluation metric and validation literature that underpins how any of these models should be judged. The final section focuses on Vuletic's FinGAN [33], which is the direct predecessor of the work presented here.

2.1 Classical Approaches to Financial Forecasting

Before the deep learning era, financial time series forecasting was dominated by two families of parametric models, ARMA and GARCH. Auto regressive Moving Averages (ARMA) are for the conditional mean of returns, and Auto Regressive conditional heteroskedasticity (ARCH/GARCH) models are for mean of returns variance. These models are the standard baseline in applied finance and this project uses an ARMA family benchmark in Chapter 5.

The ARMA family was formalized in the Box Jenkins methodology [8], which provides a systematic procedure for identifying, estimating, and diagnosing time series models for stationary series. An ARMA(p, q) model expresses the current observation X_t as a linear combination of p past observations and q past innovations:

$$X_t = \alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}, \tag{2.1}$$

where $\{\varepsilon_t\}$ is a white noise sequence. When the series is non stationary, an integration order d is applied (differencing the series d times to induce stationarity), giving the more general ARIMA(p, d, q) specification [8]. Model order is typically selected using information criteria such as AIC[1], which trade off log likelihood against model complexity.

[1] More explanation in Chapter 5

ARMA family models have well known limitations for financial data. They assume the conditional mean is the only time varying quantity of interest and treat the residual variance as constant. But financial returns exhibit *volatility clustering*: large moves tend to be followed by large moves of either sign, and calm periods by calm periods. Engle's ARCH model [13] addressed this by letting the conditional variance depend on recent squared residuals, and Bollerslev's GARCH [7] extended this with additional lags on the conditional variance itself, giving a compact model of volatility dynamics. GARCH and its variants remain the benchmark for volatility forecasting in finance[32].

The fundamental reason these classical methods struggle as the basis for a modern forecasting system is that they are strictly linear in their conditioning. ARMA models can only capture linear dependencies between past and future returns, and GARCH models can only capture linear dependencies in squared returns. Cross asset dependencies, regime dependent dynamics, and non linear interactions between features all lie outside what these models can express. This motivates the shift to more flexible learning architectures that the rest of this chapter reviews.

2.2 Alpha Discovery Neural Network (ADNN)

Fang et al. [12] propose the Alpha Discovery Neural Network (ADNN) as a way to automatically construct technical factors from raw OHLCV (open, high, low, close, volume)[2] time series for Chinese equities. The framework is flexible about the feature extractor and evaluates fully connected networks[3], Convolutional Neural Networks (CNN)[4], and recurrent[5] and Transformer based extractors[6]. CNNs are well suited to detecting local patterns in time series such as short

[2]Are the raw price data for individual stocks. More explained in 1.2

[3]Fully connected networks are the simplest neural network architecture, where every neuron in one layer connects to every neuron in the next. They have no built in notion of temporal order or spatial locality.

[4]CNNs are neural network architectures that apply learned filters across their input to detect local patterns regardless of position. Originally developed for image recognition, they have been adapted to time series by sliding filters along the temporal axis to detect short-range patterns such as momentum bursts or volatility clusters.

[5]Recurrent Neural Networks (RNNs) process sequences one element at a time, maintaining a hidden state that carries information forward across timesteps. This makes them suited to modeling temporal dependencies in financial data.

[6]Transformers process sequences in parallel using self attention, which lets each element of the sequence directly access information from every other element. They have been applied to financial time series for tasks where long range dependencies matter.

term price movements or repeated structures in technical indicators, while RNNs are better at modeling the sequential dependencies that link past market conditions to future outcomes [30]. ADNN treats the choice of extractor as a hyperparameter and reports results across all four.

ADNN differs from traditional approaches in two ways. First, its training batches group all stocks from a single trading day together, rather than forming batches by ticker. The authors argue that this aligns with how quantitative investors actually think: what matters is the *cross sectional*[7] strength of stocks on a given day rather than how a single stock moves across time. Second, instead of optimizing a standard regression loss against future returns, ADNN optimizes a differentiable approximation of the Spearman rank correlation between predicted factor values and realized future returns. This makes the learning objective reflect the cross sectional ranking quality that matters in practice.

The model is trained in two stages. First, the network is pretrained as a multilayer perceptron to reproduce classical technical indicators including moving averages (MA, EMAA.1), MACD, RSIA.1, Bollinger Bands, and Donchian channels.[8] The pretraining gives the network an informed starting point that already encodes well known factor structure. Second, the embedding matrix is pruned to remove noisy entries and the model is fine tuned to maximize rank correlation with future returns. The inputs are the most recent 30 trading days of OHLCV, and the model predicts returns five days ahead.

The authors evaluate ADNN with a rolling backtest[9], training on 250 days, validating on 30 days, and testing on 90 days, then advancing the window forward. The factors generated by ADNN improve multi factor portfolio return, Sharpe ratio, and maximum drawdown compared to genetic programming baselines and to handcrafted traditional indicators.

Two limitations are worth flagging. First, the learned features are not explicit functions, which makes economic rationale and risk validation harder than for handcrafted indicators. Second,

[7] Cross sectional relative strength means that for each trading day, we care about which stocks are stronger or weaker relative to one another instead of considering one stock's strength or weakness across some time frame.A.2

[8] These are standard momentum and volatility indicators in technical analysis. Refer to math glossary for better definitionsA

[9] Backtesting is the process of evaluating a trading strategy by applying it to historical market data and measuring how it would have performed.

pretraining on classical indicators may bias the network toward rediscovering refined versions of existing factors rather than producing genuinely novel structures.

2.3 Reinforcement Learning for Symbolic Alpha Search

Building on research in formulaic alpha generation, recent work explores reinforcement learning (RL) as a mechanism for searching the space of symbolic trading factors. Reinforcement learning is a learning paradigm in which an agent learns to act in an environment by receiving rewards for its actions, rather than by being given correct outputs to imitate. The agent's behavior is governed by a policy that maps observed states to actions, and the training objective is to find the policy that maximizes expected cumulative reward over time.

Yu et al. (2023)[35] introduced a Proximal Policy Optimization (PPO) [10] framework that generates formulaic alphas from raw stock features (Open, high, close, low, volume, VWAP) and jointly optimizes the weights that used to combine them. PPO select tokens (operands, operators, and constants) one at a time to build symbolic expression, while a separate module determines the weights that combine the resulting alpha. This study expands an earlier search space by including richer set of operations; signA.4, CSRank (cross sectional rank)A.4, Product (rolling product)A.4, Scale A.4, Pow A.4, Skewness A.4, Kurtosis A.4, Rank (time series rank)A.4, Delta A.4, ArgmaxA.4/Argmin, and the Cond (conditional)A.4. The aforementioned addition let the agent build more expressive formulas spanning a wider range of statistical and temporal transformations.

The performance is reported using two standard metrics: the information coefficient (IC), the Pearson correlation between factor values and the target return, and Rank IC, the Spearman correlation of cross sectional ranks. Together these metrics assess whether factors correlate with future movements and rank stocks correctly within each day's cross section.

[10]PPO is type of reinforcement learning (RL) algorithm developed by OpenAI and is widely used because it is stable, relatively easy to implement, and performs well in many environments. PPO trains an agent to take actions that maximize long term reward. PPO improves the policy (the agent's decision rule) while making sure the new policy does not move too far away from the old one too quickly. This prevents instability, which is common in reinforcement learning.

RL training proceeds in two stages. First, the model generates a large "mega-alpha" set from an empty initialization. Second, this generated set is used as a seed for subsequent mining, biasing the policy toward productive regions of the search space and improving sample efficiency. The authors also handle survivorship bias carefully: each stock enters the dataset at its actual listing date, and delisted stocks remain in the universe for the periods during which they were tradable. Experiments use CSI300 data with a training period of 2009-2018, validation in 2019, and testing over 2020-2021 in the Qlib framework.[11]

Empirically, the framework produces higher IC and Rank IC than the earlier baseline, and performance improves further when both search space expansion and seed initialization are applied. Several limitations remain. The model imposes a formula length cap, which restricts how expressive the generated factors can be. The authors also identify a practical issue: if the experience replay buffer is not reset when the alpha set is reinitialized, early stage IC deteriorates because of stale experiences. More broadly, RL methods in finance tend to be sensitive to non-stationary market conditions, where the statistical properties of market data change over time. Patterns learned from past data may not remain stable in the future, which limits how well RL discovered factors generalize across regimes.

2.4 Deep Learning for Limit Order Book Forecasting

A limit order book (LOB) is an instruction to buy or sell a security at a specified price or better, and the LOB aggregates all outstanding limit orders at each price level. Because buyers and sellers post orders at different prices and depths, the LOB contains rich information about near term price pressure. The LOB is a different part of the market which is considered a high frequency trading. Kolm et al. [22] derive *order flow imbalance* (OFI) [12] from the top ten bid

[11] Qlib is an open-source quantitative-research platform originally developed by Microsoft, covering the full workflow from data processing to model training and backtesting. https://github.com/microsoft/qlib

[12] Order flow imbalance is computed as the difference between volume added to the bid side and volume added to the ask side of the order book over a short time window. Increases in posted bid volume signal buying pressure, increases in posted ask volume signal selling pressure, and the net difference summarizes which side is currently dominating.

and ask [13] levels of the NASDAQ LOB, a signal that measures the net buying or selling pressure at a given moment by tracking changes in posted volume on the bid and ask sides [4]. They then use OFI to forecast returns over multiple short horizons.

They compare four LSTM-based architectures: a many-to-one LSTM, a multi-head LSTM [14] with separate output heads for each forecast horizon, a sequence-to-sequence encoder decoder, and a sequence-to-sequence model with attention. The dataset is terabytes of trade level data from 2014 and 2015 for the ten largest NASDAQ stocks. Their main finding is that a simple LSTM is surprisingly hard to beat. Adding depth yields only marginal gains, increasing width can reduce performance, and the attention-based variants perform worse than the plain LSTM on this task.

The paper produces several useful practical insights for network design in high frequency settings. First, smaller batch sizes (64 or 256) lead to better generalization. Second, including explicit time representations substantially improves forecasts. The authors test duration between updates, time of day, and Time2Vec embeddings of intraday time,[15] and the combination of time of day and inter update duration is the most informative. Third, look-back windows beyond roughly 100 LOB updates do not help and can hurt performance. Finally, stocks with information rich microstructure[16], where the order book updates frequently relative to price changes, yield the most accurate forecasts.

The work has several limitations. The experiments focus on a narrow set of large and highly liquid US technology names. This limits the paper's insight into how microstructure varies across stocks. A second concern is the reliance on order flow imbalance, which is informative

[13]The *bid* is the highest price a buyer is currently willing to pay, and the *ask* is the lowest price a seller is currently willing to accept. The gap between them is the bid ask spread. The "top ten levels" means the ten best bid prices and the ten best ask prices, with all the orders posted at each.

[14]Not to be confused with multi-head attention. A multi-head LSTM produces separate output predictions for each forecast horizon from a shared LSTM body, with one "head" (a small output network) per horizon.A.2

[15]Time2Vec, introduced by Kazemi et al.[21], is a learned low dimensional vector embedding of time. Unlike scalar timestamps, the vector representation can capture periodic and continuous temporal patterns and is used as an input feature alongside the rest of the model's inputs.

[16]Market microstructure refers to the fine grained mechanics of how trades actually happen: the rate at which orders arrive and get canceled, the depth of the order book at each price level, the size of the bid-ask spread, and how frequently the book updates relative to actual price changes. A stock has "information-rich" microstructure when its order book updates many times between price changes, so each update carries meaningful information about near term supply and demand pressure rather than just confirming the last price.

but extremely noisy in isolation. Models that lean heavily on OFI risk learning fragile patterns that do not survive regime changes. The evaluation also leans heavily on the R^2A.2 statistic for return prediction, which understates the model's true predictive value for trading purposes. Financial returns are dominated by noise, so even a model that captures useful directional information will explain only a small fraction of total variance and produce a misleadingly low R^2. More economically meaningful measures such as the Sharpe ratio, hit rate, and directional accuracy receive less attention than they deserve.

2.5 Hybrid Architectures for Cross Sectional Selection

Dong and Liang [11] propose a hybrid CNN-LSTM-GNN [17] architecture (CLGNN) for stock selection in the Chinese A-share market. They argue that A-share markets [18] differ materially from developed US markets, so the volume price patterns and empirical regularities found in US data do not transfer directly. Their model combines three components: Convolutional and LSTM layers extract local and global temporal patterns from 60 day windows of daily return, turnover [19], RSIA.1, volume, and forward adjusted close prices [20]; and a graph neural network A.2, inspired by MTGNN [`?Wu2020MTGNN`], learns a directed adjacency structure across stocks to capture cross sectional dependencies. Each stock is classified into one of five weekly return categories (Surge, Rise, Flat, Fall, Plummet)[21], and the classifications are combined into a long

[17] The architecture combines three neural network families: a *convolutional neural network* (CNN) applies learned filters across local neighborhoods of the input to detect short range patterns, a *long short term memory network* (LSTM, described in chapter 5.7.5) captures longer range temporal structure through gated recurrent connections, and a *graph neural network* (GNN) propagates information along edges of a learned adjacency matrix to capture relationships between different stocks.

[18] The Chinese A-share market refers to stocks of mainland Chinese companies that trade on the Shanghai Stock Exchange and the Shenzhen Stock Exchange, denominated in Chinese yuan and historically restricted to domestic investors. The market differs from US equity markets in several ways: retail investors account for a much larger share of trading volume, daily price moves are capped at $\pm 10\%$ for most stocks, short selling is heavily restricted, and the listing universe excludes foreign companies. These structural differences mean that empirical patterns documented in US data do not always transfer directly.

[19] Turnover is the ratio of trading volume to shares outstanding over a given period. It measures how actively a stock is being traded and is often used as a proxy for liquidity and investor attention.

[20] Forward adjusted close prices are historical prices restated to account for dividends and stock splits, so that returns computed from the adjusted series reflect the actual economic returns an investor would have earned rather than mechanical price discontinuities from corporate actions.

[21] The five categories partition stocks by their realized weekly return into discrete bins. The model is trained as a five way classifier on these labels rather than as a regression on the underlying returns, which is why the paper's evaluation centers on F1 A.2 score and cross-entropy loss rather than mean squared error or risk adjusted performance metrics.

short selection strategy over the China-A universe, excluding special treatment and short history stocks.

The most interesting methodological contribution is a two stage feature selection procedure applied before anything enters the neural network. The first stage computes the Pearson correlation between each candidate feature and daily returns for every stock, weighted by market capitalization so that large firms dominate the selection. It also looks for redundancy among features, discarding those with low relevance ($|\rho| < 0.1$) or high mutual correlation ($\rho > 0.8$). The second stage evaluates the remaining features with information gain, where the target is the discretized return direction. Each feature gets a combined score from its Pearson relevance and its information-gain contribution, and the features are ranked accordingly. Starting from 15 technical indicators (including EMA, RSI, Stochastic %K/%D, and Williams %R A.1), this procedure narrows the input set to five features: daily return, turnover, RSI, volume, and forward adjusted close. The authors emphasize that dividend and split adjusted prices are important because they reflect realized economic returns rather than mechanical price discontinuities.

On their test set, the CLGNN model with an MLP output layer achieves an F1A.2 score of approximately 0.65 and a cumulative return of 20.7%, outperforming traditional baselines. The stocks selected by the model cluster in emerging sectors like artificial intelligence and new energy, whereas simpler baselines tend to overweight traditional industries. The authors interpret this as evidence that the hybrid architecture captures cross sectional growth dynamics that simpler models miss.

The evaluation has several limitations that qualify these results. The test window is only 60 days of weekly predictions, roughly three months, which covers a single market regime. A complex CNN-LSTM-GNN is prone to fitting regime specific patterns rather than persistent relationships, and the short window does not allow the authors to test for stability across regime changes. The paper also computes cumulative returns without incorporating transaction costs, slippage, liquidity constraints, or turnover, any of which could erode the reported profits in a real trading setting. The graph neural network learns its cross stock adjacency entirely from data

without any economic structure, which makes the learned relationships hard to interpret and potentially unstable under changing conditions. Finally, the evaluation relies on classification metrics (F1 and cross entropy) rather than risk adjusted return, cost adjusted profitability, or cross regime robustness, which are the metrics that determine whether the strategy is tradable.

2.6 Generative Models for Financial Scenarios

A fifth line of work approaches the forecasting and risk problem from a different angle. Rather than predicting specific returns, these models learn to generate realistic scenarios from which downstream quantities such as portfolio variance, tail risk, and stress outcomes can be estimated.

Chen et al. (2025) proposes a diffusion factor model that integrates an economic factor structure into a diffusion based generative model for high dimensional return simulation. Diffusion models work in two stages: a forward process gradually corrupts the data with noise, and a reverse process learns to map noise back to clean samples. The clean samples are typically parameterized as a sequence of denoising steps trained by score matching. [22]

The reason diffusion models need adaptation for financial data is that the number of assets can be enormous while historical data is limited. A researcher may want to model thousands of stocks from only a few hundred observations, yet still estimate full covariance structures, tail risk, stress scenarios, and joint return distributions. Standard diffusion models treat each asset as an independent coordinate, ignoring the fact that financial returns typically live in a much lower dimensional factor subspace. Such models consequently suffer from the curse of dimensionality in small sample settings.

The diffusion factor model modifies this process by explicitly incorporating a latent factor representation into the score function. It projects high dimensional returns into a low dimensional factor subspace using time varying orthogonal projections, learns the diffusion dynamics

[22] Score matching is a training objective for generative models that learns the gradient of the log density of the data distribution rather than the density itself. By matching this gradient to noise corrupted versions of the data, the model learns to denoise samples. With enough denoising steps, this process can also generate new samples from the underlying distribution.

in this lower dimensional space, and reconstructs full asset returns through a controlled linear factor loading structure.

This reduces the effective dimensionality of the learning problem so that score estimation depends primarily on the intrinsic number of factors rather than the total number of assets. The authors provide non asymptotic statistical guarantees showing that both score estimation error and generated distribution error scale with the factor dimension rather than the asset dimension, which makes the method viable for markets with thousands of assets. Numerical experiments demonstrate strong recovery of the latent factor subspace under small sample conditions, and empirical studies show improvements in constructing mean-variance optimal portfolios and factor portfolios. [23]

The work offers a principled bridge between econometric factor models and generative diffusion models. Two limitations are worth noting. First, the factor structure is assumed to be linear, which may under represent the nonlinear interactions between assets that become especially important during market stress, exactly the regime where scenario simulation is most needed. Second, the evaluation focuses on portfolio construction tasks (mean-variance weights, factor portfolios) and statistical recovery of the latent subspace, rather than on whether the simulated scenarios produce calibrated tail risk estimates. For the risk management application the model is motivated by, calibration of the tails of the simulated distribution is the binding question, and a more direct evaluation through proper scoring rules or backtests of risk metrics like VaRA.3 and expected shortfall A.3 would strengthen the claim.

2.7 FinGAN: Extending ForGAN for Financial Forecasting

Most directly relevant to the present work is the FinGAN framework introduced by Vuletic[33]. FinGAN builds on ForGAN[23], which was the first conditional generative adversarial network (GAN) applied to general time series forecasting. In a conditional GAN, a generator and discriminator compete in the standard adversarial game, but both networks additionally receive

[23] A factor portfolio is a portfolio constructed to have unit exposure to one factor and zero exposure to others, used to isolate the return premium attributable to that factor.

a conditioning input which is a window of past returns. The generator learns to produce samples of the next return given that window, and the discriminator learns to tell real next return samples from generated ones. The output is not a point prediction but an empirical conditional distribution, which is precisely what is needed for distributional forecasting.

ForGAN's [23] generator is trained on Binary Cross Entropy alone. Binary Cross Entropy (BCE) is the standard loss function for binary classification. Given a true label $y \in \{0, 1\}$ and a predicted probability $\hat{p} \in (0, 1)$, the BCE loss is

$$-[y \log \hat{p} + (1 - y) \log(1 - \hat{p})].$$

The loss is small when $\hat{p}$ is close to y and grows large when the prediction is confident but wrong. In the GAN setting, BCE is used to train the discriminator to distinguish real samples from generated ones, with $y = 1$ for real and $y = 0$ for generated. Vuletic [33] observes that this objective has a well known failure mode for financial return data: excess returns cluster near zero, so a generator that consistently outputs near zero values is statistically plausible to the discriminator even though it carries no trading information. Under pure Binary Cross Entropy loss function training, this null signal equilibrium is stable, and the generator never learns to capture directional signals.

FinGAN addresses this by augmenting the generator's objective with economics driven loss terms: the expected profit and loss (PnL) of a sign based trading rule, the Sharpe ratio of that PnL stream, the mean squared error between generated and realized returns, and the standard deviation of PnL. These terms push the generator beyond producing statistically plausible returns. To prevent any single term from dominating the optimization, the loss terms are scaled by coefficients derived from a gradient norm matching procedure that equalizes their gradient magnitudes at initialization. FinGAN then trains the generator on ten different combinations of these losses (nine financial combinations plus a BCE only baseline) and reports the best performing combination by validation Sharpe.

Vuletic[33] also discusses why FinGAN specifically uses a conditional GAN rather than other generative families. Reinforcement learning requires large amounts of stationary data and re-

training is expensive under regime changes, making it costly for typical financial datasets. Variational autoencoders and neural Stochastic Differential Equations (SDEs) are explicit probabilistic models of the data distribution: VAEs assume a Gaussian prior over latents and a parametric decoder, while neural SDEs assume Itô [24] dynamics driven by Brownian motion. These probabilistic priors constrain the marginal distribution the model can represent. By contrast, GANs are implicit models: the generator is a deterministic function of noise, with the discriminator alone enforcing distributional realism, and so they can in principle represent arbitrary distributions including heavy-tailed or multimodal ones. This distinction is preserved in FinTimeGAN despite its addition of an autoencoder: the Embedder and Recovery are deterministic and trained on reconstruction alone, providing only a reversible mapping between input space and a learned latent space; the generative model is still the GAN that operates in this space.

This project builds directly on FinGAN. It reimplements the framework from scratch, extends it with ideas from TimeGAN's [34] latent representation learning, and investigates variations in input representation and architecture. The full description of FinGAN's architecture, loss design, and training algorithm is given in chapters5 and7, where the distinction between the reference implementation and the implementation used in this project is made precise.

[24] One or more terms include an stochastic component

Chapter 3

Project overview

Building on Chapters 1 and 2, this project uses conditional GANs as a tool to generate distributional forecasts of individual stock returns. Using the ForGAN [23] and FinGAN [33] frameworks as a starting point, we extend them using ideas from the TimeGAN[34] framework, specifically incorporating the Embedder and Recovery networks to learn a latent representation of condition window. The extended model, which we call FinTimeGAN, is one of the three models studied in this project.

3.1 Project Scope

This project extends Vuletic's [33] framework in the four ways:

3.1.1 A Modular TensorFlow Implementation of the FinGAN Framework

Vuletic's reference code is written in PyTorch and organized as a research script intended to reproduce the results of a single paper. For this project, the LSTM-Fin and FinGAN baselines have been reimplemented from in TensorFlow and Keras and restructured as a modular Python package with a command-line entry point. A single invocation runs the full pipeline: it selects between single ticker and universal training modes, specifies data and model hyperparameters, and controls plotting, logging, and device placement. Every run produces a self contained output directory containing model weights, plots, per seed evaluation results, aggregated summaries, and a human readable run notes file. The pipeline uses ArcticDB as a persistent time-series cache, an open source database originally developed at Man AHL for institutional time-series

workflows. Using it here avoids repeated API calls to the underlying market-data provider and lets the same data be reused across runs and experiments.

3.1.2 FinTimeGAN, an Architectural Extension of FinGAN

FinGAN operates directly on return space windows. The generator produces next step returns conditioned on past returns, and the discriminator classifies real and generated returns in the same space. FinTimeGAN adds an Embedder and a Recovery network in the style of TimeGAN between the input and the adversarial game, so that the generator instead operates in a learned latent space and the recovered sample is decoded back to return space only at the end. The Embedder and Recovery are pretrained on a reconstruction objective with both full sequence and single position terms. The single position term ensures that each latent position carries enough information to decode its corresponding return in isolation. The adversarial training procedure inherited from FinGAN is then applied in this latent space, with the Embedder and Recovery frozen. The framework supports two ways for the generator and discriminator to consume the latent context: a *compressed* mode in which both networks condition on the final latent vector of the encoded window, and an *attention* mode in which both networks have access to the full latent sequence and use attention to weight contributions across positions. The two modes share the same Embedder, Recovery, and adversarial procedure, and differ only in how the latent representation is consumed.

3.1.3 An Implementation of Independent-branch Loss Training

The FinGAN procedure trains the generator and discriminator on each of ten loss combinations and selects the best-performing combination by Sharpe ratio on the validation set. Algorithm 2 of Vuletic et al.[33] describes these ten combinations as *independent branches* starting from a shared post-calibration checkpoint, but the accompanying reference code chains them sequentially, so the performance attributed to each combination includes the cumulative effect of every combination trained before it. This project implements the independent-branch variant that the

algorithm describes: each combination starts from the same checkpoint, with a fresh optimizer and a reseeded random number generator. The differences in measured performance therefore depend only on each loss combination in isolation.

3.1.4 An Empirical Comparison of Input Return Representations

Vuletic's framework uses the ETF-excess return of each stock as the model input. Subtracting the sector ETF return removes the sector wise systematic movement, leaving the idiosyncratic component. A natural refinement is the *beta-adjusted* excess return, which accounts for the fact that individual stocks exhibit different sensitivities to their benchmark. This project runs the same training pipeline on both representations and compares the resulting forecasts, quantifying whether the additional modeling effort of beta adjustment translates into better directional accuracy.

3.2 Project Outline

The remainder of this document is organized as follows. Chapter 4 describes the data source and the pipeline that turns raw prices into model-ready condition windows. Chapter 5 defines the network architectures that make up LSTM-Fin, FinGAN, and FinTimeGAN. Chapter 7 covers the training procedures, including Phase 1 pretraining for FinTimeGAN and the Phase 2 gradient norm calibration and ten loss-combination branches shared across all three models. Chapter 8 sets out the evaluation protocol: how distributional forecasts are converted into trading positions and which baselines and diagnostics distinguish genuine forecasting skill from market drift, finite-sample noise, and distributional collapse. Chapter 9 presents the results across tickers, input representations, and architectural variants. Chapter 10 concludes with a discussion of limitations and future work.

Chapter 4

Data Pipeline and Description

This chapter dives deeper into how raw equity prices and ETF[1] price data are processed and turned into condition windows before being fed to any model. There are four important concerns about data description that we go through before modeling returns:

1. Why it is better to work with returns rather than raw prices.
2. How returns are constructed from Sharadar's Core US Equity bundle.
3. The two inputs: relative excess return and beta adjusted returns.
4. How the time series are split and windowed for training, validation, and test.

4.1 Why returns and not prices?

Raw price series are unsuitable as model inputs for two reasons. Firstly, raw prices are non-stationary which means their mean and variance drift[2] with underlying trend. For example, a stock that traded at $20 ten years ago may trade at $200 today and so a model that learn the patterns from the $20 period will have no idea what to do with $200 prices. This means the model is not able to generalize well on past data to forecast the future. Second, they are not scale invariant. For example, a one dollar move means very different things for a $10 stock and a $500 stock. Returns, defined as the relative change in price over a time interval, addresses both problems. They are approximately stationary under the standard efficient markets assumption that price changes have no persistent mean, and they are scale invariant because they are

[1] Exchange Traded Funds is an investment fund that holds a basket of different assets like stocks, bonds, or commodities and trades on stock exchange just like an individual stock

[2] In statistical terms, it refers to a systematic change in the mean of time series over time.[2]

dimensionless. Every model in this project therefore operates on returns. We dive into a clear mathematical definition of it in section 4.2.1

4.2 Interleaved Intraday and Overnight Returns

A single trading day actually contains two distinct return intervals with different statistical properties. The *intraday* return runs from the market open to the close of the same day and the *overnight* return runs from the close of one day to the open of the next. Overnight returns reflect information that arrives while the market is closed (earnings releases, overseas price moves, macroeconomic announcements) and tend to be directionally persistent. Intraday returns reflect continuous price formation during market hours and are typically noisier. Collapsing them into a single close-to-close daily return discards this distinction.

Following Vuletic [33], we preserve both by interleaving open and close log prices into a single sequence $[\log O_1, \log C_1, \log O_2, \log C_2, \ldots]$ and taking consecutive differences. The result alternates between intraday and overnight returns, doubling the effective sample size and letting the model learn the distinct dynamics of each half session. Raw returns are then clipped to the range $[-0.15, +0.15]$ to reduce the influence of outliers driven by corporate events[3] that survived the adjustment process.

4.2.1 Log Returns

Log Returns are a specific way of measuring how much an investment's price has changed. While a simple return might tell you how much profit you have made in a day, log return uses logarithm to keep track of that growth over time. The log returns are additive so it makes the math of trading easier. For example, if you want to compute your total profit over a 3 day period from simple returns the formula is

$$(1 + r_1) \times (1 + r_2) \times (1 + r_3), \tag{4.1}$$

[3]Corporate events include stock splits, Dividends, Merger, Spin offs

formally represented as

$$\prod_{i=1}^{3}(1+r_i), \tag{4.2}$$

which can get very messy when you want to calculate over a long period of time. On the other hand, with log returns you can just add them all up. To mathematically represent the log returns, let asset A have price S_t^A at time t, and its log return over time period Δt can be represented as

$$R_{t+\Delta t}^A = \log\left(\frac{S_{t+\Delta t}^A}{S_t^A}\right). \tag{4.3}$$

They are a preferred over simple returns for several reasons. First, they are symmetric. Simple percentages are misleading and hard to measure the gain and loss with. However, with the log returns, if the price goes from \$100 to 50 and back to \$100. The move down is -0.693 and the move up is $+0.693$ which makes a total of 0. The numbers perfectly mirror each other, making it easier to see when you are actually "even". Secondly, they are normally distributed. In finance, the normal distribution would let you predict risk.

Our goal is to forecast the component of a stock's return that is not explained by broader market or sector movement. There are two reasonable ways to remove the part of return that is shared with the sector, and the two produce different residual structures[4]. This section defines both. The relative ETF excess representation is the default used by Vuletic's [33] FinGAN and is the primary input for all three models. The beta adjusted representation is an experimental alternative that we test on a small set of data.

4.2.2 Relative Excess Returns

FinGAN model uses relative ETF excess returns which measure the return of stock A in excess of the return of its relative sector's ETF. The formal definition for relative excess return is as follows. Let stock A belong to sector S, with corresponding sector ETF I^S. The ETF-excess

[4]A residual is what is left after you subtract the model's prediction from the actual value. The residual structures refers to statistical properties of the residual. For example, its distribution, autocorrelation, variance, heavy tails, or whatever the values cluster.

return of stock A over time period Δt is[29]

$$\mathrm{re}^A_{t+\Delta t} = R^A_{t+\Delta t} - R^I_{t+\Delta t}. \tag{4.4}$$

There are two reasons for using relative excess return in this project. First, it will let you remove the systematic risk. The sector ETF captures the common factor driving returns across all stocks in the sector, such as the sector wide market movements, macro shocks, and risky or non risky dynamics. Subtracting this systematic component from the stock return will let you isolate the stock specific residuals,

$$\mathrm{ret}^A_{t+\Delta t} = \underbrace{R^A_{t+\Delta t}}_{\text{total return}} - \underbrace{R^I_{t+\Delta t}}_{\text{systematic}} = \underbrace{\mathrm{ret}^A_{t+\Delta t}}_{\text{idiosyncratic}} \tag{4.5}$$

A model trained on raw returns would need to simultaneously learn sector wise dynamics and stock specific dynamics which is a harder task with more noise. Training on relative excess return centers the model's focus around the idiosyncratic component which can be predicted from stock specific information.

4.2.3 Beta Adjusted Excess Returns

A natural refinement is to let each stock's exposure to its sector vary with its estimated beta. The beta adjusted excess return of asset A against benchmark I^S is

$$\epsilon^A_t = R^A_t - \beta^A_t \cdot R^{I^S}_t, \tag{4.6}$$

where β^A_t is the stock's rolling estimated beta against the benchmark. The formula is the residual of a linear regression of R^A on R^{I^S}, and ϵ^A_t represents the portion of A's return that cannot be explained by its proportional exposure to the sector. When $\beta^A_t = 1$, beta adjustment reduces to simple excess returns; when β^A_t differs from one, the two representations produce meaningfully different residuals.

We estimate β^A_t from a rolling window of 60 observations as

$$\beta^A_t = \frac{\mathrm{Cov}_{60}(R^A, R^{I^S})}{\mathrm{Var}_{60}(R^{I^S})}, \tag{4.7}$$

where the subscript denotes the rolling-window length. The estimated beta is shifted forward by one observation before being applied, so that the beta used at time t is computed only from data up to $t-1$. This prevents look ahead bias which means no information from the future enters the residual at any time. The estimate is also clipped to the range $[-2,+2]$ before application, because rolling beta estimates can become unstable during low volatility periods where $\mathrm{Var}(R^{I^S})$ is small. A short stretch of noisy betas, once applied, can distort the residual series for weeks. Clipping prevents one unusable estimate from contaminating many training windows.

Rather than estimating a single beta on daily returns and applying it to both half sessions, we estimate *separate* betas for intraday and overnight returns:

$$\beta_t^{A,\mathrm{day}} = \frac{\mathrm{Cov}_{60}(R^{A,\mathrm{day}}, R^{I^S,\mathrm{day}})}{\mathrm{Var}_{60}(R^{I^S,\mathrm{day}})}, \qquad \beta_t^{A,\mathrm{night}} = \frac{\mathrm{Cov}_{60}(R^{A,\mathrm{night}}, R^{I^S,\mathrm{night}})}{\mathrm{Var}_{60}(R^{I^S,\mathrm{night}})}. \tag{4.8}$$

Intraday and overnight returns are statistically distinct. Overnight returns include gap dynamics and information accumulation during closed hours, intraday returns reflect continuous price formation. Pooling them into a single beta estimate blurs this distinction. Separate betas recover the half session structure that the interleaving of section 4.2 preserves, at the cost of roughly halving the effective sample count in each beta estimate.

The residual series is rebuilt by interleaving $\{\epsilon_t^{A,\mathrm{day}}\}$ and $\{\epsilon_t^{A,\mathrm{night}}\}$ into the same alternating structure as the simple ETF excess series and clipped to $[-0.15,+0.15]$ for training stability. The 60 day rolling window shortens the usable time range of beta adjusted inputs by roughly 60 observations relative to simple ETF excess inputs because the early windows yield undefined betas. This is an unavoidable cost of the rolling estimation approach and is handled by dropping those observations before window construction.

4.3 Sliding window setup

The models are trained by supervised learning on windows of returns. Each training example is a pair (c,y), where $c \in \mathbb{R}^{L\times 1}$ is a condition window of L consecutive past returns and $y \in \mathbb{R}$ is the next step return. The chapter experiments use $L=10$ for direct comparison with Vuletic's configuration and $L=42$ for the extended context variants introduced with FinTimeGAN.

Windows are built by sliding a fixed length window of $L + 1$ observations along the time series with step size $h = 1$. Every possible window in the series becomes a training example, maximizing the number of samples the model sees. A step of $h = 1$ on interleaved returns means each successive window advances by half a trading day (either open-to-close or close-to-open), which is the finest resolution the interleaving allows.

The time series is split into training, validation, and test sets in the ratio 80/10/10 in strict chronological order before windowing. The training fold contains the earliest data, followed by the validation fold, followed by the test fold. Windowing is then performed independently within each fold; no window crosses a fold boundary. This ordering is critical. Windowing before splitting would produce training windows whose target returns fall inside the validation or test fold, silently leaking future information into training. Splitting before windowing guarantees that every target return used in training predates every return in validation and test, which is the only honest setup for a time series forecasting experiment.

Test set performance under this protocol is a genuine out-of-sample measure. The model saw no information at or after the test window's start date during training.

4.4 Data Source and Adjustments

The underlying price data is the Sharadar Core US Equities bundle which can be accessed through the Nasdaq Data Link API. Sharadar was chosen over the more widely used CRSP[5] database for cost reasons. The two cover the same universe of tickers and return characteristics suitable for this study. Sharadar provides daily open, high, low, close, and volume, together with a dividend and split adjusted close column (*closeadj*). We have gathered the data for all 22 equity prices on all trading days over 25 years of data. The data starts from year 2000 up to the end of 2025, including all trading days, excluding holidays and weekends as no trades heppen then. The pipeline requires both open and close prices adjusted consistently for corporate actions. Because Sharadar's raw *open* is split adjusted but not dividend adjusted, we back out

[5]CRSP is a research ready data that has been well maintained and so some of the discrepancies in results might be explainable by this fact [9]

the dividend factor from the close columns and apply it to the open. Concretely, for each row,

$$f_t = \frac{closeadj_t}{close_t}, \qquad AdjOpen_t = open_t \cdot f_t, \tag{4.1}$$

and $AdjClose_t = closeadj_t$ is used directly. This gives open and close prices that are consistently adjusted for both splits and dividends. The intraday and overnight log returns in Section 4.2 are then computed from these adjusted prices.

The data cutoff for all experiments is $2022 - 01 - 01$. This matches the window used by Vuletic [33] so that our LSTM-Fin and FinGAN baselines are directly comparable to her reported results.

4.4.1 Stock's (Ticker) universe

The ticker universe follows Vuletic [33] and spans 22 stocks across the nine select Sector SPDR[6] ETFs, giving multi stock coverage within each GICS sector.[7]

Each stock is mapped to its sector ETF through a fixed lookup table that is summarized in Table 4.1.

Sector	Sector ETF	Stocks
Consumer Discretionary	XLY	AMZN, HD, NKE
Consumer Staples	XLP	CL, EL, KO, PEP
Energy	XLE	APA, OXY
Financials	XLF	WFC, GS, BLK
Health Care	XLV	PFE, HUM
Industrials	XLI	FDX, GD
Information Technology	XLK	IBM, TER
Materials	XLB	ECL, IP
Utilities	XLU	DTE, WEC

Table 4.1: Ticker universe and sector ETF mapping, following Vuletic [33].

A more systematic ticker selection procedure to consider would be by market capitalization, liquidity, or sector balance. It would be a natural next step to explore and is discussed as future work in Chapter 10.

[6]SPDR or commonly referred to as spider is a brand ETF issued by State Street Global Advisors. This includes 9 sectors and the two excluded sectors from it are Communication and Real State

[7]GICS stands for Global Industry Classification Standard. Its a classification system developed by MSCI and S&P that slots every publicly traded company into one of eleven sectors.

4.4.2 ArcticDB as a persistent cache

To manage the large volume of market data efficiently, we use ArcticDB as a persistent local cache. ArcticDB is a high performance time series database developed by Man Group and open sourced in 2023 [25], with a Python centric API backed by a C++ storage and compression engine. It is designed for exactly the workflow this project requires which allows storing and retrieving large numbers of pandas DataFrames indexed by time.

A flat file cache (for example, one CSV per ticker) would be simpler to set up but introduces several practical issues at the scale of this project. CSV reads are slow on repeated access and so they offer no atomicity guarantees when a training run is interrupted mid write. They do not handle concurrent access gracefully when multiple processes attempt to read the same file. ArcticDB addresses all three issues: reads are fast because data is stored in a columnar binary format; writes are atomic at the symbol level; the underlying LMDB backend supports concurrent readers without locking. For a training pipeline that restarts many times over the course of development, this reliability matters as much as raw speed.

The cache is organized into two libraries: $fingan\_stocks$ for individual equity price series and $fingan\_etfs$ for sector ETF price series. Each symbol stores the full history of adjusted open and close prices for one ticker. On first access, the loader fetches data from Nasdaq Data Link, computes adjusted prices, and writes the resulting DataFrame to the appropriate library and subsequent runs read directly from ArcticDB and skip the API call entirely.

Chapter 5

Model Architecture

This chapter covers the choices that led to the model architecture that was built as part of this project.

5.1 Generative Models

A generative model is any model that takes a training set of samples drawn from an unknown data distribution and learns to represent that distribution. Generative models are good for two main reasons: they can estimate the density of distribution directly, and can generate new samples that look like they came from the training data. Our model of choice is Generative Adversarial Networks which primarily focus on sample generation.

GANs are useful whenever we need more than one single point prediction. Many real world tasks have multiple correct answers for the same input. A given sequence of past stock returns could plausibly be followed by many different future returns depending on market condition. Models like regression, which outputs a single point estimate and is trained on mean squared error (MSE), cannot represent the described type of uncertainty. Generative models address this by learning the full conditional distribution of possible outcomes, from which many different scenarios can be sampled. This is the core reason why generative models are well suited for financial timeseries forecasting. The fact that we can generate a possible distribution of possible returns and derive a trading signal from expected direction of that distribution is making GANs a topic of study for this project.

5.2 Maximum Likelihood (MLE) and Why GANs are better for our task?

The standard way that generative model can learn is through principles of maximum likelihood estimation (MLE). The idea of MLE is straightforward: given a model with parameter θ, MLE finds the θ that makes the training data as probable as possible under the model. Formally, given a training set of m examples $x^{(1)}$, $x^{(2)}$, $x^{(3)}$, ..., $x^{(m)}$ drawn from the true data distribution p_{data}, we define the likelihood as the probability the model assigns to the entire training set,

$$\prod_{i=1}^{m} p_{model}(x^{(i)}; \theta). \tag{5.1}$$

This product is mathematically the right objective but numerically unstable because multiplying many small probabilities together produces values too small for a computer to represent accurately. We instead work with the *log-likelihood*, which converts the product into a sum,

$$\log \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta) = \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}; \theta). \tag{5.2}$$

Sums of the moderately sized numbers are well within a computer's range, so underflow problem disappears. The logarithm is also monotonically increasing function, which means maximizing $\log f(\theta)$ results in the same optimal θ as maximizing $f(\theta)$ itself. Maximizing the log likelihood is therefore equivalent to maximizing the likelihood and so the MLE objective becomes

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}; \theta). \tag{5.3}$$

Intuitively, MLE rewards the model for assigning high probability to the points it has actually seen. Because the density function must integrate to 1 [1], raising probability in one region forces it to drop on other regions. The result is a distribution that is dense where the data is dense, and sparse everywhere else.

MLE has a natural interpretation in terms of distribution distance. Maximizing the log likelihood over training samples is equivalent to minimizing the KL divergence[2] between empirical

[1] The model's total probability is fixed at 1.

[2] Kullback-Leibler divergence is a way to measure how much one probability distribution differs from another or from a "reference distribution".

data distribution and the model,

$$\theta^* = \arg\min_{\theta} D_{KL}(\hat{p}_{data}(x) \| p_{model}(x;\theta)), \tag{5.4}$$

where,

$$D_{KL}(p\|q) = \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right]. \tag{5.5}$$

KL divergence quantity is always non-negative and equals zero only when q and q are identical everywhere. Minimizing the divergence rewards the model to assign high probability to the regions where real data actually appears. The penalty is asymmetric: assigning low probability to a point where data actually appears incurs a large loss, while assigning probability to the regions where data is absent is penalized only mildly. This is what makes MLE a *mode covering* [3] objective. The model is forced to explaining every training point, even at the cost of spreading probability mass into empty regions.

To use MLE, a model must be able to evaluate $p_{model}(x;\theta)$ for any specific point x. This requires explicitly writing down a density function, which can be computationally difficult or even impossible for complex, high dimensional data. We choose GANs because they sidestep this requirement entirely. A GAN generator is simply a function that transforms a noise vector $Z \sim \mathcal{N}(0, I)$ into a sample. You can draw from it, but you can never ask "What is the probability of this specific point?" This distribution exists only implicitly through the sampling process, it is never written down. In GoodFellow's taxonomy of generative models, GANs sit at the far end of implicit density models that generate samples in a single step without ever defining an explicit probability distribution[15].

This distinction matters directly for the models developed in this project. The adversarial components of both FinGAN and FinTimeGAN are implicit density models in which generator produces simulated excess returns without ever defining their density. However, both models selectively reinforce MLE where is it beneficial. The MSE term in both models, discussed further

[3] A *mode* of a distribution is a region of high probability, typically a peak. A mode-covering objective is one that forces the model to assign at least some probability to every mode of the data distribution, even if it has to assign extra probability to empty regions in between. The opposite is a mode-seeking objective, which lets the model latch onto a single mode and ignore the others.

in 6, is MLE in disguise. Specifically, we assume

$$x_{t+\Delta t} = G(a_t, Z; \theta_g) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{5.6}$$

which is the Gaussian forecast error assumption. The probability density function of a Gaussian has the form $\exp(-(x-\mu)^2/2\sigma^2)$ up to a normalization constant. Taking the log cancels the exponential and leaves

$$\log p(x_{t+\Delta t}) \propto -\frac{(x_{t+\Delta t} - G(a_t, Z; \theta_g))^2}{2\sigma^2}, \tag{5.7}$$

which is exactly the negative MSE up to a constant. Maximizing the log likelihood is therefore equivalent to minimizing the MSE. This bridges probabilistic modeling and gradient based optimization.

Similarly, the Embedder and Recovery pre-training phase in FinTimeGAN minimizes a reconstruction loss that is identically MLE under the same Gaussian assumption. Both models combine the two paradigms; implicit adversarial learning captures the rich conditional distribution of returns that MLE cannot represent while explicit MLE term keep the generated return magnitudes closer to real values.

5.3 GANs Framework

Generative Adversarial Networks were first introduced by Ian Goodfellow et al. [15] as a framework for training generative models by setting two neural networks work together. The first network, the generator, learns to produce synthetic samples that resemble the training data. The second network, the discriminator, learns to distinguish true data from output of the generator. The following picture provides an intuitive overview of how GAN's networks work on a image generation task.

The discriminator is trained using standard supervised learning because it receives labeled inputs marked as real or fake and learns to classify them correctly. The generator is trained to fool the discriminator. It has no direct access to training data and so all the learning signals comes from discriminator's feedback. This is often described as a game between a crook and a detective. The crook tries to produce fakes that are indistinguishable from genuine currency

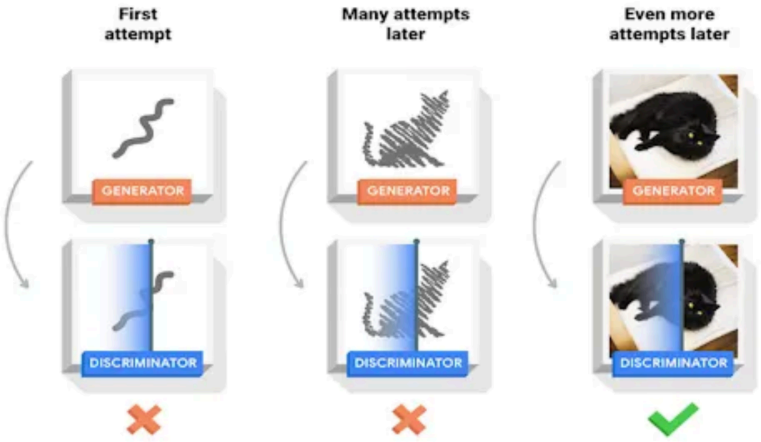


Figure 5.1: Both networks achieve high levels of realism together [1]

while the detective tries to catch them. As the training goes on, the generator is forced to produce increasingly realistic samples in order to keep fooling the increasingly skilled discriminator.

The discriminator's objective is t assign a high probability to real samples and a low probability to generated ones, which is a standard binary classification task. The generator's objective is to make the discriminator assign high probability to its output which is to generate samples that discriminator can't distinguish from real ones. Both networks are trained with the binary cross entropy *(BCE)* loss, but on different terms. For a discriminator D producing $D(x) \in (0, 1)$ interpreted as the probability that x is real, the full BCE loss is

$$\mathcal{L}_{\text{BCE}}(D, G) = -\mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]. \tag{5.1}$$

The discriminator minimizes the full expression, which pushes $D(x)$ toward 1 on real samples and $D(G(z))$ toward 0 on generated samples. The generator only sees the second term, since the first does not involve G, and it wants the opposite outcome: it pushes $D(G(z))$ toward 1, which is equivalent to minimizing $-\mathbb{E}_z[\log D(G(z))]$. We refer to this generator side BCE component as $\mathcal{L}_{BCE}$ in the generator's loss.

Goodfellow et al.[15] proves that, in theory, this game has a unique optimum. When the generator's distribution exactly matches the real data distribution, the best the discriminator can do is guess: it cannot find any property that systematically distinguishes real from generated

samples, so it assigns roughly equal probability to both. The optimum is therefore a Nash equilibrium where the generator has reproduced the data distribution and the discriminator cannot do better than chance. In practice this exact equilibrium is rarely reached, but the training dynamics push the generator toward it.

A well known challenge in GAN training is mode collapse, where the generator converges to producing a narrow or nearly constant output regardless of the noise input. In financial forecasting this is particularly damaging because a collapsed generator predicts exactly the same return everyday and provides no useful trading signal. This is the motivation behind why FinGAN's economic driven loss function, which is designed to prevent this collapse and is described in chapter 6.

5.4 Conditional GANs (CGAN)

The main contribution of this project is through use of Conditional Generative Adversarial Networks (cGANs) which were introduced by Mirza and Osindero in 2014[27]. This family of models were introduced as an alternative framework for training generative models by bypassing the difficulty of approximating many intractable probabilistic computations. Compared to other generative model families, this adversarial framework has several practical advantages. It does not require the Markov Chains and only backpropagation is used to obtain the gradients with no inference during the learning. In addition, it enables the incorporation of a wide variety of factor and interactions into the model, and it can produce state of the art log-likelihood estimates alongside of realistic samples.

The core contribution for conditional models is the one-to-many mapping problems. As discussed earlier, some tasks have multiple correct outputs for a single input. For example, the same sequence of past returns can lead to many different future returns. The standard supervised learning models, like regression trained with MSE loss, learn one-to-one mapping and collapse to an average output which causes a loss of information about the diversity of possible outcomes.

A conditional GAN learns the full distribution over a plausible outputs for each specific input which is exactly what financial scenario generation needs.

To understand why a conditional version of GANs is necessary, it helps to consider what a standard unconditional GAN is missing. In an unconditional GAN, there is no way of controlling the mode of data generated and so the generator learns to produce samples like training data in a general sense. On the other hand, in a conditional GAN both generator and discriminator can have additional context information y as input which makes it possible to directly control the generated data. The condition y can be any form of auxiliary information like class labels, data from another model, or, as in this project, a time series of past historical observations. This formally changes the objective function to

$$\min_G \max_D V(D,G) = \mathbb{E}[\log D(x|y)] + \mathbb{E}[\log(1 - D(G(z|y)))], \tag{5.1}$$

where both generator and discriminator are now conditioned on y[27].

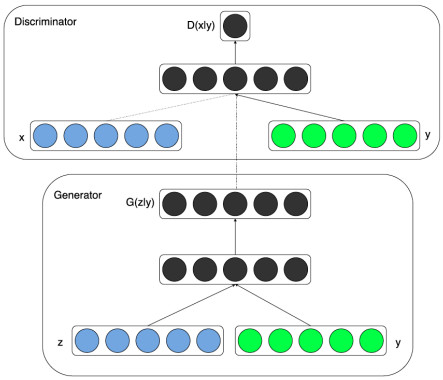


Figure 5.1: The structure of a simple conditional adversarial net, with the condition y flowing into both the generator and the discriminator alongside their other inputs [27].

In practice, it is implemented by feeding y as an additional input to both networks as illustrated in figure 5.1. In the generator the condition y and the noise vector Z is both combined to a joint hidden representations. In the discriminator, both sample X and condition y are presented as inputs and so the discriminator learns not just whether as a sample looks realistic in general but whether it is consistent with the specific condition it was generated from.

In this project, the condition a_t is a sliding window of the most recent ETF-excess returns of a given stock and the target is the next day excess return. The generator takes both a noise vector $Z \sim \mathcal{N}(0, I)$ and this condition as input, producing a simulated next day return $G(a_t, Z; \theta_g)$ that is consistent with recent market history. The discriminator receives either a real pair $(a_t, x_{t+\Delta t})$ from the data or a fake pair $(a_t, G(a_t, Z))$ from the generator, and assesses whether the sample is plausible given the condition. This conditional structure is what transforms the GAN from a data simulator into a forecasting tool, and it is the foundation on which both FinGAN and FinTimeGAN models are built.

5.5 ForGAN Architecture

FinGAN is built on the ForGAN architecture. The ForGAN architecture was introduced by Koochali et al.[23] as the first attempt to apply a GAN directly to the time series forecasting problem. The problem they identified with existing approaches was fundamental: most forecasting methods collapse a distribution of possible futures into a single number. This is fine if you only care about the average outcome, but it throws away everything about uncertainty, tail risk, and the range of what could plausibly happen next. ForGAN addresses this by framing forecasting as a distribution learning problem. Given a window of past observations $c = \{x_o, \ldots, x_t\}$, the goal is to learn the full conditional distribution $P(x_{t+1}|c)$, from which any number of future scenarios can be drawn by sampling[23].

The overall setup follows the conditional GAN framework of section5.4 directly, with the historical window c taking the role of the condition y. Both networks receive c as context. The generator takes this window alongside a noise vector drawn from a standard Gaussian distribution and produces a candidate next value. The discriminator receives either a real next value from the data or the generator's output, always paired with the same historical window, and decides which one it is. Through repeated adversarial training, the generator is pushed toward producing outputs whose distribution matches the true conditional distribution

of the data. The training objective is therefore the standard conditional GAN minimax game of equation 5.1, with $y = c$ and $x = x_{t+1}$. Figure **??** shows the overall architecture.

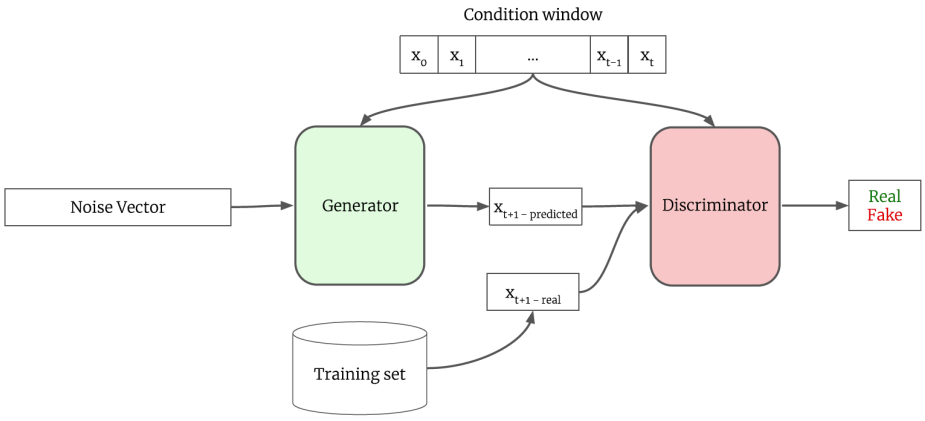


Figure 5.1: The ForGAN architecture, with the conditioning window c passed to both the generator G and the discriminator D [23].

ForGAN's contribution is therefore not a new architecture or a new objective. It is the observation that the conditional GAN framework, already in use for image generation, can be applied to forecasting tasks where the input is a sequence and the output is its next value. This framing is what FinGAN inherits and extends with the economics driven loss terms described in chapter 6.

5.6 FinGAN Architecture

[4] FinGAN, introduced by Vuletic et al. [33], applies the ForGAN framework to financial return forecasting. The generator and the discriminator architectures are inherited from ForGAN unchanged. Both networks uses a single recurrent layer with LSTM cells [5] to process the condition window of the past returns, followed by a dense layer.

All hidden dimensions and the noise dimension are set to $d_h = d_z = 8$. This is a deliberate decision given that individual stock datasets contain only around $10,000$ training observations,

[4] The accompanying code for FinGAN's re-implementation can be found at the following link: `https://github.com/RoseRahimi/FinGAN-Reimplementation-in-TF`

[5] LSTMs are gated recurrent cells designed to learn long range dependencies in sequential data. They are described in details in section 5.7.5

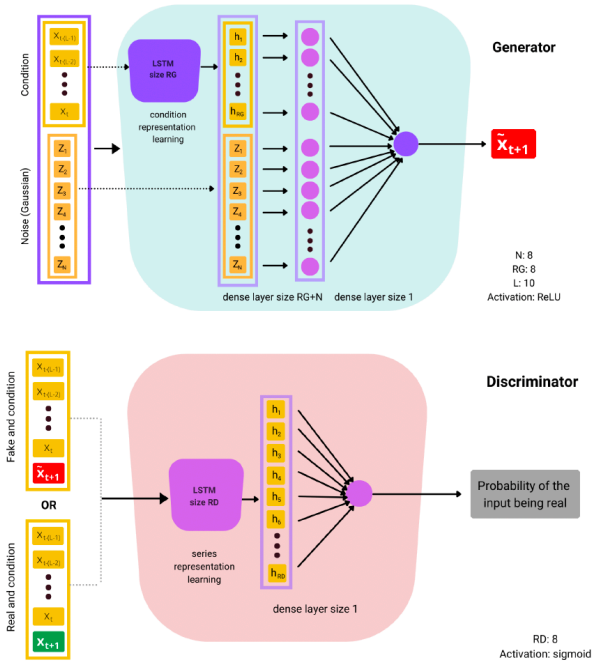


Figure 5.1: The FinGAN architecture with LSTM based Generator and Discriminator processing the conditioning window of past excess returns. [33]

and a larger network would be at risk of overfitting. The generator uses ReLU [6] activations in its hidden layers and no activation on the output, since the output is a real valued return. The discriminator uses ReLU in the hidden layers and a sigmoid [7] on the output, since the output is the probability that the input pair was real.

FinGAN's contribution relative to ForGAN is therefore not architectural. The networks, the conditioning structure, and the adversarial framework are all inherited unchanged. What FinGAN adds is the extension of ForGAN's generator loss with four economics driven loss terms: Profit and Loss 6.2, Mean Squared Error 6.3, Sharpe ratio 6.4, and Standard Deviation of PnL series 6.5. The Binary Cross Entropy component that drives ForGAN remains in FinGAN's generator loss as $\mathcal{L}_{BCE}$, alongside these new terms rather than replaced by them. The full

[6] The Rectified Linear Unit (ReLU) is defined as $\text{ReLU}(x) = \max(0, x)$. It outputs the input unchanged if positive and zero otherwise, which makes it computationally cheap and keeps gradients from vanishing on positive inputs.

[7] The sigmoid function $\sigma(x) = 1/(1+e^{-x})$ maps any real number into the range $(0, 1)$, which makes it a natural choice for outputs that should be interpreted as probabilities.

combined loss is described in Chapter 6, and is the substantive contribution of Vuletic et al. [33].

5.7 FinTimeGAN

[8] The FinGAN model described in the previous section uses a condition window of $L = 10$ half day returns, which corresponds to one trading week. While this is computationally efficient, it limits how much historical context the generator can draw on when forming its forecast. Financial return series contain patterns that operate over longer horizons. Momentum effects, earnings cycles, and volatility regimes often play out over weeks or months rather than days, and a model that only looks back five trading days is blind to all of this.

FinGAN's generator and discriminator each contain only a single LSTM layer, and that single layer is responsible for both learning the temporal dynamics of the condition window and supporting the adversarial game with the other network. This works at $L = 10$, where the dynamics are short enough to fit into the recurrent state, but it results in capturing more noise than real signal as L grows. A single LSTM cannot reliably learn the dynamics of a longer window while simultaneously being pushed by the adversarial signal in directions that have nothing to do with those dynamics. As a consequence, FinGAN's architecture cannot safely treat L as a free hyperparameter, even though the longer horizon patterns the model is trying to capture would benefit from a larger window.

FinTimeGAN addresses this by separating the two concerns. An Embedder network is responsible for learning the temporal dynamics of the condition window over an arbitrarily long horizon, and the adversarial game is played in the latent space the Embedder produces, where the generator and discriminator no longer have to learn temporal structure themselves. With this separation in place, L becomes a genuine hyperparameter rather than a fixed architectural commitment. Most of our experiments use $L = 42$, corresponding to roughly one month of half-day trading, but the architecture is designed to handle arbitrary L so that we can capture

[8] The accompanying code can be found at the following link: `https://github.com/RoseRahimi/FinTimeGAN-Pipeline`

market shifts, regime changes, and volatility clustering at the timescales where they actually occur.

This project's proposed solution is inspired by TimeGAN, introduced by Yoon et al. in 2019 [34]. TimeGAN showed that pre-training an embedding network before adversarial training improves quality and stabilizes time series generation. The key observation is to separate the problem into two stages. First, an Embedder network learns the temporal dynamics of the data over the long horizon and produces a compact latent representation. Second, an adversarial network learns to generate plausible next step values in the space the Embedder has prepared, without having to handle the temporal dynamics directly. We call the resulting model *FinTimeGAN*, and Sections 5.7.1 through 5.7.5 describe each component in turn.

One specific choice deserves a note. TimeGAN's full training procedure includes a supervised loss that penalizes the generator for producing temporal transitions that differ from real data. This is appropriate when the generator is producing a full sequence rather than a single next-step value. In our forecasting setting, the generator produces only one step ahead, conditioned on a latent summary of the recent past that the Embedder has already provided. The supervised loss would be redundant in this context, and we omit it from FinTimeGAN's training procedure (chapter 7).

5.7.1 Embedder and Recovery

The Embedder and Recovery networks form an autoencoder pair on financial time series data. The Embedder maps the raw return condition window into a richer latent representation that captures temporal dynamics over the long horizon. The Recovery network maps the latent sequence back to the original return space.

The Embedder receives the condition window $a_t \in \mathbb{R}^{L\times 1}$ and produces a latent sequence $H \in \mathbb{R}^{L\times d_h}$, where d_h is the hidden dimension. The input is first normalized using the reference batch mean and standard deviation introduced in chapter 7. The normalized sequence is then processed by a stack of recurrent layers, configurable as either LSTM or GRU cells, with the

number of layers also exposed as a hyperparameter. The recurrent stack returns a hidden state at every timestep, producing a tensor of shape (batch, L, d_h). A dense layer with a tanh activation[9] is applied at every position. The activation keeps the latent representation in the range $(-1, 1)$ and prevents it from drifting to arbitrarily large values during training.

The Recovery network is the inverse of the Embedder. It maps the latent sequence back to return space using a single recurrent layer (matching the Embedder's cell type) followed by a dense layer with no activation. The dense layer has no activation because the output must be able to take any real value before denormalization, and an activation function would clip the range of reconstructible values. The denormalization multiplies the output by the reference standard deviation and adds the reference mean, restoring the sequence to its original return space.

The two networks are trained jointly to minimize a reconstruction loss that combines a sequence level term and a single position term,

$$\mathcal{L}_R(\theta_e, \theta_r) = \mathbb{E}\left[\frac{1}{L}\sum_{s=1}^{L}\big([R(E(a_t))]_s - a_{t,s}\big)^2 \;+\; \frac{1}{L}\sum_{s=1}^{L}\big(R([H_s]_{\text{iso}}) - a_{t,s}\big)^2\right], \tag{5.1}$$

where $[H_s]_{\text{iso}}$ denotes the sth latent position reshaped into a length 1 sequence and decoded in isolation from the rest of H. The first term is an ordinary sequence level reconstruction. The second term is the key to making FinTimeGAN work. Without it, the Embedder would be free to distribute information about $a_{t,s}$ across other positions in H, as long as the Recovery network could unscramble it when given the whole sequence. That freedom would be a problem during adversarial training, because the generator only ever produces a single next step latent and the discriminator must be able to evaluate it on its own. The single position term forces every latent to carry enough information to decode its corresponding return on its own. As discussed in section **??**, the squared error reconstruction loss is equivalent to MLE under a Gaussian assumption on reconstruction errors.[18]

[9] See Appendix A.2.

5.7.2 Generator and Discriminator with Last State Context

The generator and discriminator can use the Embedder's latent sequence in two ways, controlled by a hyperparameter `context_mode` that takes the value `last` or `attention`. We describe the simpler last state mode first and the attention extension in Section 5.7.4.

Generator.

In last state mode, the generator takes a noise vector $Z \sim \mathcal{N}(0, I)$ of dimension d_z and the final position of the latent sequence, $h_{\text{last}} = H[:, -1, :]$, which summarizes the entire condition window. The two are concatenated into a single vector of dimension $d_z + d_h$ and passed through two dense layers. The first uses a ReLU activation[10] and produces an intermediate vector of dimension d_h. The second uses a tanh activation and produces the next latent vector H_{fake} of dimension d_h. The tanh activation matches the range of the Embedder's latent representations, which is also bounded in $(-1, 1)$, so that H_{fake} lives in the same space as the real latents the Recovery network was trained to decode.

We chose two dense layers in the generator rather than another recurrent layer for a specific reason. The Embedder has already processed the temporal dynamics of the full condition window into a compact latent vector, so a recurrent layer in the generator would be redundant. We confirmed this in early experiments. A generator that re-encoded the tanh saturated latent sequence with its own RNN suffered from mode collapse and vanishing gradients. Two dense layers are sufficient to generate the next latent vector and are more stable to train.

Discriminator.

The discriminator operates in real return space rather than in the latent space. The economics driven loss function described in Chapter 6 is only meaningful in real return space, so keeping the discriminator there ensures that the adversarial signal and the economics driven signal operate on the same quantity. For a real pair, the discriminator receives the L step condition window and the real next return from the data. For a fake pair, it receives the same condition window and

[10] See Appendix A.2.

the candidate next return generated by the generator, after the Recovery network has decoded the generator's latent output back to return space.

The discriminator normalizes the condition window and passes it through a stack of recurrent layers matching the Embedder's depth and cell type, producing a hidden state h_n of shape (batch, d_h). Matching the Embedder's depth ensures that the discriminator processes the historical context at the same capacity the Embedder used to produce it, so the two networks compare the candidate return against a representation built with the same recurrent inductive bias. The candidate return is a single value in $\mathbb{R}$ and cannot be passed through a recurrent layer directly, so it is projected into the latent space by a dense layer with tanh activation, producing a vector of dimension d_h. This projection plays a role analogous to the Embedder for the next step return: it lifts a single point into a representation comparable in dimensionality to h_n. The two vectors are concatenated to form a combined vector of dimension $2d_h$, which is processed through two dense layers. The first uses a ReLU activation. The second uses a sigmoid activation[11] to output the probability that the input pair was real.

The generator and discriminator have matched depth and cell types so that they process information at comparable capacities, and the adversarial signal remains balanced.

5.7.3 Information Bottleneck at Long Sequences

After training the last state model on multiple tickers, we observed a clear pattern. At $L = 10$, FinTimeGAN performed comparably to FinGAN. As L was increased toward 42, performance did not improve as expected, and on some tickers it degraded. The cause is structural. In the last state architecture, the only information the generator and discriminator receive about the condition window is the final hidden state h_{last}. This single vector is a summary that the Embedder must compress everything it has learned into. As L grows, the recurrent layer's hidden state at the final timestep tends to forget patterns that occurred earlier in the sequence, even when those patterns carry information relevant to the next return. The motivation for

[11] See Appendix A.2.

keeping a long condition window in the first place was to capture longer range patterns, and the last state pathway throws much of that signal away.

This is a familiar problem from the natural language processing literature. Sequence to sequence models built on RNNs faced the same bottleneck when the input sequence grew longer, and the solution was attention. Vaswani et al. [6] introduced the multi head attention mechanism that lets a model query the entire input sequence directly rather than relying on a single summary vector. We adapt the same mechanism here to operate on the Embedder's latent sequence, which removes the bottleneck and lets the generator and discriminator draw on the full L step context.

5.7.4 Attention-Based Context Aggregation

When `context_mode` is set to `attention`, the generator and discriminator receive the full latent sequence $H \in \mathbb{R}^{L \times d_h}$ rather than just h_{last}. Both networks use multi head attention with a configurable number of heads (default 2) to summarize H into a context vector that depends on what each network needs at that moment.

Multi headed Attention

The mechanism follows Vaswani et al. [6]. Given a query vector $q \in \mathbb{R}^{d_h}$ and a sequence H that serves as both keys and values, scaled dot product attention computes

$$\text{Attention}(q, H) = \text{softmax}\left(\frac{qK^\top}{\sqrt{d_k}}\right) V, \tag{5.2}$$

where $K = HW_K$ and $V = HW_V$ are learned linear projections of the latent sequence and d_k is the dimension per head. The softmax produces a weight for every position in H, and the result is a weighted average of the value projections. Multi head attention runs this in parallel across h heads, each with its own learned W_K and W_V, and concatenates the per head results before a final linear projection. The intuition is that different heads can specialize in different aspects of the latent sequence: one head might attend to recent positions while another tracks longer range trends.

The dimension per head is set to $d_k = d_h/h$, so that the total dimensionality across heads matches d_h and the output projection produces a d_h dimensional context vector. This means the choice of head count constrains the choice of hidden dimension. With $h = 2$ heads we use $d_h = 32$, giving each head 16 dimensional key and value projections, which is a reasonable per head capacity for the noise levels typical of financial returns. We default to $d_h = d_z = 32$ in attention mode rather than the $d_h = d_z = 8$ used in last state mode, because the residual connection introduced below requires that the attention output match the dimension of the query, and this is enforced by the choice $d_z = d_h$.

No Positional Encoding.

The original transformer architecture requires an explicit positional encoding to be added to the input, because the self attention operation is permutation invariant and would otherwise be unable to distinguish the order of tokens in the sequence. In our setting, the positional information is already baked into H. The Embedder is a recurrent network that processes the input sequentially, and the latent at position s depends on positions $1, 2, \ldots, s$ of the input. The latent sequence inherits the temporal ordering of the condition window through the recurrence itself, which removes the need for any explicit positional encoding when the attention mechanism reads H.

Query Construction.

The query that the attention mechanism uses to read H matters more than the attention mechanism itself, because the query is what determines which positions of H get high weight. We construct the query from a deterministic combination of the latent sequence's own statistics. For the generator, the query seed is

$$q_{\text{gen}}^{(0)} = [h_{\text{last}}, \text{mean}(H)], \tag{5.3}$$

where $\text{mean}(H)$ is the element wise mean of the latent sequence across the time axis. The seed is then projected through a dense layer with tanh activation to produce the query $q_{\text{gen}} \in \mathbb{R}^{d_h}$. The combination of h_{last} and $\text{mean}(H)$ gives the query access to two complementary kinds of

information. The last hidden state captures recent conditions: what the most recent half day returns looked like and how the recurrent state has evolved over the most recent positions. The mean pool captures the average regime over the entire window: the typical activation pattern when the market is in a particular volatility or momentum state. Either alone would be a less informative query. The last state alone would bias the attention toward recent positions and reproduce the same bottleneck that motivated attention in the first place. The mean pool alone would lose recency entirely and treat the most recent return as no different from a return L steps ago. Together, the two anchor the query in both recent activity and average regime, which gives the attention mechanism enough context to decide which positions of H are most relevant to the next return.

For the discriminator, the query also includes the candidate return projected into latent space,

$$q_{\text{disc}}^{(0)} = [x_{\text{embed}}, h_{\text{last}}, \text{mean}(H)], \tag{5.4}$$

where x_{embed} is the candidate return after passing through the same return projection layer used in last state mode. Including x_{embed} in the query lets the discriminator tailor its attention pattern to what is being scored. If the candidate return represents a large positive move, the discriminator can attend to historical positions where similar large moves preceded continuation or reversal, and use that context to judge whether the candidate is plausible. The discriminator's query seed is also projected through a dense layer with tanh activation to produce $q_{\text{disc}} \in \mathbb{R}^{d_h}$.

It is worth noting that we deliberately do not include the noise vector Z in the generator's query. The query is meant to determine *which positions of the condition window are relevant*, which is a deterministic function of the condition itself. The noise is meant to determine *which plausible next return to draw*, conditioned on the relevant information that attention has surfaced. Mixing the two would couple the attention pattern to the noise draw, which would break the interpretation of attention as a deterministic context selection step. The noise enters only at the final dense layer stage of the generator, after the attention has produced the context vector.

Residual Connection and Layer Normalization.

The attention output is combined with the query through a residual connection followed by layer normalization,

$$\tilde{c} = \text{LayerNorm}(q + \text{Attention}(q, H)). \tag{5.5}$$

The residual connection adds the query directly to the attention output, which gives the network the option of falling back to the query itself if the attention pathway is not yet useful early in training. This is the same residual pattern used in transformer encoder blocks, and it improves training stability when the attention mechanism is being trained from scratch. The residual is the reason that the dimension of the query and the dimension of the attention output must match, which in turn requires $d_z = d_h$ in attention mode.

Layer normalization normalizes the activations across the feature dimension at each position independently, with learned scale and shift parameters. It serves two purposes here. First, the residual connection $q + \text{Attention}(q, H)$ produces activations whose scale depends on the magnitudes of both the query and the attention output, which can vary substantially during training. Layer normalization removes this scale variation so that the downstream dense layers receive activations of consistent magnitude. Second, layer normalization improves gradient flow through the attention block, which matters because the attention pathway is deeper than the simple h_{last} pathway it replaces. We do not apply layer normalization in last state mode, because the simpler pathway has fewer transformations and the activation scales remain stable without it.

Final Pathway.

After the residual and norm block, the resulting context vector is what the rest of the network operates on, in place of h_{last} in last state mode. For the generator, the context vector is concatenated with the noise vector and passed through the same two dense layers (ReLU then tanh) that the last state generator uses. For the discriminator, the context vector is concatenated with x_{embed} and passed through the same two dense layers (ReLU then sigmoid) that the last state

discriminator uses. The downstream architecture is therefore identical between the two modes. The only difference is whether the context summary is the final hidden state of the Embedder or the result of attending over the full latent sequence with a query built from the same. You can see the full architecture diagram below:

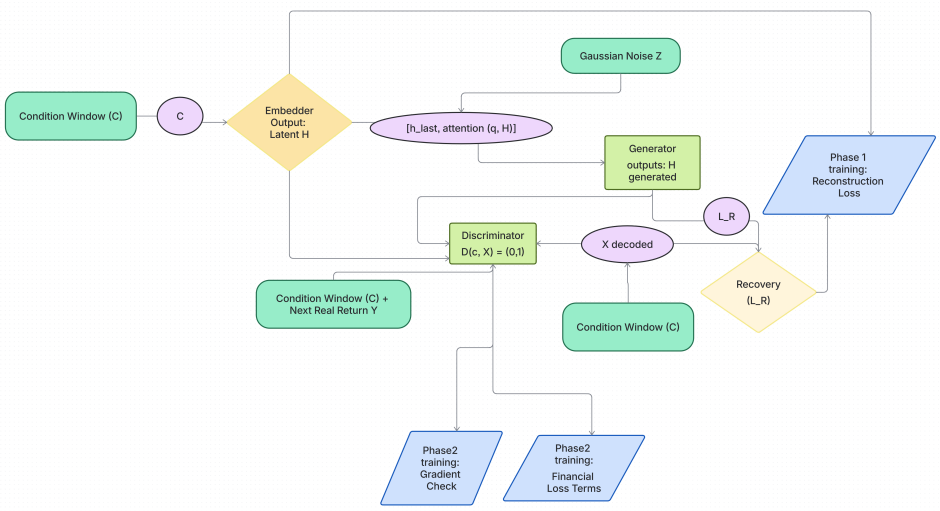


Figure 5.1: The FinTimeGAN model Architecture

5.7.5 Cell Types: LSTM and GRU

The original TimeGAN and ForGAN models treat the cell type as a hyperparameter to be selected by dataset rather than a fixed architectural choice. Empirical results from both papers show that the optimal cell type varies. In the ForGAN work, the researchers found that the GRU cells worked better on the Lorenzo and Internet Traffic datasets, while LSTM was preferred on Mackey Glass. There is no universally optimal cell type, and the right choice depends on the temporal structure of the specific data being modeled.

For financial return time series, the choice is more subtle. The data is noisy, has fat tails, and contains relevant temporal structure operating on different timescales simultaneously. Both LSTM and GRU belong to the family of gated recurrent units, designed to address the gradient vanishing problem that affects standard RNNs on long sequences. We describe each below before discussing the choice for FinTimeGAN.

LSTM.

The Long Short Term Memory cell was introduced by Hochreiter and Schmidhuber in 1997 [17] to solve the vanishing gradient problem that prevented standard RNNs from learning long range dependencies. An LSTM cell maintains two separate state vectors. The cell state c_t acts as long term memory and runs through the sequence with controlled modifications at each time step, which allows gradients to flow backward through time without shrinking to zero. The hidden state h_t is the cell's output at each timestep, and is what downstream layers see.

In TensorFlow's Keras API, an LSTM layer exposes both states through its `return_state` argument. When `return_state=True`, the layer returns a tuple of $(\text{outputs}, h_T, c_T)$, where the outputs are the per step hidden states and h_T and c_T are the final hidden state and cell state respectively. Our Embedder uses the default `return_sequences=True, return_state=False` configuration, which returns the full sequence of hidden states $\{h_t\}_{t=1}^{L}$ and discards the final cell state. This is the correct choice for our setting because the latent representation the Embedder is constructing lives in hidden state space, not cell state space. The cell state is an internal regulatory mechanism whose purpose is to let the hidden states carry long range information; the hidden states are what the downstream networks consume.

The cell state is regulated by three gates.[16][12] The forget gate decides what to discard from the previous cell state,

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f). \tag{5.6}$$

The input gate decides what new information to write into the cell state,

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{5.7}$$

along with a candidate update value,

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c). \tag{5.8}$$

[12] PP. 138-152

The cell state is then updated by forgetting part of the old cell state and adding part of the new candidate,

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t. \tag{5.9}$$

The output gate controls what portion of the updated cell state is exposed as the hidden output,

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \qquad h_t = o_t \odot \tanh(c_t). \tag{5.10}$$

In the above equations, σ is the sigmoid function, $\odot$ denotes element wise multiplication, and W, U, b are learned weight matrices and bias vectors. For the Embedder, the sequence of hidden states $\{h_t\}$ forms the latent representation of the condition window. The key property

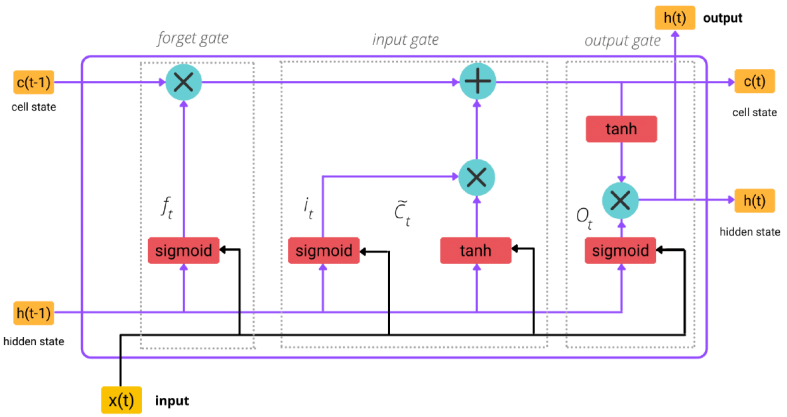


Figure 5.2: Illustration of an LSTM cell showing the three gates and the cell state [33].

that makes the LSTM useful for financial data is the additive cell state update in Equation 5.9. Because the update adds rather than multiplies, gradients can propagate backward through many time steps without vanishing, which lets the model learn which events from L time steps ago are still relevant to today's return forecast.

Gated Recurrent Unit.

The Gated Recurrent Unit (GRU) was introduced by Cho et al. in 2014[19] as a simplification of the LSTM. Rather than maintaining separate cell and hidden states, the GRU merges them into a single hidden state h_t and uses two gates instead of three.

The reset gate controls how much of the previous hidden state to use when computing the candidate new state, [16][13]

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r). \tag{5.11}$$

The update gate controls how much of the previous hidden state to carry forward versus how much of the new candidate to incorporate,

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z). \tag{5.12}$$

A candidate hidden state is computed using the reset gate to selectively suppress parts of the previous state,

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h). \tag{5.13}$$

The final hidden state interpolates between the previous state and the candidate,

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t. \tag{5.14}$$

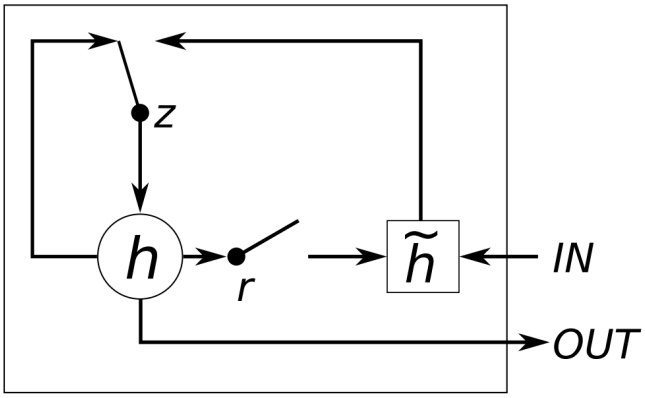


Figure 5.3: Illustration of a GRU cell showing the reset and update gates [19].

Figure 5.3 illustrates the GRU's two gate structure alongside the LSTM cell shown in figure 5.2. Comparing the two cells, the GRU's update gate plays a combined role similar to the LSTM's forget and input gates. When z_t is close to 1, the new candidate largely replaces the old state. When z_t is close to 0, the previous state is mostly preserved. The reset gate gives the

[13]PP. 138-152

model the ability to suppress the previous hidden state when computing the candidate. When $r_t \approx 1$, the previous state is fully used, and the GRU behaves like a continuous update of memory. When $r_t \approx 0$, the previous state is effectively ignored, and the candidate is computed as if from a fresh start. This matters when recent context becomes a poor basis for the next prediction. In financial return data, the most common case is a regime change, where a sudden shift in volatility or direction means that the last several time steps no longer carry useful information about what comes next.

Parameter Count.

Both LSTM and GRU cells are built from gates with the same parameter structure: a weight matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_h}$ connecting the input to the gate, a weight matrix $U \in \mathbb{R}^{d_h \times d_h}$ connecting the previous hidden state to the gate, and a bias vector $b \in \mathbb{R}^{d_h}$. Each gate therefore contributes $d_h \cdot d_{\text{in}} + d_h^2 + d_h$ parameters. The difference between the two cells comes entirely from how many gates each uses. For hidden dimension $d_h = 8$ and input dimension $d_{\text{in}} = 1$, as used in this project for last state mode, the LSTM has $4 \times (8 + 64 + 8) = 320$ parameters in the recurrent layer, while the GRU has $3 \times (8 + 64 + 8) = 240$. The GRU therefore uses roughly two-thirds the parameters of an equivalent LSTM at the same hidden dimension.

The LSTM's explicit separation between long term cell state and short term hidden state gives it a structural advantage when both kinds of dependencies are present at the same time. The cell state can carry slow moving information while the hidden state responds to recent changes. This is also why Vuletic et al. [33] chose the LSTM for FinGAN, and we adopt the LSTM as the default for FinTimeGAN as well.

That said, there are reasons to keep the GRU as a configurable option. With only around 10,000 training observations per stock, there is a genuine risk that the LSTM's extra parameters lead to overfitting. The GRU's parameter efficiency could help in settings with shorter ticker history or lower frequency data with fewer observations. Making the cell type a configurable argument rather than a hard coded decision lets us test this without architectural changes.

5.8 Benchmark Models

We evaluate FinGAN and FinTimeGAN against three benchmarks: Fin-LSTM, ARIMA, and Ridge AR(L). All three are trained on the same excess return windows and evaluated by the paired half-day Sharpe convention defined in Chapter 8, so the headline numbers can be read on the same axis as the GAN models. The shuffled signal and constant position null benchmarks introduced in Section 8.4 apply equally to all three baselines.

5.8.1 The half day parity trap

A naive auto regressive predictor at half day frequency runs into a structural problem. With sliding stride $h = 1$, the parity of lag k flips between adjacent windows. Lag 1 of window i might be an overnight return, while lag 1 of window $i+1$ is intraday. A single linear model assigns one coefficient per lag position, which therefore averages over both return types and loses whatever differential structure exists between them.

More dangerously, a single model fit on the interleaved series can latch onto the alternation phase of the sequence rather than any genuine predictive structure. Overnight returns carry a small but persistent positive drift (the overnight risk premium), while intraday returns do not. A model with even mild sign bias on overnight bars can therefore produce a Sharpe ratio that looks like skill but is in fact "be long during overnight, indifferent during intraday." This is precisely the failure mode an early ARMA implementation in this project exhibited, with a PEP test Sharpe of approximately 3.93 that was almost entirely overnight drift capture.

The fix is to commit to target parity at the model level. We build two parallel models per benchmark: one for predicting intraday targets and one for predicting overnight targets. Each test window is routed to the appropriate model based on the parity of its target. Within each parity routed model, the lag position to type assignment becomes consistent and the alternation problems disappear.

The two linear benchmarks below differ in how they implement parity routing. Ridge AR(L) routes by the parity of the target only and keeps the full interleaved feature window, so an

intraday target Ridge regression has a previous overnight bar at lag 1, a previous intraday bar at lag 2, and so on. ARIMA, by contrast, fits on a single contiguous time series and parameterizes lags at fixed integer positions of that series. To route ARIMA by parity, the train, validation, and test slices are sub sampled into a parity 0 (intraday target) sub series and a parity 1 (overnight target) sub series, and an independent ARMA is fit on each sub series. This information loss is the structural cost of doing parity routing at the ARMA level, since an intraday target ARMA can no longer see the recent overnight bar.

5.8.2 Ridge AR(L)

Ridge AR(L) is the simplest linear benchmark with the same input format as FinGAN: the previous $L = 10$ half day returns, predicting the next one. The feature vector is the interleaved bar sequence $(r_{t-1}, r_{t-2}, \ldots, r_{t-L})$, where consecutive features alternate parity.

To address the half day parity trap of Section 5.8.1, we fit two independent Ridge models. One is trained on the windows whose target is intraday, the other on the windows whose target is overnight. Within each model, the lag position to type assignment is fixed. The intraday target model always sees a previous overnight bar at lag 1, a previous intraday bar at lag 2, and so on, and the overnight target model sees the mirror image. This means cross parity lag correlations are coherently exploited rather than averaged over.

The Ridge estimator is L2 penalized least squares,

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \|y - X\beta\|_2^2 + \alpha \|\beta\|_2^2, \tag{5.1}$$

with the penalty $\alpha = 0.01$ chosen by 5-fold cross-validation on the training fold. In 5 fold cross-validation, the training data is split into five equally sized folds; for each fold, the model is trained on the remaining four and validated on the held out fold, and the value of α minimizing the average validation error across the five folds is selected. A fixed intercept term is fit, and no demeaning (subtracting the sample mean from features and target) is applied to features or targets, so the resulting hard sign Sharpe matches FinGAN's hard sign Sharpe definition

exactly. Demeaning would shift the constant position Sharpe to zero on the training set but it is not what FinGAN reports.

The model is fit on the training fold only (8,750 windows for $L = 10$ with the default 80/10/10 split). The validation fold is reported for diagnostics but does not enter the fit. This mirrors FinGAN's protocol of freezing generator weights after training and running them unchanged on validation and test.

The trading rule converts the linear point forecast to a hard ± 1 position,

$$w_t^{\text{ridge}} = \text{sgn}(x_t^\top \hat{\beta}), \tag{5.2}$$

which is the same hard signal rule defined in Section 8.2. We therefore compare Ridge against the Generator's hard signal performance rather than its weighted signal, since both models emit a single number per window and trade ± 1 on its sign.

In addition to the hard Sharpe, we report two null benchmarks for every Ridge fit. The constant position Sharpe (`const_SR`) computes the same paired Sharpe with $\text{sgn}(\hat{r}_t) \equiv +1$, giving the Sharpe of being passively long throughout the test window. The shuffled target Sharpe (`shuf_SR`) keeps the predictor signs but applies them to a fixed random permutation of the targets, making the Sharpe achievable by a predictor with the same sign distribution but no temporal alignment with reality. We define the directional skill of the model as

$$\text{net} = \text{SR}_{\text{hard}} - \text{SR}_{\text{const}}, \qquad \text{skill} = \text{SR}_{\text{hard}} - \text{SR}_{\text{shuf}}, \tag{5.3}$$

and require both to be clearly positive before attributing the result to genuine forecasting skill.

5.8.3 ARIMA

$ARIMA$, Auto Regressive Integrated Moving Averages, is one of the classical ways of forecasting for financial time series. It is built of three components; AR which is auto regression,the integrated component I differences the series until it is stationary, and MA which is moving averages. Auto Regression indicates that the evolving variable of interest is regressed on its own lagged (prior) values. The MA component indicates that the regression error is modeled

as a linear combination of error terms that happened in the past and moving averages usually smooths out the noise from the time series. Furthermore, the I in ARIMA indicates that the data values have been replaced with the difference between their values and the previous values. The non seasonal ARIMA model has three parameters (p, d, q), where p is the order of the AR component, d is the number of differencing operations applied, and q is the order of the MA component [28]

The differencing order is fixed at $d = 0$ for all our experiments because half day excess returns are stationary by construction.[14] To confirm the $d = 0$, we perform the Augmented Dickey Fuller test (ADF) [5] in which the null hypothesis states that the series contains a unit root or equivalently $\phi = 1$ in an $AR(1)$ representation. [32] [15] The test returns several statistics including p-value, critical values, the number of lags used, and the number of observations. Our focus is on the p-value. A small p-value ($p < 0.05$) indicates strong evidence against the null hypothesis. Rejecting the null hypothesis means that the data are stationary[`?rules`].

On the training set of every ticker in our universe. The resulting p-values were below 10^{-6} in every case. With $d = 0$ the model reduces to ARMA(p, q) of the form

$$X_t = \alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}, \tag{5.4}$$

where ε_t are white-noise terms, α_j are autoregressive coefficients, and θ_i are moving average coefficients.

Order Selection

Following Vuletic et al.[33], For each ticker and each parity, the order $(p, 0, q)$ with the lowest Akaike Information Criterion (AIC) is chosen from the candidate set $\{(1,0,0), (1,0,1), (2,0,0), (2,0,1), (2,0,2)\}$. AIC is a standard model selection criterion that balances goodness of fit against model complexity.

Akaike Information Criterion (AIC) is a standard model selection criterion that balances goodness of fit against model complexity. For a model with k parameters and maximized log

[14] Recall from Chapter 4 that returns are computed at half day frequency, producing one overnight and one intraday observation per trading day.

[15] The referenced book give a better mathematical expression of this that can be found in page 41-42.

likelihood $\hat{L}$, it is defined as

$$AIC = 2k - 2\ln(\hat{L}). \tag{5.5}$$

A better fit increases $\hat{L}$ and therefore decreases the second term while a more complex model with more parameters increases the first term. The model with lowest AIC is preferred. This prevents the candidate selection from always choosing the richest model purely because it can fit the training data more closely, which would lead to overfitting.

No Trend Specification

Models are fit with no intercept term. Stationary excess returns have a near zero unconditional mean, so the constant offers little predictive value at the cost of an extra parameter.

5.8.4 Parity Routing on Sub sampled series

Unlike Ridge, ARIMA fits on a single contiguous time series and parameterizes lags at fixed integer positions of that series. To route by parity, the train, validation, and test slices are each subsampled into a parity-0 (intraday target) and parity-1 (overnight target) sub series, and an independent ARMA is fit on each parity-0 train sub series and each parity-1 train sub series. Forecasts are produced separately on the parity-0 and parity-1 test sub series and then re-interleaved into a single half day stream so that the Sharpe pairing convention from Chapter 8 still applies.

The conditioning length on each sub series is $\ell_{\text{per}} = L/2 = 5$, which keeps the total skipped half day positions at $2\ell_{\text{per}} = L = 10$, matching the condition window convention of every other model.

Rolling one step forecast

For the test fold, the procedure is the following. First, ARMA(p, q) is fit on the training sub series. Second, the validation sub series is appended to the state via Kalman filter update only, with no refit. This gives the model recent history without changing parameters, mirroring the warm start that FinGAN's LSTM gets at evaluation time. Third, the first ℓ_{per} test sub-series

observations are appended to the state. Fourth, for each subsequent test position t, the model forecasts one step ahead, the forecast is recorded, and then the realized observation is appended to the state before the next step. The forecast for position t uses information up to $t-1$ only, which makes the procedure non anticipative.

The forecasts are point forecasts, not distributions, so the trading rule is again the hard sign rule. The same const_SR and shuf_SR null benchmarks defined in Section 5.8.2 are reported for ARIMA on each ticker.

5.8.5 Fin-LSTM

Fin-LSTM is a recurrent baseline trained with the same economics driven loss combinations as FinGAN, but without the adversarial discriminator. It serves to isolate the contribution of the GAN framework: any improvement of FinGAN over Fin-LSTM can be attributed to the adversarial training rather than to the loss function design. The architecture is a single layer LSTM with hidden dimension $d_h = 8$, matching FinGAN's recurrent layer, followed by a dense output layer with no activation. The training procedure is identical to Phase 2 of Chapter 7, including the gradient norm matching calibration and the ten loss combination branches. The only difference is the absence of a discriminator and therefore the absence of a $J^{(G)}$ term in the generator loss.

Fin-LSTM produces a point forecast rather than a distribution, so its trading rule is the hard sign rule, and it is compared against FinGAN's hard signal performance under the same evaluation conventions.

Chapter 6

Loss Functions

The standard GAN training objective rewards the generator for producing samples the discriminator cannot distinguish from real data. For most applications this is a sensible goal, but for financial forecasting it is not enough. A generator that produces statistically plausible returns is not necessarily a generator that produces *profitable* forecasts. The whole point of a forecasting model in finance is to support trading decisions, and a model that scores well on distributional realism but poorly on directional accuracy is useless to a trader. FinGAN's central contribution is a set of additional loss terms that close this gap. Each term measures something a trader would actually care about, and including them in the generator's training objective pushes the generator toward forecasts that are economically useful, not just statistically realistic.

This chapter defines the four economics driven loss terms FinGAN introduces: profit and loss (PnL^*), mean squared error (MSE), Sharpe ratio (SR^*), and the standard deviation of strategy PnL (STD). Each term gets its own section with a financial motivation, a mathematical definition, and a discussion of how it enters the generator's loss. section 6.6 explains which combinations of these terms are sensible to include together, and section 6.7 writes out the full combined generator objective.

6.1 Notation and Conventions

Throughout this chapter, a_t is the conditioning window of past excess returns at timestep t, $x_{t+\Delta t}$ is the realized next return that the model is trying to forecast, and $Z_t \sim \mathcal{N}(0, I)$ is the

noise vector that drives the generator's stochasticity. The generator's forecast is $G(a_t, Z_t; \theta_g)$, with parameters θ_g. To keep formulas readable, we abbreviate the generator's forecast as $\hat{f}_t$ and the realized return as r_t when no ambiguity arises.

Loss terms are written as expectations over the joint distribution of the data and the noise. In practice each expectation is approximated by the sample mean over the N windows in a mini-batch.

A superscript star throughout this chapter denotes a smooth, differentiable approximation of a financial quantity. The starred version is what we use during training, where we need gradients. The unstarred version is the true financial quantity, which we use during evaluation, where we do not. The two versions agree closely on the inputs that matter most for trading. The star is a deliberate compromise: we accept a small approximation error in exchange for being able to train at all. Both PnL^* and SR^* are smooth proxies in this sense, and we make the relationship between each starred quantity and its true counterpart explicit when each term is introduced.

6.2 Profit and Loss Term (PnL^*)

The profit and loss of a trading strategy is the sum of profits and losses across all trades. For a strategy that takes a unit long or short position based on the sign of the model's forecast, the per step profit and loss is

$$PnL_t = \operatorname{sgn}(\hat{f}_t) \cdot r_t, \tag{6.1}$$

where the sign function returns $+1$ for a positive forecast (signaling a long position) and -1 for a negative forecast (signaling a short). If the model predicts the direction correctly, the realized return r_t has the same sign as the forecast and the product is positive. If the model gets the direction wrong, the product is negative. The strategy is rewarded for correct calls and penalized for incorrect ones, which is precisely what a directional trading rule should do.

We want to push the generator toward predictions that produce a positive PnL_t on average, which means using PnL_t as a training signal. There are two practical obstacles. The first is scale. Half day excess returns sit in the range of basis points, so PnL_t takes values around 10^{-4},

which produces gradients of inconvenient magnitude. We address this by averaging over the mini-batch and converting from decimal returns to basis points,[1]

$$PnL = \frac{10{,}000}{N} \sum_{t=1}^{N} \operatorname{sgn}(\hat{f}_t) \cdot r_t. \tag{6.2}$$

The factor of 10,000 is the conversion from decimal form to basis points and brings the loss values into a range that produces useful gradients.

The second obstacle is more fundamental. The sign function is not differentiable at zero and has zero derivative everywhere else, so it provides no gradient signal at all. We cannot use PnL as a training loss in its hard form. To make it differentiable, we replace the sign with a smooth approximation,

$$\tanh(\kappa \cdot \hat{f}_t), \tag{6.3}$$

where $\tanh(\cdot)$ is the hyperbolic tangent function and κ is a hyperparameter that controls how closely the approximation matches the true sign. The hyperbolic tangent is bounded with horizontal asymptotes at ± 1, has the same sign as its input everywhere, and is differentiable with non zero gradient throughout the real line. As κ grows, the approximation becomes sharper. As κ shrinks, the approximation becomes gentler and the gradients spread over a wider range of forecast values.

The smoothed version of PnL, which we call PnL^*, is then

$$PnL^*(\theta_g) = \mathbb{E}[\, \tanh(\kappa \cdot G(a_t, Z_t; \theta_g)) \cdot x_{t+\Delta t} \,], \tag{6.4}$$

or in mini-batch sample form,

$$PnL^* \approx \frac{10{,}000}{N} \sum_{t=1}^{N} \tanh(\kappa \cdot \hat{f}_t) \cdot r_t. \tag{6.5}$$

The star indicates that this is the smooth proxy used during training. The true PnL defined above is what gets reported during evaluation, where the hard sign is fine because we are not computing gradients.

[1] One basis point equals 0.01%, the standard unit for expressing small return differences in finance.

There are two practical issues with using this expression directly as a loss. First, raw excess returns are very small numbers near zero, so the resulting PnL values are hard to read and produce gradient signals at an inconvenient scale. To address this, we sum across all N time steps in a batch and convert from decimal returns to basis points,

$$PnL = \frac{10{,}000}{N} \sum_{t=1}^{N} \operatorname{sgn}(\hat{f}_t) \cdot r_t.$$

The factor of 10,000 converts from decimal form to basis points, where one basis point equals 0.01 percent which is the standard unit for expressing small return differences in finance. The choice of κ trades off approximation quality against gradient strength. A large κ produces a sharp approximation that closely matches the hard sign for any forecast away from zero, but its gradient saturates quickly because tanh flattens out. A small κ keeps gradients alive over a wider range of forecast values but makes the approximation poor even for confident predictions.

The right κ depends on the typical scale of the underlying return distribution. If returns have a standard deviation σ_r, then $\kappa \approx 1/\sigma_r$ puts the steep portion of the tanh curve in the region where most forecasts will fall. As an example, the stock PFE has a half day return standard deviation of approximately 0.011. With $\kappa = 100$, the smooth sign at one, two, and three standard deviations from zero is

$$\tanh(100 \times 0.011) = \tanh(1.1) \approx 0.80,$$

$$\tanh(100 \times 0.022) = \tanh(2.2) \approx 0.976,$$

$$\tanh(100 \times 0.033) = \tanh(3.3) \approx 0.997.$$

At one standard deviation the approximation captures 80% of the true sign, at two standard deviations it captures 97.6%, and at three standard deviations it is within 0.3% of the hard sign. The large directional moves that matter most for trading are therefore well approximated. The smooth gradient near zero ensures stable backpropagation, and the saturation at the tails ensures that the loss does not reward the generator for producing forecasts of unrealistic magnitude.

The PnL^* is a quantity we want to maximize, since larger values mean more profitable forecasts. Standard optimizers minimize, so we negate PnL^* before adding it to the loss. The

relationship is

$$\arg\max_{\theta} f(\theta) = \arg\min_{\theta} -f(\theta), \tag{6.6}$$

which means that minimizing $-\alpha \cdot PnL^*$ is equivalent to maximizing $\alpha \cdot PnL^*$ for any positive coefficient α. The generator's loss with the PnL^* term included becomes

$$\mathcal{L}_{gen} = \mathcal{L}_{BCE} - \alpha \cdot PnL^*, \tag{6.7}$$

where $\mathcal{L}_{BCE}$ is the adversarial component inherited from the standard GAN framework and $\alpha \geq 0$ is a coefficient that controls how strongly the PnL^* term influences training relative to $\mathcal{L}_{BCE}$. The choice of α is not a manual hyperparameter; it is set automatically by the gradient norm matching procedure described in Chapter 7, which calibrates each coefficient so that the corresponding loss term contributes a gradient of comparable magnitude to the BCE gradient.

When the generator improves its directional predictions, PnL^* increases, the term $-\alpha \cdot PnL^*$ decreases, and the total loss $\mathcal{L}_{gen}$ decreases. The optimizer rewards the generator for producing better directional forecasts, which is the behavior we want.

6.3 Mean Squared Error term (MSE)

The PnL^* rewards the generator for getting the direction of the forecast right. It does not constrain the magnitude. The hyperbolic tangent saturates at ± 1 as soon as $\kappa \cdot \hat{f}_t$ is large enough, so once the forecast is sufficiently signed, making it more signed has no further effect on the loss. A generator could in principle satisfy PnL^* by producing strongly signed forecasts that point in the right direction without ever learning to predict the actual size of the return. This would be a problem at evaluation time, where a forecast of $+0.5$ on a day when the realized return was $+0.008$ counts as a directional hit but is two orders of magnitude too large in absolute terms. A trader sizing a position from such a forecast would take on far more risk than the underlying signal justifies. The mean squared error term anchors the generator's forecast to the realized return,

$$MSE(\theta_g) = \mathbb{E}\big[\,(x_{t+\Delta t} - G(a_t, Z_t; \theta_g))^2\,\big], \tag{6.1}$$

or in sample form,

$$MSE \approx \frac{1}{N}\sum_{t=1}^{N}(\hat{f}_t - r_t)^2. \tag{6.2}$$

MSE is non negative and equals zero only when the forecast equals the realized return. A lower MSE means forecasts are closer to the realized values in absolute magnitude, not just in sign. The two terms together push the generator in complementary directions: PnL^* takes care of the sign, MSE takes care of the scale.

MSE is a quantity to be minimized, so it enters the generator loss with a positive sign,

$$\mathcal{L}_{gen} = \mathcal{L}_{BCE} - \alpha \cdot PnL^* + \beta \cdot MSE, \tag{6.3}$$

where $\beta \geq 0$ is the gradient norm matching coefficient for the MSE term.

6.4 Sharpe Ratio term (SR^*)

The Sharpe ratio is one of the most widely used metrics in finance for evaluating risk adjusted strategy performance. It captures profitability and volatility in a single number,

$$SR = \frac{\mathbb{E}[PnL]}{\sqrt{\mathrm{Var}[PnL]}}, \tag{6.1}$$

that is, the ratio of mean strategy returns to its standard deviation. A high Sharpe ratio indicates a strategy that earns consistently high returns relative to the risk it takes. A low Sharpe ratio indicates a strategy whose returns are eaten by their own volatility. Two strategies with the same average PnL are ranked differently by their Sharpe ratios if one of them is more consistent than the other.

Following Vuletic [33], we define a smooth proxy for the Sharpe ratio at training time by computing the ratio of mean to standard deviation of PnL^* rather than of the hard PnL,

$$SR^*(\theta_g) = \frac{\mathbb{E}[\, PnL_a^*(x_{t+\Delta t}, G(a_t, Z_t; \theta_g))\,]}{\sqrt{\mathrm{Var}[\, PnL_a^*(x_{t+\Delta t}, G(a_t, Z_t; \theta_g))\,]}}, \tag{6.2}$$

where $PnL_a^*(x, \hat{x}) = \tanh(\kappa\hat{x}) \cdot x$ is the per-sample smooth PnL for a single $(x, \hat{x})$ pair, in contrast to the batch averaged PnL^* from Section **??**. The expectation and variance in SR^* are

approximated by their mini-batch sample analogues,

$$SR^* \approx \frac{\bar{s}}{\sigma_s} = \frac{(1/N)\sum_{t=1}^{N} s_t}{\sqrt{(1/N)\sum_{t=1}^{N}(s_t - \bar{s})^2}}, \tag{6.3}$$

where $s_t = \tanh(\kappa \hat{f}_t) \cdot r_t$ is the smooth per time step PnL and $\bar{s} = (1/N)\sum_t s_t$ is its sample mean.

Two implementation details about SR^* deserve attention. First, SR^* is computed over a single mini-batch and is not annualized. The standard $\sqrt{252}$ annualization factor used at evaluation time is omitted here because training data is shuffled, so consecutive samples in a mini-batch are not consecutive in time. The per window sample is the natural unit for the training time loss. Second, we do not pair half sessions into daily PnL during training. The c2o (close-to-open) and o2c (open-to-close) half sessions of the same calendar day may end up in different mini-batches once the training data is shuffled, so daily aggregation is not possible at this stage. The training time SR^* is therefore a tractable proxy for the daily annualized Sharpe ratio reported in chapter 8, not the same quantity. The relationship between the two is examined in chapter 9.

Since SR^* is a quantity to be maximized, it enters the generator loss with a negative sign,

$$\mathcal{L}_{gen} = \mathcal{L}_{BCE} - \alpha \cdot PnL^* + \beta \cdot MSE - \gamma \cdot SR^*,$$

where γ is the gradient norm matching coefficient for SR term.

In principle, SR^* contains all the information that PnL^* and the standard deviation term provided. Maximizing the Sharpe ratio simultaneously maximizes the mean and minimizes the standard deviation of the smooth PnL series, so a model trained with SR^* alone is being pushed toward both objectives at once. In practice, however, the gradient structure of SR^* differs from that of the individual terms, and the two formulations push the generator toward different regions of the return distribution. Vuletic [33] reports an average pairwise correlation of approximately 33.5% across the daily PnL series produced by the ten loss combinations on her researched tickers universe, which suggests that the combinations encourage the generator to learn meaningfully different forecasts despite their mathematical relationship. The cross combination correlation on our ticker universe is examined in chapter 9.

6.5 PnL's Standard Deviation Term (STD)

The Sharpe ratio combines the mean and standard deviation of the smooth PnL series into a single quantity. If we want the generator to feel these two pressures separately, with independent gradient norm coefficients, we can include them as separate loss terms instead. The standard deviation term penalizes the generator for producing inconsistent trading signals, and is defined as

$$STD(\theta_g) = \sqrt{\mathrm{Var}[\, PnL_a^*(x_{t+\Delta t}, G(a_t, Z_t; \theta_g))\,]}, \tag{6.1}$$

or in sample form,

$$STD \approx \sqrt{\frac{1}{N}\sum_{t=1}^{N}(s_t - \bar{s})^2}, \tag{6.2}$$

where s_t and $\bar{s}$ are as in the SR^* definition above. A lower STD means the strategy is producing more consistent returns across time steps. We use standard deviation rather than variance because the denominator of SR^* also uses standard deviation, and the gradient norm matching coefficients should operate on quantities with the same units.

The STD is a quantity to be minimized, so it enters the generator loss with a positive sign,

$$\mathcal{L}_{gen} = \mathcal{L}_{BCE} - \alpha \cdot PnL^* + \beta \cdot MSE + \delta \cdot STD, \tag{6.3}$$

where $\delta \geq 0$ is the gradient norm matching coefficient for the STD term.

6.6 The Hierarchy Principle

The four economics driven terms PnL^*, MSE, SR^*, and STD cannot all be combined freely. Some combinations would push the generator toward degenerate solutions, and others would encode the same objective twice through different gradient structures. Vuletic [33, p.110] states four conditions on the non-negative coefficients $\alpha, \beta, \gamma, \delta$ that constrain which subsets are sensible to include together. We follow these throughout this project, and they determine the ten loss combinations enumerated in Phase 2 of training in chapter 7.

1. $\max(\alpha, \gamma, \delta) > 0$: at least one economics driven term other than MSE must be included. This prevents the loss from reducing to standard supervised learning (MSE alone) or pure

adversarial training (BCE alone), both of which exist as named baseline combinations rather than as FinGAN itself.

2. If $\beta > 0$ then $\max(\alpha, \gamma, \delta) > 0$: if MSE is included, at least one of PnL^*, SR^*, or STD must also be included. MSE is never the only economics driven term in a FinGAN combination.

3. If $\delta > 0$ then $\alpha > 0$: STD is included only if PnL^* is included. The STD term measures the standard deviation of the smooth PnL series. If the generator is not also being rewarded for the *mean* of that series through PnL^*, then minimizing STD alone rewards the generator for producing constant predictions, since zero variance is achievable by collapsing the output to a single value. Vuletic [33, p.110] confirms this empirically: SR^* and MSE alone, with high coefficients, produce very narrow generated distributions, and including PnL^* alleviates this collapse.

4. $\min(\gamma, \delta) = 0$: SR^* and STD are never included at the same time. Because $SR^* \approx \mathbb{E}[PnL^*]/\sqrt{\mathrm{Var}[PnL^*]}$, the SR^* term and the $\{PnL^*, STD\}$ pair convey the same information through different gradient structures. Including both creates redundant pressure on the same underlying objective.

These four rules produce the ten loss combinations enumerated in Chapter 7: eight valid economics driven combinations (PnL^*; $PnL^*\&STD$; $PnL^*\&MSE$; $PnL^*\&SR^*$; $PnL^*\&MSE\&STD$; $PnL^*\&MSE\&SR^*$; SR^*; $SR^*\&MSE$), an MSE only combination, and a BCE only combination that recovers the original ForGAN objective.

6.7 Combined Generator Loss

Pulling all four economics driven terms together with the discriminator feedback, the FinGAN generator loss as given by Vuletic [33, Eq.5.13] is

$$L^G(\theta_d, \theta_g) = J^{(G)}(\theta_d, \theta_g) - \alpha \cdot PnL^*(\theta_g) + \beta \cdot MSE(\theta_g) - \gamma \cdot SR^*(\theta_g) + \delta \cdot STD(\theta_g), \quad (6.1)$$

where $J^{(G)}$ is the generator's adversarial loss term (the generator-side BCE component defined in Equation 5.1, in our setting) and $\alpha, \beta, \gamma, \delta \geq 0$ are the gradient norm matching coefficients. Vuletic [33, Eq.5.13] writes this equation with $J^{(G)}$ left as one of the standard GAN losses (zero-sum, binary cross entropy, or Wasserstein), allowing flexibility in the choice of adversarial component. We follow her in using binary cross entropy throughout this project, since she reports that the Wasserstein GAN with gradient penalty suffered from numerical explosion in her experiments.

The signs in Equation 6.1 follow from whether each term is to be maximized or minimized. PnL^* and SR^* enter negatively because they are quantities we want to maximize, and the optimizer minimizes; MSE and STD enter positively because they are quantities we want to minimize directly. The coefficients $\alpha, \beta, \gamma, \delta$ are calibrated by the gradient norm matching procedure of Chapter 7, so that each economic -driven term contributes a gradient of comparable magnitude to the $J^{(G)}$ gradient. Subject to the hierarchy principle of Section 6.6, the ten loss combinations enumerated in chapter 7 correspond to different choices of which subset of $\{\text{PnL}^*, MSE, SR^*, STD\}$ to include.

The four economics driven terms each play a distinct role in shaping the generator's behaviour. PnL^* and SR^* enforce sign consistency between the forecast and the realized return: they push the generator to get the direction right. MSE enforces magnitude calibration: it keeps the forecast close to the realized value rather than only pointing in the right direction. STD provides risk control: it penalizes inconsistent strategy returns. The discriminator feedback through $J^{(G)}$ is what allows the generator to learn the conditional distribution of returns rather than only point wise forecasts [33, p.111]. Removing any of these terms changes the kind of skill the generator is being asked to learn, which is the underlying reason FinGAN's per combination performance varies meaningfully across tickers and why we test all ten combinations rather than committing to one.

Chapter 7

Training

This chapter describes the training process for the three models of FinGAN, LSTM-Fin, and FinTimeGAN. LSTM-Fin and FinGAN are baseline models that we implemented from scratch following Vuletic [33] using Tensorflow and Keras libraries in order to create production ready code and modularize so that it can be run as a script. FinTimeGAN is our contribution in which we extend the FinGAN with a Embedder and Recovery networks inspired by TimeGAN [34]. This separation lets the adversarial training operate in a learned latent space rather than directly on returns.

Since FinTimeGAN is the only model having the Embedder and Recovery networks, we separate the training into two phases. Phase 1 applies only to FinTimeGAN and is a pre-training stage for the Embedder and Recovery on a reconstruction objective. Phase 2 starts with a gradient norm matching algorithm to avoid gradient vanishing or exploding in the recurrent layers, and is followed by training the generator and discriminator on ten different loss combinations. Phase 2 is shared across all three models.

The training is designed to run through loss combination as an independent branch. After the gradient norm matching phase, we take a snapshot of the model weights as a baseline. This allows us to start the training of each combination by restoring the model from the same baseline weights, so that the only difference between combinations is the loss function each one is trained against.

A note on FinTimeGAN's two context modes. The Embedder and Recovery networks are the same in both `last` mode and `attention` mode, so Phase 1 runs identically regardless of the context mode chosen. Phase 2 is also identical at the algorithm level. The generator and discriminator architectures differ between the two modes (Section 5.7.4), but the optimizer, normalization, gradient norm matching procedure, loss combination set, and branched training structure are unchanged. Wherever this chapter refers to *the generator* or *the discriminator*, the description applies to whichever architectural variant is active.

7.1 Notation and Hyperparameters

The symbols used throughout this chapter and the next are collected in Table 7.1. The hyperparameters of the training procedure are listed in Table 7.2. The specific numerical values for each experiment are reported in the corresponding section of the results chapter.

Symbol	Meaning
T	Set of tickers (training pool $\cup$ unseen)
L	Condition window length
$c \in \mathbb{R}^{L\times 1}$	Condition (last L excess log returns)
$y \in \mathbb{R}$	Next-step realized return (target)
$z \in \mathbb{R}^{d_z}$	Latent noise, $z \sim \mathcal{N}(0, I)$
E, R, G, D	Embedder, Recovery, generator, discriminator
$H_c = E(c)$	Latent encoding of the condition
$\hat{H} = G(z, H_c)$	Generated latent
$\hat{x} = R(\hat{H})[-1, 0]$	Decoded next-step return
κ	tanh sharpness applied to normalized forecasts (default 100)
$\alpha, \beta, \gamma, \delta$	Gradient norm matching coefficients
$\theta^\star$	Post-calibration checkpoint $(E^\star, R^\star, G^\star, D^\star)$

Table 7.1: Notation used in the training procedure.

Parameter	Description
L	Condition window length
d_h	Embedding/hidden dimension
d_z	Latent noise dimension
N_1	Phase 1 (Embedder/Recovery) pre-training epochs
N_{grad}	Gradient norm matching epochs
N_2	Phase 2 epochs per combination branch
η_1, η_2	Phase 1 and Phase 2 learning rates
κ	tanh sharpness for the sign proxy
n_D	Discriminator updates per generator update (default 1)

Table 7.2: Hyperparameters of the training procedure. Specific values are reported in Chapter 9.

7.2 Phase 1 Training

As stated above, Phase 1 applies only to FinTimeGAN. The Embedder E_θ maps a return space window $x \in \mathbb{R}^{L\times 1}$ to a latent sequence $H \in \mathbb{R}^{L\times d_h}$. The Recovery network R_ϕ inverts this mapping back to return space. In this phase, they are trained jointly to minimize the reconstruction loss

$$\mathcal{L}_{\text{rec}}(\theta,\phi) = \mathbb{E}\left[\frac{1}{L}\sum_{s=1}^{L}\big([R_\phi(E_\theta(x))]_s - x_s\big)^2 \;+\; \frac{1}{L}\sum_{s=1}^{L}\big(R_\phi([H_s]_{\text{iso}}) - x_s\big)^2\right], \tag{7.1}$$

where $[H_s]_{\text{iso}}$ denotes the s-th latent position reshaped into a length-1 sequence and decoded in isolation from the rest of H.

The first is an ordinary sequence level reconstruction loss: it asks whether the Recovery network can rebuild the full input window from the full latent sequence. The second, which we call the per position term, is the key to making FinTimeGAN work. Without it, the Embedder would be free to distribute information about x_s across other positions in H, as long as the Recovery network could unscramble the result when given the whole sequence. That freedom would be a problem at adversarial training time, because the generator produces each next latent vector on its own and the Recovery network must be able to decode that vector in isolation. The per position term forces every latent to carry enough information to decode its corresponding return without the surrounding context, which is exactly the property the generator's output needs.

Phase 1 uses Adam with learning rate 10^{-3}, mini-batches of size 100, for up to 50 epochs with early stopping once the per-batch reconstruction loss falls below a small threshold of 10^{-9}. At convergence, the reconstruction RMSE on the test split is on the order of 1% of the data standard deviation, which we take as sufficient. Any distortion introduced by freezing the Recovery network during Phase 2 is small relative to the signal the generator is trying to learn.

Before moving to Phase 2, we report two latent space diagnostics. The first is the number of hidden units whose activation standard deviation falls below 0.01, which flags per unit collapse. The second is the mean pairwise Euclidean distance between final position latent summaries across the training condition windows. A value close to zero would indicate that all inputs are being mapped to the same region of latent space, which would indicate aggregate collapse. Both diagnostics need to pass before Phase 2 begins.

After Phase 1, both E_θ and R_ϕ are frozen. Their weights are not added to any Phase 2 optimizer, so no gradients flow into them during adversarial training. The latent space is learned once and serves as a fixed point for everything that follows.

7.3 Phase 2 Training

This phase is shared across all three models. It has three parts: normalization and initialization setup, gradient norm matching for loss scaling, and training across the different loss combinations, each branching from a shared post calibration checkpoint.

7.3.1 Optimizer, Normalization, and Initialization

Following Vuletic [33], both the generator and the discriminator use Root Mean Square Propagation (RMSProp), an adaptive learning rate optimization algorithm. We set decay rate $\rho = 0.99$, numerical stabilizer $\epsilon = 10^{-8}$, and learning rate 10^{-4} on each network. Each mini-batch consists of $n_D = 1$ discriminator update followed by one generator update. In our experiments, this 1:1 ratio was sufficient to avoid the adversarial game saturating on either side.

In conditional GANs each sample's output is meant to depend only on its own noise vector Z and the conditioning window. Standard batch normalization breaks this property because it normalizes activations across the mini-batch, which means the discriminator's evaluation of one sample depends on the other samples in the batch through the shared batch statistics. The discriminator can then exploit batch level correlations that are absent at inference time, when each sample is scored on its own. The standard fix for this problem was proposed by Salimans et al. [31], who introduced *Virtual Batch Normalization*, in which the normalization statistics are computed once from a fixed reference batch and held constant throughout training.

We take the first 100 chronologically ordered training windows as the reference batch and compute scalar reference mean μ_{ref} and scalar standard deviation σ_{ref} from them. Because we use a chronological 80/10/10 train/validation/test split, these 100 windows are the earliest 100 windows in the dataset and therefore precede every validation and test point which makes the normalization non anticipative by construction. Inputs to all networks operating in return space are transformed as

$$\tilde{X} = \frac{X - \mu_{\text{ref}}}{\sigma_{\text{ref}}}, \tag{7.1}$$

and the Recovery network's outputs are denormalized by the inverse transform.

The weights on network layers with tanh or sigmoid activations are initialized using Glorot normal initialization [14], in which weights are drawn from a zero-mean Gaussian with variance

$$\sigma^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}, \tag{7.2}$$

[14] where n_{in} and n_{out} are the input and output dimensions of the layer. Glorot initialization keeps activation variance roughly constant across layers under tanh and sigmoid nonlinearities, which mitigates the gradient vanishing problem that recurrent architectures are particularly prone to. Following the activation aware initialization guidance [24], we initialize layers with ReLU activations using He initialization [20] instead. The current architecture is shallow enough that the distinction has little practical effect, but we adopt this per activation scheme in anticipation of deeper variants such as longer condition windows or attention-based context aggregation, where variance preservation across depth becomes more consequential.

The hidden and cell states of all recurrent layers are reset to zero at the start of every mini-batch. This forces each forecast to depend on its explicit conditioning window rather than on residual state carried over from whichever windows happened to precede it in the batch stream, and it allows the training data to be reshuffled at every epoch.

7.3.2 Gradient Norm Matching

The financial loss terms and the BCE loss have gradient norms whose magnitudes depend on the numerical scale of returns rather than on how much each term ought to contribute to the final objective. Without a scaling correction, the financial terms would either dominate the BCE gradient or vanish beside it. To address this, we follow Vuletic's gradient norm matching algorithm [33].

Over a 25 epoch calibration phase, for each mini-batch we record the L^2 norms of the generator gradients induced by each candidate loss term (BCE,PnL, MSE, SR, $and STD$) with respect to the generator's parameters. Let g_{BCE}, g_{PnL}, g_{MSE}, g_{SR}, g_{STD} denote the per iteration norms. The scaling coefficients are the expectations of the ratios of BCE to each financial loss norm

$$\alpha = \mathbb{E}[g_{BCE}/g_{PnL}]\,, \qquad \beta = \mathbb{E}[g_{BCE}/g_{MSE}]\,,$$
$$\gamma = \mathbb{E}[g_{BCE}/g_{SR}]\,, \qquad \delta = \mathbb{E}[g_{BCE}/g_{STD}]\,.$$

The expectation is estimated as the sample mean over all mini-batches in the $Ngrad_{grad}$ epoch calibration phase, with $\alpha, \beta, \gamma, \delta$ applied as scaling coefficients during Phase 2. After calibration, each financial loss gradient contributes on the same order of magnitude as the BCE gradient, so the relative influence of any term reflects the objective we chose to include rather than the numerical scale of returns.

Our implementation of gradient norm matching differs from Vuletic's in one respect: during calibration we update only the discriminator, not the generator. Vuletic's implementation records the gradient norms and additionally applies the BCE gradient as an optimizer step on the generator, which makes the 25 epoch calibration simultaneously a BCE-only warmup. We compute and measure the generator's gradients with TensorFlow's `GradientTape`, but no op-

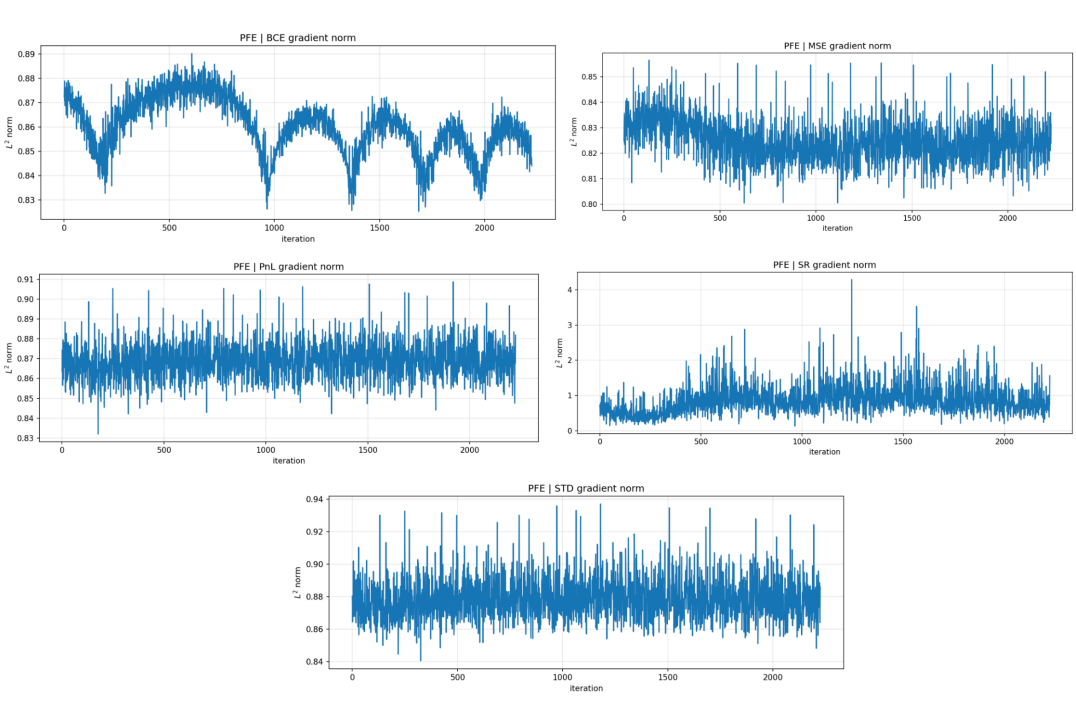


Figure 7.1: Per iteration gradient norms recorded during calibration for the PFE ticker. After the calibration coefficients are applied, all norms are on the same order of magnitude.

timizer step is taken, so the generator's weights at the end of calibration are identical to its Glorot initialized weights. This keeps the implementation conceptually clean, because the coefficients α, β, γ, δ are properties of the *initial* loss landscape rather than of an already warmed up generator.

7.3.3 Loss combinations

Phase 2 concludes with the training of the generator and discriminator on each of ten loss combinations. Following [33], the first nine combinations are the economics driven set: PnL^*, $PnL^* + MSE$, $PnL^* + MSE + STD$, $PnL^* + MSE + SR^*$, $PnL^* + SR^*$, $PnL^* + STD$, SR^*, $SR^* + MSE$, and MSE. The tenth combination is BCE-only, which recovers the original ForGAN [23] objective and serves as an adversarial baseline.

Each combination is trained as an *independent branch* from a shared starting point. Let $\theta^\star$ denote the post calibration state of the networks: the frozen Embedder and Recovery (Fin-TimeGAN only), the Glorot initialized generator, and the discriminator after 25 epochs of BCE

updates during calibration. At the start of each combination's 100-epoch training run, the generator and discriminator weights are restored to $\theta^\star$, fresh RMSProp optimizers are instantiated for both networks, and the random number generator is reseeded to a shared value s_0. The generator, discriminator, noise samples, and batch shuffling order are therefore identical across combinations at the start of training. The only thing that varies is the loss function. Any difference in performance between two combinations is a result of their loss term choice and not of different initializations, different optimizer states, or different random draws.

Algorithm 1: Phase 2: independent loss combination branches

Input: Post-Phase-1 Embedder E and Recovery R (FinTimeGAN only); generator G; discriminator D; training data; combo set $\mathcal{C}$; branch epochs $N_2 = 100$; gradient norm matching epochs $N_{\text{grad}} = 25$; seed s_0.

Output: Per-combo generator/discriminator snapshots $\{(\hat{G}_k, \hat{D}_k)\}_{k\in\mathcal{C}}$.

Compute reference statistics $\mu_{\text{ref}}, \sigma_{\text{ref}}$ from the first 100 training windows ;

Initialize layers with tanh or sigmoid activations using Glorot normal, and ReLU layers using He initialization ;

`// Gradient norm matching`

for N_{grad} *epochs* **do**

  foreach *mini-batch* (c, y) **do**

    Update D via RMSProp on BCE loss ;

    Record $\|\nabla_G \mathcal{L}\|_2$ for $\mathcal{L} \in \{BCE, \text{PnL}^*, MSE, SR^*, STD\}$ (no optimizer step on G) ;

  end

end

Compute $\alpha, \beta, \gamma, \delta$ as means of BCE-to-loss gradient ratios ;

$\theta^\star \leftarrow$ snapshot(E, R, G, D) ;

`// Independent combo branches`

foreach *combo* $k \in \mathcal{C}$ **do**

  Restore $(E, R, G, D) \leftarrow \theta^\star$;

  Instantiate fresh RMSProp optimizers for G and D ;

  Reseed RNG to s_0 ;

  for N_2 *epochs* **do**

    foreach *mini-batch* (c, y) **do**

      $H_c \leftarrow E(c)$; `// frozen for FinTimeGAN, identity otherwise`

      Sample $z \sim \mathcal{N}(0, I)$; compute $\hat{x} = R(G(z, H_c))$ (or $G(z, c)$ for FinGAN/LSTM-Fin) ;

      Take one D step on BCE ;

      Sample fresh z; take one G step on $\mathcal{L}_G^{(k)}$;

    end

  end

  $(\hat{G}_k, \hat{D}_k) \leftarrow$ snapshot(G, D) ;

  Evaluate $(\hat{G}_k, \hat{D}_k)$ on validation and test windows ;

end

There are two reasons to train the combinations independently rather than chaining them. The scientific reason is that if combinations were trained sequentially, combination $k+1$ would start from combination k's endpoint. The performance attributed to combination k would then reflect the cumulative effect of combinations $1, \ldots, k$ rather than the effect of combination k alone. Independent branches make each per-combination Sharpe ratio a standalone statement about that loss term. The practical reason is that the comparison between combinations is the

point of the experiment. We aim to identify which loss combination produces the best forecasts, and an experimental design that tangles the combinations together cannot answer that question.

A brief note on the reference implementation. Algorithm 2 of [33] describes independent branches by stating that each combination's loop begins with a copy of the generator from the post-calibration checkpoint. The accompanying code,[1] however, threads a single shared generator, discriminator, and pair of optimizers through every combination call in the `FinGAN_combos` function. None of the operations performed inside those loops resets the network state, the optimizer state, or the running statistics that RMSProp accumulates. Each combination after the first therefore inherits the weights and optimizer state produced by the previous combination, which produces sequential training steps rather than a set of independent runs. The k-th variant ends up with $k \cdot N_2$ epochs of total training, with each preceding loss shaping the initialization seen by the next.

The technical explanation of how this chained training happens is:

- *FinGAN_combos* function, a single generator, discriminator, and pair of *RMSprop* optimizers are constructed once and threaded through all ten $TrainLoopMain*$ calls in sequence. None of the operations performed inside those loops resets this state, $gen.train()$ only toggles the training mode flag.

- *zero_grad*() : Clears .grad (per the PyTorch documentation, optionally to None) but leaves *param.data* and the optimizer's running statistics (RMSprop's v_t buffer) untouched. Also, the *optimizer.step*() mutates *param.data* in place. Because the $gen, disc, gen\_opt, disc\_opt$ arguments are Python references, every call after the first inherits the weights and optimizer state produced by the previous variant. The gen PnL, gen PnL MSE, bindings returned by each loop alias the same underlying module rather than naming distinct models.

The consequence is that the reported per loss metrics, the saved checkpoints, and the cross strategy correlation matrix in the reference implementation confound loss function effects with

[1] `https://github.com/milenavuletic/Fin-GAN`

training stage effects. The order in which the combinations are called becomes load bearing in a way Algorithm 2 does not anticipate. Restoring the algorithm's intent requires snapshotting the post calibration state and reloading it, along with reinitialized optimizers, before each combination's training loop. We follow the algorithm's stated intent rather than the reference code: we snapshot the weights after gradient norm matching and restore from those weights at the start of every combination. To produce comparable results, we have applied the same fix to FinGAN and LSTM-Fin and obtained new per-ticker numbers, which are reported in Chapter 9.

The full pipeline (Phase 1 for FinTimeGAN, gradient norm matching, and the ten independent combination branches of training) is run on 5 different seeds. The reseeding inside the combination loop ensures that combinations are directly comparable within a seed, and seed-to-seed variation gives us an estimate of the uncertainty in each reported Sharpe ratio. Final results are reported as mean and standard deviation across seeds in Chapter 9.

7.3.4 Adversarial Equilibrium Diagnostics

A standard sanity check for adversarial training is to monitor the discriminator's mean output on real and generated samples during the BCE-only branch. At the theoretical optimum of the minimax game [15], the discriminator cannot distinguish real from generated samples and outputs approximately 0.5 on both. In practice, on financial return data this limit is rarely reached, because the true conditional $p(x_{t+\Delta t} \mid a_t)$ concentrates near a small number of realized returns, which encourages the generator to produce those specific values and the discriminator to reward them [33]. What matters for training stability is not that the game reaches equilibrium but that neither side dominates. A discriminator that consistently outputs near 1 on real samples and near 0 on generated samples (or the reverse) indicates the adversarial signal has collapsed. In all of our BCE-only runs, both $D(\text{real})$ and $D(\text{fake})$ remain within a band around 0.5 throughout training, which confirms that the game is well behaved and that the financial losses in the other nine combinations are being added to a stable base.

Chapter 8

Model Evaluation

A generative model of financial returns is only as useful as its forecasts can be turned into trading decisions that result in profit accumulation. Training loss and sample diagnostics tell us whether the generator has learned the statistical properties of the data and is generating realistic looking returns. This chapter is dedicated to evaluating model forecasts and answering the question: "If we use a generator's forecasts to place trades, would we make a profit, and what is the probability of that tradable signal being genuine rather than luck, noise, market drift, or a broken model?"

8.1 Distributional Forecast or a Point Forecast?

A standard supervised model, such as linear regression, gradient-boosted trees, or a plain LSTM, produces a single predicted value per window. That point forecast becomes the strategy: long if positive, short if negative. These models are typically evaluated on point error metrics like MAE and RMSE.

The FinGAN and FinTimeGAN models do not forecast a point. For each conditioning window they draw $B = 1000$ Monte Carlo samples from a learned conditional distribution over the next return. A point forecast is not rich enough to make trading decisions when uncertainty matters. Consider two distributions: one tightly clustered above zero, the other wide and roughly symmetric around zero. Both can have the same mean, but they imply very different trading decisions. Confidence about position size only emerges from the shape of the distribution,

not from its mean. Our evaluations are therefore centered around measuring what additional distributional information is worth, relative to a rule that uses the mean alone.

This distinction also affects how results should be visualized. For a regression model, plotting realized returns against predicted points conveys model performance directly. For a distributional forecast, that plot does not reflect economic performance: the sample mean of the conditional distribution is conservative by construction. To illustrate, in a near zero mean return series, the conditional expectation is close to zero even when the generator is extracting a real signal. A flat looking mean overlay is therefore consistent with a model that is making confident, profitable trades through the shape of its distribution. For these reasons, we rely on cumulative PnL curves, the annualized Sharpe ratio, and permutation statistics rather than on mean forecast overlays.

8.2 From Forecast Distribution to Trading Position

The generator's output for a window at time t is a set of $B = 1000$ Monte Carlo samples $\{\hat{x}_{t,i}\}_{i=1}^{B}$, conditioned on the previous L returns. We convert this output into a trading position in two ways, each corresponding to a different claim about what the generator has learned.

8.2.1 Hard Signal

The simplest reduction is to compute the sample mean $\bar{\hat{x}}_t = \frac{1}{B}\sum_i \hat{x}_{t,i}$ and take its sign:

$$h_t = \operatorname{sgn}(\bar{\hat{x}}_t) \in \{-1, 0, +1\}. \tag{8.1}$$

This discards the distribution and trades at full size in whichever direction the mean points. It is also how a regression-style model would be used.

8.2.2 Weighted Signal

The second approach uses the samples to estimate the probability that the next return is non-negative:

$$p_t^{\text{up}} = \frac{1}{B}\sum_{i=1}^{B}[\hat{x}_{t,i} \geq 0], \qquad p_t^{\text{dn}} = 1 - p_t^{\text{up}}. \tag{8.2}$$

The weighted signal is the model's net directional confidence:

$$w_t = p_t^{\text{up}} - p_t^{\text{dn}} \in [-1, +1]. \tag{8.3}$$

If 95% of the samples are positive, $w_t = 0.9$ and the strategy is nearly fully long.[1] If the samples are split evenly around zero, $w_t = 0$ and the strategy sits out. The magnitude of w_t scales the position size by confidence: the strategy is proportionally rewarded for being confidently right and proportionally penalized for being confidently wrong. This is analogous to treating the softmax margin as a position sizing signal in a classification problem.

8.2.3 Per Window PnL

Given either signal $s_t \in \{h_t, w_t\}$, the per half session PnL is

$$\text{PnL}_t = 10{,}000 \cdot s_t \cdot r_t, \tag{8.4}$$

where r_t is the realized excess return for that half-day session and the 10,000 factor converts the PnL into basis points.[2] Reporting both h_t and w_t tells us whether the weighted rule is earning its extra complexity. If the weighted PnL consistently beats the hard signal PnL, then the shape of the generator's distribution carries information that the mean alone does not.

8.2.4 Signal Availability Across Model Classes

The hard signal h_t and the weighted signal w_t are not interchangeable across model classes. Point forecast models (ARIMA5.8.3, Ridge AR(L)5.8.2, LSTM-Fin5.8.5) produce a single predicted value per window and are therefore evaluated under the hard signal rule only. Distributional models (FinGAN 5.6, FinTimeGAN 5.7) produce $B = 1000$ samples per window and are evaluated under both rules. The hard signal Sharpe of a distributional model is directly comparable to the Sharpe of a point forecast model; the weighted signal Sharpe is the additional contribution from leveraging distribution shape. The diagnostics described in Section 8.4 are applied to

[1] A *long* position profits if the asset goes up.

[2] More detail about this conversion is given in Chapter 6.

whichever signal a given model produces, with the weighted signal taking precedence when both are available.

8.3 Daily PnL and Sharpe Ratio

As introduced in Chapter 4, each calendar day has an interleaved return series for intraday (open-to-close, $o2c$) and overnight (close-to-open, $c2o$). The PnL produced by a model is therefore an interleaved series of intraday and overnight values, which we combine by summing within each day:

$$\mathrm{PnL}_d^{\mathrm{daily}} = 10{,}000 \cdot \left(s_d^{\mathrm{c2o}} \cdot r_d^{\mathrm{c2o}} + s_d^{\mathrm{o2c}} \cdot r_d^{\mathrm{o2c}}\right). \tag{8.1}$$

The annualized Sharpe ratio of the resulting daily PnL series is

$$\mathrm{SR} = \sqrt{252} \cdot \frac{\overline{\mathrm{PnL}^{\mathrm{daily}}}}{\sigma(\mathrm{PnL}^{\mathrm{daily}})}, \tag{8.2}$$

where the $\sqrt{252}$ factor annualizes the per-day PnL statistics under the assumption of roughly 252 trading days per year.

In addition, we report the intraday and overnight PnL separately and plot them as a sanity check. Overnight and intraday returns have different statistical properties: overnight returns are smaller and more strongly driven by macroeconomic and overnight news factors, while intraday returns are noisier. A Sharpe ratio that is dominated by one half session may indicate that the model is exploiting structural features rather than forecasting genuine returns.

8.4 Evaluation Baselines and Diagnostics

Suppose a generator achieves a Sharpe ratio of 1.5 on a test ticker. Before drawing a positive conclusion, we need to rule out three failure modes:

- The Sharpe could be picking up the asset's *unconditional drift*[3] rather than any forecast information from the model.

[3]The long-run average direction of the asset over the test window, which may be entirely incidental to the model.

- It could be a *finite-sample artifact*[4] that would disappear on another year of data.

- The generator could have *mode collapsed* into a degenerate distribution that happens to produce reasonable-looking aggregate numbers by accident.

The baselines and diagnostics below are designed to address each.

Before describing them, we note one methodological choice. Vuletic et al. [33] evaluate FinGAN performance descriptively, by comparing per ticker Sharpe ratios against named baselines (long only, ARIMA, LSTM, LSTM-Fin, ForGAN-BCE) without a formal statistical test. We follow this comparative evaluation for direct comparability with Vuletic's [33] results, and additionally introduce a permutation test on the weighted signal Sharpe for the distributional models. As detailed in Section 8.4.2, this permutation test is run at two levels: at the portfolio level as the basis for the headline statistical significance claim of this thesis, and at the per ticker level as a descriptive diagnostic.

8.4.1 Ruling Out Market Drift

The simplest way that a model can appear successful is by always trading in the direction of an asset that happened to trend during the test window. The following benchmark and diagnostic address this failure mode.

Constant sign Sharpe ($\mathrm{SR}_{\mathbf{const}}$).

The strategies $w_t \equiv +1$ and $w_t \equiv -1$ hold a unit position throughout the test window, ignoring the model entirely. Their Sharpe ratios are the risk adjusted return of being unconditionally long or short the asset.[5] The constant long benchmark tests the null hypothesis

$$H_0^{\text{const}} : \mathrm{SR}_{\text{model}} \leq \mathrm{SR}_{\text{const}}, \tag{8.1}$$

[4]The test sample may be insufficient. Sharpe ratios have noisy statistics because they depend on the ratio of a small mean to a small standard deviation.

[5]For ETFs forecast on raw returns this corresponds to buy and hold. For single stock excess returns this corresponds to long the stock and short the sector ETF in equal notional.

that the model's directional forecasts produce no better risk adjusted return than constant long exposure. If the model's Sharpe is close to $\mathrm{SR}_{\mathrm{const}}$, the model has learned to stay long and nothing more.

Forecast target correlation.

We report the Pearson correlation $\rho(\bar{\hat{x}}_t, r_t)$ between the sample mean forecast and the realized return. Unlike Sharpe, correlation is indifferent to position sizing and to the asset's average drift, so a non-trivial correlation is direct evidence that the generator is tracking something about r_t itself. This is a diagnostic rather than a trading benchmark, because correlation cannot be traded directly, but it provides a clean separation of directional skill from drift.

8.4.2 Ruling Out Finite-Sample Noise

To distinguish genuine signal from noise on a finite test window, we apply the diagnostics below. The permutation test is run at two levels of aggregation, with different roles assigned to each.

Portfolio level shuffled signal Sharpe.

For each trained distributional model (FinGAN, FinTimeGAN), we form the portfolio daily PnL series by summing the per ticker daily PnL across the universe, as defined in Section 8.3. We then draw $N_{\mathrm{shuffle}} = 200$ independent random permutations of each ticker's weighted signal time series, recompute the portfolio PnL against the original realized returns, and compute the annualized Sharpe of each permutation. This yields a portfolio level null distribution $\{\mathrm{SR}_k^{\mathrm{shuf}}\}_{k=1}^{200}$, from which we extract:

- The mean $\overline{\mathrm{SR}^{\mathrm{shuf}}}$, which should be approximately zero by construction.
- The standard deviation σ_{shuf} across the 200 permutations, which is the noise floor of the portfolio Sharpe on a test window of this length.
- The one-sided p-value $\mathbb{P}(\mathrm{SR}^{\mathrm{shuf}} \geq \mathrm{SR}^{\mathrm{model}}_{\mathrm{portfolio}})$, the empirical fraction of permutations whose portfolio Sharpe equals or exceeds the model's.

The null hypothesis is

$$H_0^{\text{shuf}} : \text{the model's predicted signs are exchangeable with random signs.} \tag{8.2}$$

Rejection at level α corresponds to $p < \alpha$. The portfolio level test is the primary statistical significance claim of this thesis. Aggregation across tickers raises the signal to noise ratio of the test statistic, which is otherwise too low at the per ticker level to support reliable hypothesis testing on test windows of ~ 500 observations.

Per ticker shuffled signal Sharpe.

We additionally compute the per ticker permutation distribution by the same procedure, applied to each ticker's individual daily PnL series rather than to the portfolio. Per ticker p-values are reported descriptively in the results tables to support qualitative assessment of which tickers a given model handles well, but no per ticker accept/reject decision is made. The reasoning is methodological: with $\sim$ 500 observations per ticker and non-negligible serial dependence, individual rejection of H_0^{shuf} on a single ticker is suggestive rather than conclusive, and naive aggregation of per ticker p-values across the universe raises multiple testing concerns. We therefore report per ticker results in three tiers based on the descriptive p-value:

- *Strong evidence* ($p < 0.05$): the per ticker Sharpe is unlikely to have arisen by chance.
- *Suggestive* ($0.05 \leq p < 0.20$): the per ticker Sharpe is consistent with skill but not statistically dispositive given the test window length.
- *Inconclusive* ($p \geq 0.20$): no meaningful evidence of skill at the per ticker level.

These tiers are descriptive labels for reporting purposes, not significance tests. The headline statistical claim of this thesis is the portfolio level rejection described above.

Shuffled signal Sharpe for point forecast baselines.

For ARIMA, Ridge AR(L), and LSTM Fin, we report a single permutation draw SR_{shuf} as a descriptive benchmark rather than running the full permutation test. These baselines exist to

establish a comparison floor for the distributional models, and a single draw shuffle is sufficient for that purpose: we ask whether the baseline Sharpe is plausibly attributable to chance against the realized return distribution, not whether it formally rejects H_0^{shuf}. Single draw SR_{shuf} values for baselines should therefore be interpreted as one realization of the null distribution, not as a summary of it. In particular, baseline tickers where the single draw SR_{shuf} is comparable to the model Sharpe are flagged as cases where the test-set return distribution is sufficiently directionally asymmetric that the single draw null can land high. This is a property of the data, not evidence against the baseline.

Hit Rate.

The fraction of half sessions on which $\text{sgn}(s_t) = \text{sgn}(r_t)$, excluding sessions on which s_t or r_t is zero. A model with no directional information has an expected hit rate of 0.5. Our test sets contain roughly $N = 1{,}098$ half sessions, so the binomial standard error of the hit rate under the null is $\sqrt{0.25/N} \approx 0.015$. A hit rate of 0.53 is therefore about two standard errors above chance, and we use this threshold as a descriptive marker for which tickers a model handles well, not as a binary acceptance criterion.

Magnitude Accuracy.

We also report MAE and RMSE of $\bar{\hat{x}}_t$ against r_t. Directional accuracy alone is not sufficient, because a generator can score well on the hit rate while producing forecasts that look nothing like real returns. The mechanism is the tanh saturation built into the PnL^* and SR^* loss terms. Once $\bar{\hat{x}}_t$ is large enough that $\tanh(\kappa \cdot \bar{\hat{x}}_t)$ saturates to ± 1, increasing it further has no effect on the loss. A generator can therefore satisfy PnL^* and SR^* by producing strongly signed forecasts that point in the right direction, without learning the actual scale of r_t. For example, a forecast of $\bar{\hat{x}}_t = +0.5$ on a day when the realized return is $r_t = +0.008$ counts as a directional hit, even though the magnitude is two orders of magnitude too large. MAE and RMSE catch this. We require $\text{RMSE}(\bar{\hat{x}}_t, r_t) \leq 2\sigma(r_t) \approx 2 \times 10^{-2}$, which is the bar that the trivial forecaster

$\bar{\hat{x}}_t \equiv 0$ already meets.[6] A generator that fails this check is producing noise that happens to be correctly signed, not forecasts. This diagnostic matters most for the PnL*-only and SR*-only loss combinations, which lack the explicit magnitude anchor that MSE provides.

8.4.3 Ruling Out Distributional Collapse

GAN generators can silently collapse to a near deterministic output, to samples of a single sign, or to a fixed distribution that is independent of the conditioning window. Any of these can still produce reasonable looking aggregate numbers by accident. We check for four failure modes, using the scale of the underlying returns ($\sigma \approx 10^{-2}$) to set thresholds:

- Narrow_1: the standard deviation of one day's sample distribution falls below 2×10^{-4}, indicating that the generator is producing essentially the same value B times for that window.

- Narrow_m: the standard deviation of $\bar{\hat{x}}_t$ across the full test window falls below 2×10^{-4}, indicating that the generator produces essentially the same forecast every day.

- Positive_{mn} and Negative_{mn}: the fraction of test windows for which $\bar{\hat{x}}_t > 0$, respectively < 0. Healthy generators produce fractions near 0.5. Values near 1 or 0 indicate sign degenerate output.

The threshold of 2×10^{-4} is two orders of magnitude below the typical scale of a half day excess return, so any generator producing meaningful variation will be comfortably above it. These four flags collectively replace the single mode collapse threshold ϵ used by Vuletic et al. [33], which compares only the standard deviation of the generated samples to a fixed cutoff.

8.5 Selection Rule and Acceptance Criteria

The metrics defined in the preceding sections play three distinct roles, and it is worth separating them explicitly.

[6] Since $\sigma(r_t) \approx 10^{-2}$ implies $\text{RMSE}(0, r_t) = \sigma(r_t) < 2 \times 10^{-2}$.

8.5.1 Selection criterion

For each ticker and seed, the loss combination chosen for test set reporting is the one that maximizes the validation weighted signal Sharpe ratio. This follows Vuletic et al. [33] and is the only mechanism by which a single number is used to rank loss combinations against one another.

8.5.2 Statistical Significance Gate

The headline statistical significance claim is made at the portfolio level. A model is considered to have produced a statistically significant signal at level α when the portfolio level shuffled signal Sharpe rejects H_0^{shuf} at $p < \alpha$, where the portfolio is the equal weighted aggregate of per ticker daily PnL across the 28 stock universe. We adopt $\alpha = 0.05$ as the headline threshold.

8.5.3 Distributional Sanity Gate

A model that passes the portfolio level statistical test must additionally not exhibit distributional collapse, as measured by the four flags described in the preceding section. A model that triggers any of these flags has produced numerically favorable aggregate statistics through a degenerate generator and is reported as a failure of the sanity gate even when the portfolio Sharpe is statistically significant.

The remaining metrics (per ticker p-values, hit rate, magnitude RMSE, forecast target correlation, constant sign Sharpe) are reported descriptively in the per ticker results tables. They support qualitative interpretation of which tickers a model handles well and why, but they do not enter a binary acceptance decision.

8.5.4 Why the headline claim is at the portfolio level?

Per ticker test windows contain on the order of 500 observations with non-negligible serial dependence, so individual rejection of H_0^{shuf} at $p < 0.05$ on a single ticker is suggestive rather than conclusive. Per ticker hit rates above 0.53 are similarly only about two standard errors

above chance, which is too small a margin to support a binary pass/fail decision per ticker. Aggregating across the 28 tickers raises the signal to noise ratio of the portfolio Sharpe and makes the resulting permutation test substantially more powerful. The portfolio level test is therefore the unit at which we test whether the model produces a non-trivial signal beyond chance, and it is also the unit at which we report the headline result of the project.

8.6 Evaluation Protocol and Aggregation for FinTimeGAN and FinGAN

The evaluation framework defined in the preceding sections applies to any model that produces forecasts under the trading rules of Section 8.2. For FinTimeGAN specifically, the framework must be replicated across multiple seeds and loss combinations to control for the well known sensitivity of GAN training to initialization. This section describes the protocol used to do so.

The metrics defined in the preceding sections are computed per trained model. An individual FinTimeGAN, however, is not a stable object. Its behavior depends on the random seed used during training, on the ticker it was trained on, and on which of the ten loss combinations discussed in Chapter 6 was used during Phase 2. Drawing conclusions from a single model would therefore conflate genuine modeling effects with the noise inherent in GAN training. The protocol below describes how evaluations are replicated and aggregated so that our conclusions reflect the former rather than the latter.

8.6.1 Per seed protocol

For each ticker and seed, the full training pipeline is executed once. Phase 1 pretrains a single Embedder and Recovery network on the reconstruction objective. The gradient norm matching procedure of Algorithm 1 then calibrates the loss coefficients $\alpha, \beta, \gamma, \delta$ from a shared post-calibration checkpoint. Ten branched copies of this checkpoint are then fine tuned independently, one for each loss combination. This branched design, rather than a chained one in which the output of one loss combination is used as the starting point for the next, ensures that the only difference between the ten final generators is the loss combination itself. Each of these ten

generators is then evaluated on both the validation and test splits by the procedure of Sections 8.2 through 8.4, producing ten rows in the results table for that ticker and seed. The combination with the highest validation weighted signal Sharpe is then reported as the selected loss for that ticker and seed, following the rule of Section **??**.

8.6.2 Seed aggregation

GAN training is sensitive to initialization. Two seeds with identical hyperparameters can produce final models whose weighted Sharpe ratios differ by a full unit. To prevent this variance from driving our conclusions, each ticker and loss combination is trained under five independent seeds, and the five resulting metric values are reduced to a mean and a standard deviation. A loss combination that achieves a mean Sharpe of 0.8 with standard deviation 0.1 across seeds is treated very differently from one that achieves 0.8 with standard deviation 0.9, even though their point estimates are identical. Because five seeds remain a small sample, the standard deviation reported is indicative rather than a formal confidence interval, and we do not perform hypothesis tests on it.

This multi seed protocol is an extension over Vuletic et al. [33], who report a single training run per ticker and loss combination. Reporting seed variance allows us to distinguish loss combinations that are genuinely effective from those whose apparent effectiveness depends on a fortunate initialization.

8.6.3 Ticker aggregation

A trading strategy that works on one stock and fails on nine is not a trading strategy. To argue that a loss combination produces useful signal in general, we aggregate across tickers. For each loss combination we report the median seed averaged metric across the set of tickers covered by the experiment. The median is preferred to the mean because Sharpe ratios have heavy tails: a single ticker whose test period contained an unusually favorable regime can dominate a cross ticker average, which would misrepresent what the loss combination typically achieves. The

median is also what Vuletic et al. [33] report, which makes our numbers directly comparable to theirs.

In addition to the median, we report the interquartile range of the seed averaged metric across tickers. A loss combination with median Sharpe 0.7 and interquartile range $[0.6, 0.8]$ is substantively different from one with the same median and range $[0.1, 1.3]$, even though they have the same headline number.

8.6.4 Cross combination analysis

Beyond per combination performance, we examine whether the ten loss combinations produce *diversified* signals or *redundant* ones. For each pair of loss combinations (a, b) on a given ticker, we compute the Pearson correlation of their daily PnL series on the test set. Averaging across tickers yields a 10×10 correlation matrix whose off-diagonal entries summarize the extent to which different loss choices induce different trading behavior. A correlation near unity indicates that the two losses, despite their different training objectives, converge to functionally similar generators. A correlation near zero, or negative, indicates that they extract different features of the data and would in principle be combinable in a portfolio sense. This matrix is presented in Chapter 9 and motivates the discussion of which loss combinations carry distinct information.

Chapter 9

Results

This chapter reports the test window performance of the four models introduced in Chapter 5: ARIMA two parity, Ridge AR(10), LSTM-Fin, and FinGAN, together with the FinTimeGAN extension introduced in this project. All models are evaluated on a fixed test window of 549 trading days from 2019-10-30 to 2021-12-31, using the trading rules and diagnostics defined in Chapter8.

Each subsection follows a similar structure. The headline portfolio result is reported first. Per ticker test Sharpes under the validation best loss combination are then unpacked, with attention to which tickers the model handles well, which it struggles on, and whether the per ticker pattern is consistent with sector level structure or driven by ticker specific behavior. For the distributional models, additional diagnostics on loss combination behavior, selection diversity, and distributional collapse are reported. Cross model comparison is treated in Section9.7.

A note on framing. The headline statistical claim of this project is made at the portfolio level, where aggregation across tickers raises the signal to noise ratio of the permutation test enough to support formal hypothesis testing on a $\sim$ 549-day test window. Per ticker results are reported descriptively to support qualitative assessment of where each model succeeds and fails, but are not used for accept/reject decisions. The full justification for this choice is given in Section8.5.

9.1 The Universe

The universe consists of 28 tickers (stocks) drawn across nine GICS [1] sectors, each forecast on its ETF excess return (Relative Excess Return 4.2.2). Each ticker has a benchmark sector ETF assigned by the GICS classification. The benchmarks are the SPDR [2] sector ETFs (XLB, XLE, XLF, XLI, XLK, XLP, XLU, XLV, XLY). All stocks and their corresponding benchmark ETFs are listed in Table 9.1. The nine sector ETFs serve two roles in this project: they are the benchmarks used to compute ETF excess returns for the individual company stocks, and on the Vuletic comparable universe of 22 stocks plus 9 ETFs (reported in Section 9.7.1) they are also tradable instruments contributing their own daily PnL series to the portfolio aggregate.

Ticker	Company	Ticker	Company	Ticker	Company
AMZN	Amazon.com	FDX	FedEx	OXY	Occidental Petroleum
APA	APA Corp.	GD	General Dynamics	PEP	PepsiCo
BLK	BlackRock	GS	Goldman Sachs	PFE	Pfizer
CCL	Carnival	HD	Home Depot	SYY	Sysco
CERN	Cerner	HUM	Humana	TER	Teradyne
CL	Colgate-Palmolive	IBM	IBM	TROW	T. Rowe Price
DTE	DTE Energy	IP	International Paper	TSN	Tyson Foods
EBAY	eBay	KO	Coca-Cola	WEC	WEC Energy
ECL	Ecolab	NKE	Nike	WFC	Wells Fargo
EL	Estée Lauder				

Table 9.1: The 28 single stocks used in the Results chapter, with corresponding ticker symbols.

The sectoral composition of the universe is relevant to the interpretation of cross sectional results. Consumer staples, utilities, and health care tickers are conventionally referred to as defensive sectors, because their underlying businesses (beverages, household products, electricity, hospitals, pharmaceuticals) generate revenue that is relatively insensitive to the broader economic cycle. Defensive tickers tend to have lower volatility and more stable cash flows than cyclical or discretionary tickers.

[1] A hierarchical, four level framework that organizes public firms into 11 major sectors according to their main business focus. For more information visit `https://www.fidelity.com/learning-center/trading-investing/markets-sectors/global-industry-classification-standard`

[2] SPDR Sector ETFs, or "Select Sector SPDRs" are State Street managed funds that split the S&P 500 into 11 specific industry categories.

Consumer discretionary covers companies whose revenue depends on disposable income (apparel, e-commerce, travel, restaurants, home improvement). Discretionary tickers are more cyclical and more news driven than defensive tickers. Information technology and financial sector sit between these two groups in cyclicality, with substantial dispersion within each. The Results chapter will refer to this taxonomy when interpreting cross sectional patterns in model performance.

9.2 FinGAN Results

FinGAN is run on two universes. The first is the 28 stock universe used by ARIMA, Ridge, and LSTM-Fin, providing an apples to apples comparison against the baselines. The second is the Vuletic comparable universe of 22 stocks plus 9 sector ETFs found at Section 9.7.1, providing a like for like comparison against Vuletic et al.[33]'s published portfolio Sharpe. Each (ticker, seed) pair trains all ten loss combinations from a shared post calibration checkpoint under the independent branch protocol of Chapter 7. The validation best loss combination is selected by validation weighted signal Sharpe per Section 8.5. Five seeds per ticker and the results per ticker is the mean of the seeds.

9.2.1 The 28 stock universe

Headline. FinGAN's validation best portfolio achieves a weighted signal Sharpe of +2.022 over 549 trading days, with a one sided p-value below 10^{-4} under the 200 permutation null. Cumulative portfolio PnL is +1,291.5 basis points. The model passes the portfolio level statistical significance gate of Section 8.5 by a wide margin. The model's Sharpe is more than three standard deviations above the null mean, and the 95th percentile of the null lands at +0.942 which is less than half of the model's value.

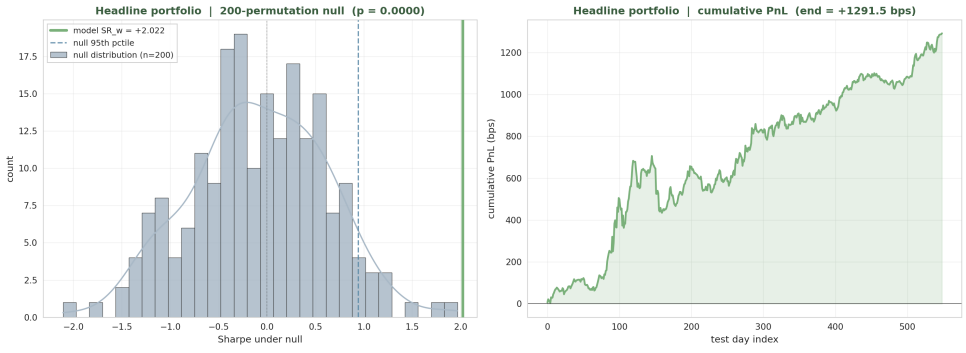


Figure 9.1: Left: 200 permutation null distribution of the portfolio weighted signal Sharpe with the model's Sharpe marked. Right: cumulative portfolio PnL across the test window.

Loss combination behavioral clusters. The ten loss combinations do not produce ten independent trading signals. Cross combination Pearson correlation of daily PnL, averaged across

(ticker, seed), separates them into three behavioral groups: {BCE, MSE, PnL, PnL+MSE} (0.88 to 0.96 within cluster), {PnL+SR, PnL+MSE+SR, SR, SR+MSE} (≥ 0.99 within cluster), and {PnL+STD, PnL+MSE+STD} (0.84 within cluster).

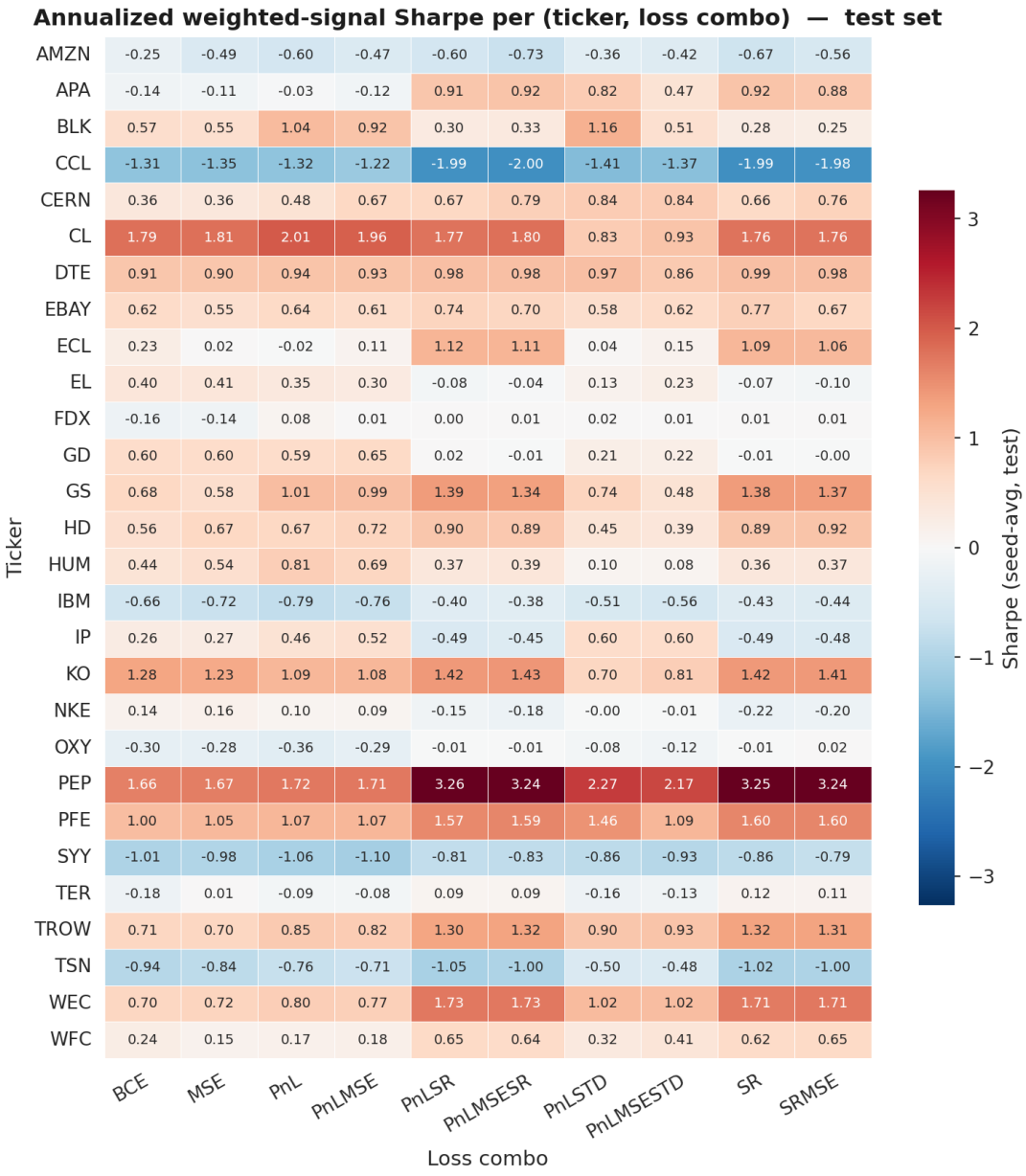


Figure 9.2: Test Sharpe per (ticker, loss combination) under the weighted signal rule, averaged across 5 seeds. The strongest cells concentrate in the SR family combinations.

Cross cluster correlations sit between 0.30 and 0.45. The substantive finding is that the SR-family is so internally correlated as to be effectively one signal: adding MSE or PnL to an SR

anchored loss makes a smaller behavioral difference than adding STD. From a methodological standpoint, this argues for treating the ten combinations as approximately three families rather than ten independent runs when interpreting selection results.

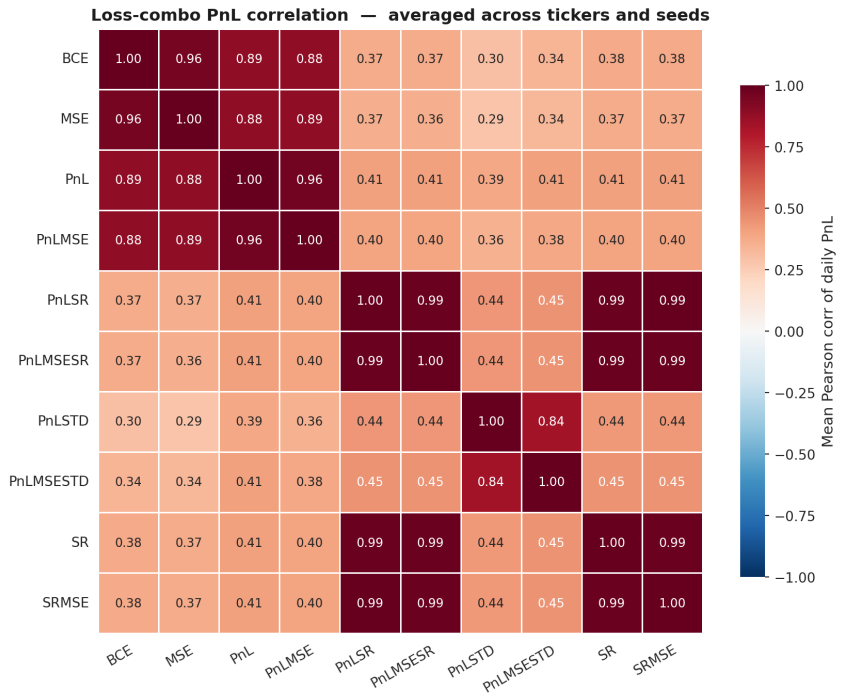


Figure 9.3: Pearson correlation of daily PnL between pairs of loss combinations, averaged across (ticker, seed) runs. The SR-family is internally correlated above 0.99, behaving as a single signal.

Where the gain over baselines comes from. FinGAN's +2.022 exceeds ARIMA's +0.642 and Ridge's +0.227 on the same universe. The cross sectional pattern is broadly the same as the linear baselines: defensive and utility tickers lead, discretionary tickers trail. The gain is therefore not from finding signals where the linear models found none, but from extracting the existing signal more cleanly. Two specific properties make this visible. First, the per ticker spread widens (PEP at +2.82 versus Ridge's +2.69 and ARIMA's parity corrected −0.87 on the same ticker; CCL at −1.31 versus Ridge's −1.31). Second, the portfolio Sharpe of +2.022 exceeds the per ticker mean of +0.65 on this universe by enough that the diversification benefit is itself a finding. The per ticker signals are weakly correlated, so summing them across the universe raises the ratio of signal to noise meaningfully more than any individual ticker contributes.

Selection. The validation best rule produces a spread across all three families rather than a single dominant combination, with PnL+STD and PnL+MSE+STD selected most frequently across the 28 tickers ($\sim$ 15 ticker seed cells each). This is the cleanest evidence for the seed to seed validation noise discussed in Section 8.5: if there were one objectively best combination, the validation best rule would converge to it across seeds. The fact that selections spread across families is one reason this project's headline statistical claim is at the portfolio level rather than per ticker.

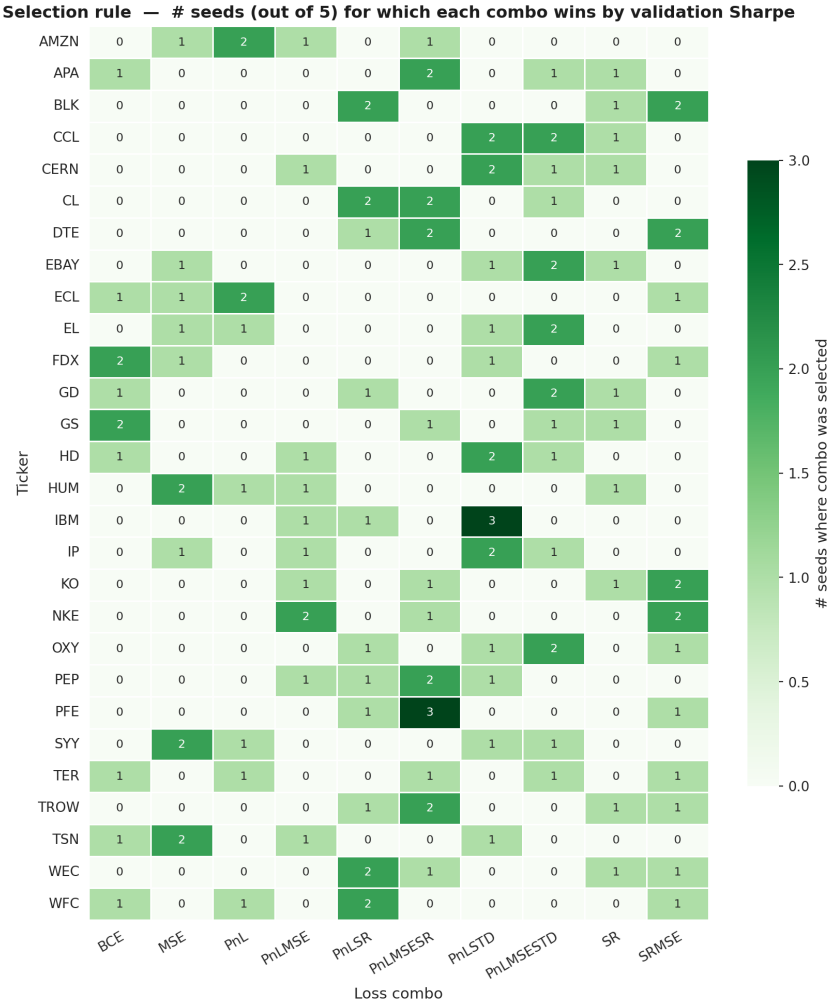


Figure 9.4: Number of seeds (out of 5) for which each loss combination is selected by the validation best rule, per ticker. No single combination dominates the selection across the universe.

Hit rate and confidence weighting. Cross ticker hit rates cluster tightly around 0.50 across most loss combinations, with a small subset of 19 cells (out of 280) exceeding the 0.53 threshold

defined in Section 8.4.2. These exceeding cells concentrate on CL, PEP, KO, and WEC under SR-family and PnL+MSE-family combinations as shown in Figure 9.6. The cells exceeding 0.53 produce meaningful directional edge above chance, but for the bulk of the universe directional accuracy is at the noise floor and contributes little to daily PnL.

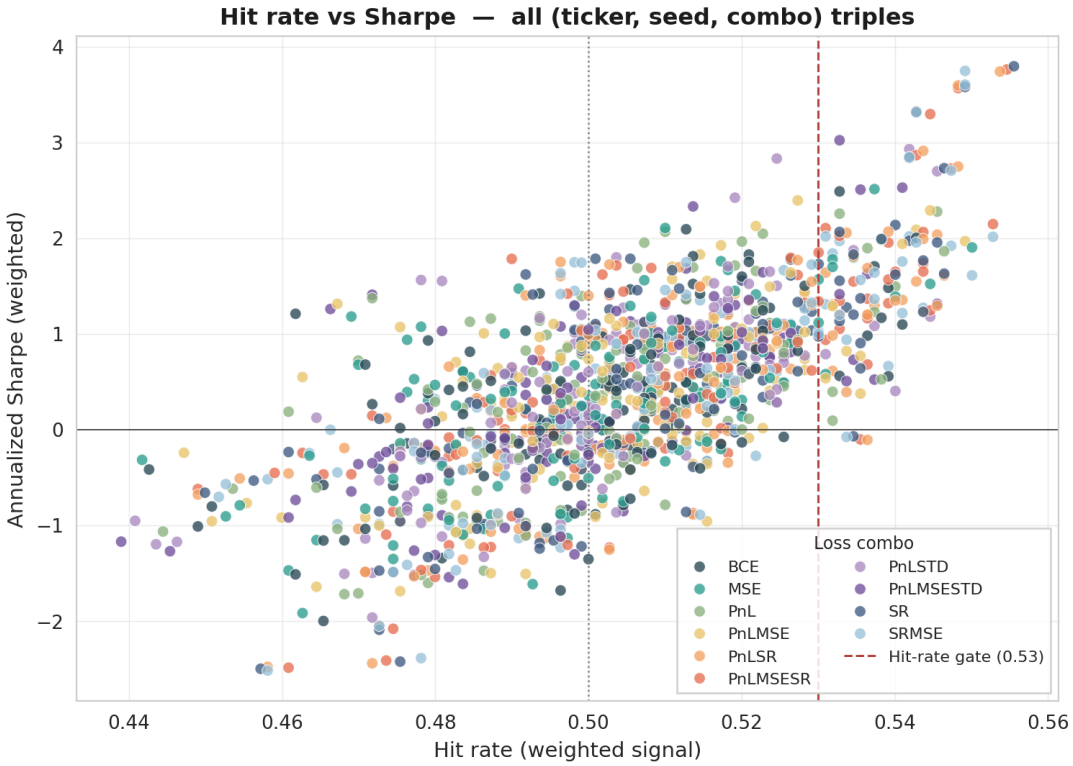


Figure 9.5: The fraction of half session that the model's directional forecast matches the realized return. The pure chance is 0.50 and 0.53 is to distinguish the signals that passes the pure chance point.

Two channels therefore drive the portfolio Sharpe. The directional channel works on a small subset of forecastable tickers. The magnitude channels are ,first, confidence weighted position sizing which applies universally. In which the half session forecast with high absolute magnitude contributes proportionally more to PnL than one with low magnitude, regardless of whether the cell's hit rate clears the noise floor. The weighted signal rule scales position size by magnitude; the Hard signal rule reduces every forecast to a sign and discards this information.

This is the structural reason FinGAN beats LSTM-Fin in places where their per ticker Sharpes look comparable. LSTM-Fin's point forecast encodes confidence only in its magnitude; Fin-

GAN's distributional output encodes confidence in both the mean magnitude and the width of the forecast distribution around it. The richer confidence encoding plausibly explains why distributional models systematically extract more value from the magnitude channel in this universe.

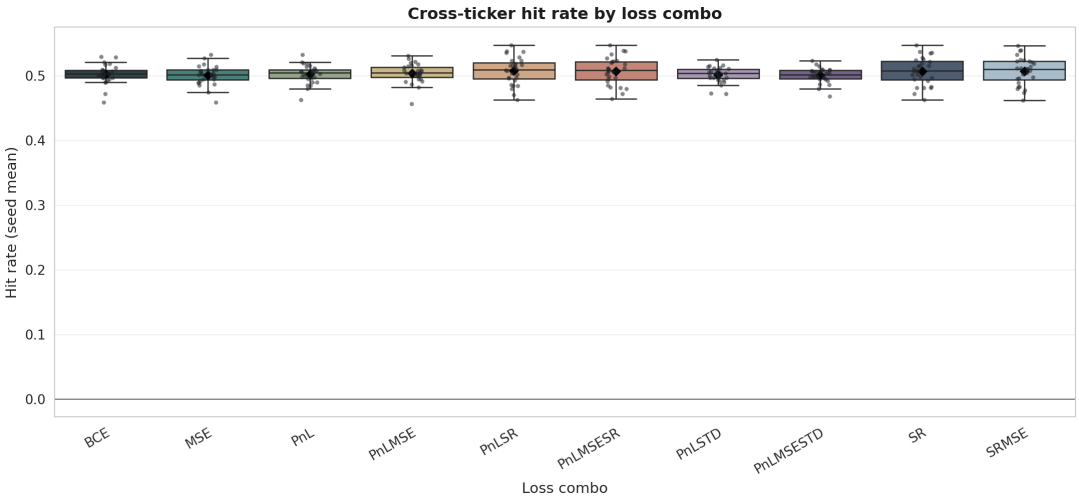


Figure 9.6: Distribution of per-ticker test hit rates by loss combination. Median hit rates cluster at chance (0.50) across all combinations.

Per cell acceptance. As a descriptive companion to the portfolio level acceptance gate, we report a per cell count of how many (ticker, loss combination) cells pass the three diagnostic criteria of Section 8.4 simultaneously: per ticker permutation $p < 0.05$, hit rate above 0.53, and no distributional collapse flag. Of the 280 cells, 19 pass all three. These concentrate on four tickers (CL, PEP, KO, WEC) and on the SR-family and PnL+MSE-family of loss combinations. This patter is consistent with both the cross sectional and the loss combination findings above. Two cells fail all three: AMZN under PnL+STD and AMZN under PnL+MSE+STD. The diagnostic does not feed an accept/reject decision and so it identifies where the model's signal is strongest by all three criteria at once.

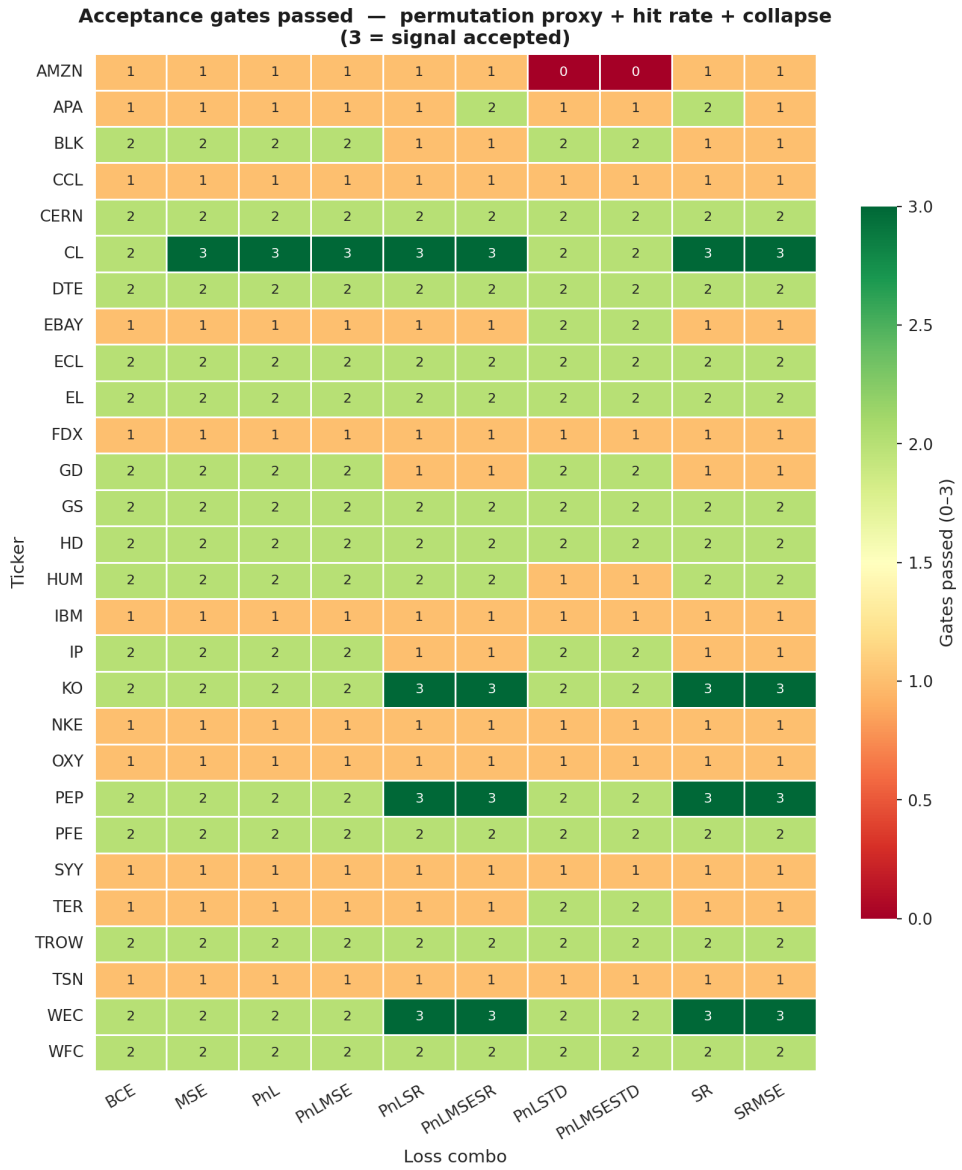


Figure 9.7: Descriptive count of diagnostic criteria passed (out of three: per ticker permutation $p < 0.05$, hit rate above 0.53, no distributional collapse) per (ticker, loss combination) cell. Strong cells concentrate on CL, PEP, KO, and WEC; the only cells failing all three are AMZN under PnL+STD and PnL+MSE+STD.

Distributional collapse. The four collapse flags of Section 8.4 are essentially never triggered for eight of the ten loss combinations. PnL+STD and PnL+MSE+STD trigger the narrow distribution flag at $\sim 3\%$ of (ticker, seed) runs, and exhibit substantial sign degeneracy spread in the Pos_{mn} statistic, with whiskers extending across nearly the full $[0, 1]$ range (Figure9.8). The mechanism is plausibly an artifact of variance anchored training: pressuring the generator

to produce wide distributions can incentivize degenerate solutions where samples concentrate at the distributional extremes, raising apparent variance without preserving healthy distribution shape.

At the model level, the collapse rate is too low to trigger the binary distributional sanity gate, but the within combination concentration on the STD family identifies these two combinations as more fragile than the others.

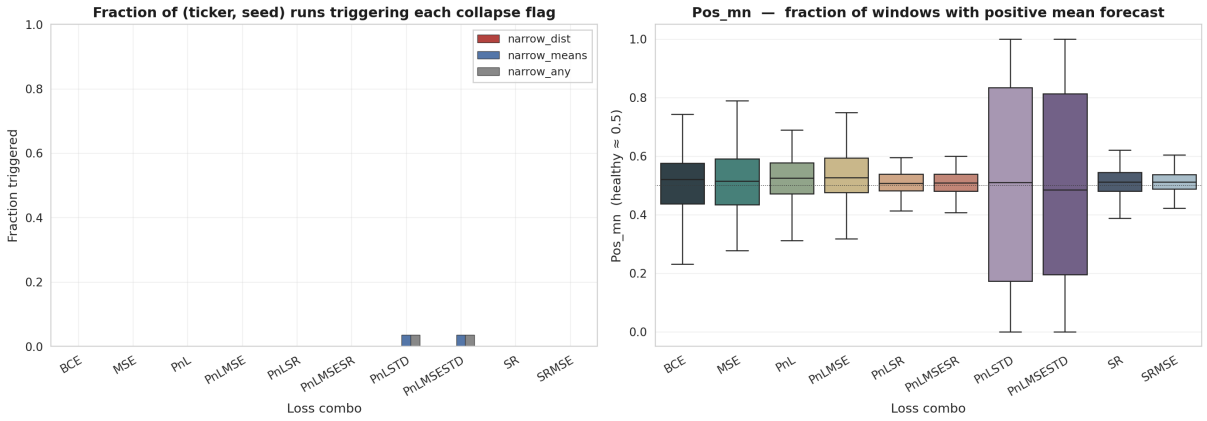


Figure 9.8: Left: fraction of (ticker, seed) runs triggering each distributional collapse flag. Right: distribution of Pos_{mn} per loss combination. PnL+STD and PnL+MSE+STD are identified as more fragile than the other eight combinations.

9.3 FinTimeGAN Results

This section reports the results for the compressed mode of FinTimeGAN, introduced in Section 5.7. The compressed mode conditions both the generator and the discriminator on the final latent vector produced by the Embedder when it encodes a conditioning window of past returns. The Embedder and Recovery networks are pretrained on the reconstruction objective and frozen during adversarial training, which is run with the same ten loss combinations and validation best selection rule used for FinGAN.

Headline. On the 27 stock universe, the validation best portfolio achieves a weighted signal Sharpe of +1.083 over the 549 day test window. The 200 permutation null produces a p value of 0.040, just below the 0.05 threshold of Section 8.5. Cumulative portfolio PnL ends at +522.8 basis points. The model passes the portfolio level statistical significance gate, but with substantially less margin than FinGAN on the same universe ($p < 10^{-4}$ versus $p = 0.040$ here): the model Sharpe lands close to the 95th percentile of the null rather than well into its right tail.

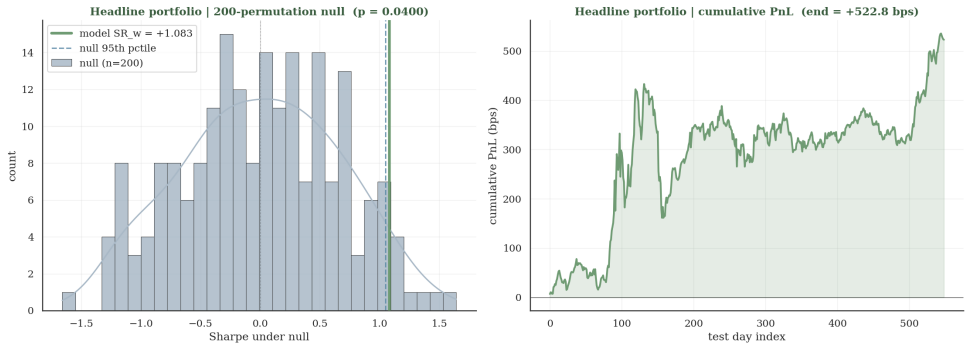


Figure 9.1: Left: 200 permutation null distribution of the portfolio weighted signal Sharpe. The model's Sharpe of +1.083 sits at the 95th percentile of the null, with $p = 0.040$. Right: cumulative portfolio PnL across the 549-day test window, ending at +522.8 bps.

Cross sectional pattern. Per ticker weighted signal Sharpes range from +2.75 (PEP) at the top to −1.36 (TSN) at the bottom. The top of the ranking is dominated by defensive, utility, and health care names: PEP (+2.75), KO (+1.78), CERN (+1.72), WEC (+1.40), and PFE (+1.30) all sit comfortably above +1.0. The bottom is occupied by a more heterogeneous group of

tickers in food distribution (TSN, SYY), financials (BLK), and technology and industrials (IBM, FDX) where short term forecastability is weak across all model families. The same defensive leads discretionary trails pattern observed under FinGAN therefore recurs here, despite the architectural change to latent space conditioning. This consistency across architectures supports the interpretation developed in the Cross Model Comparison section that the cross sectional pattern is a property of the universe rather than of any particular model's inductive bias.

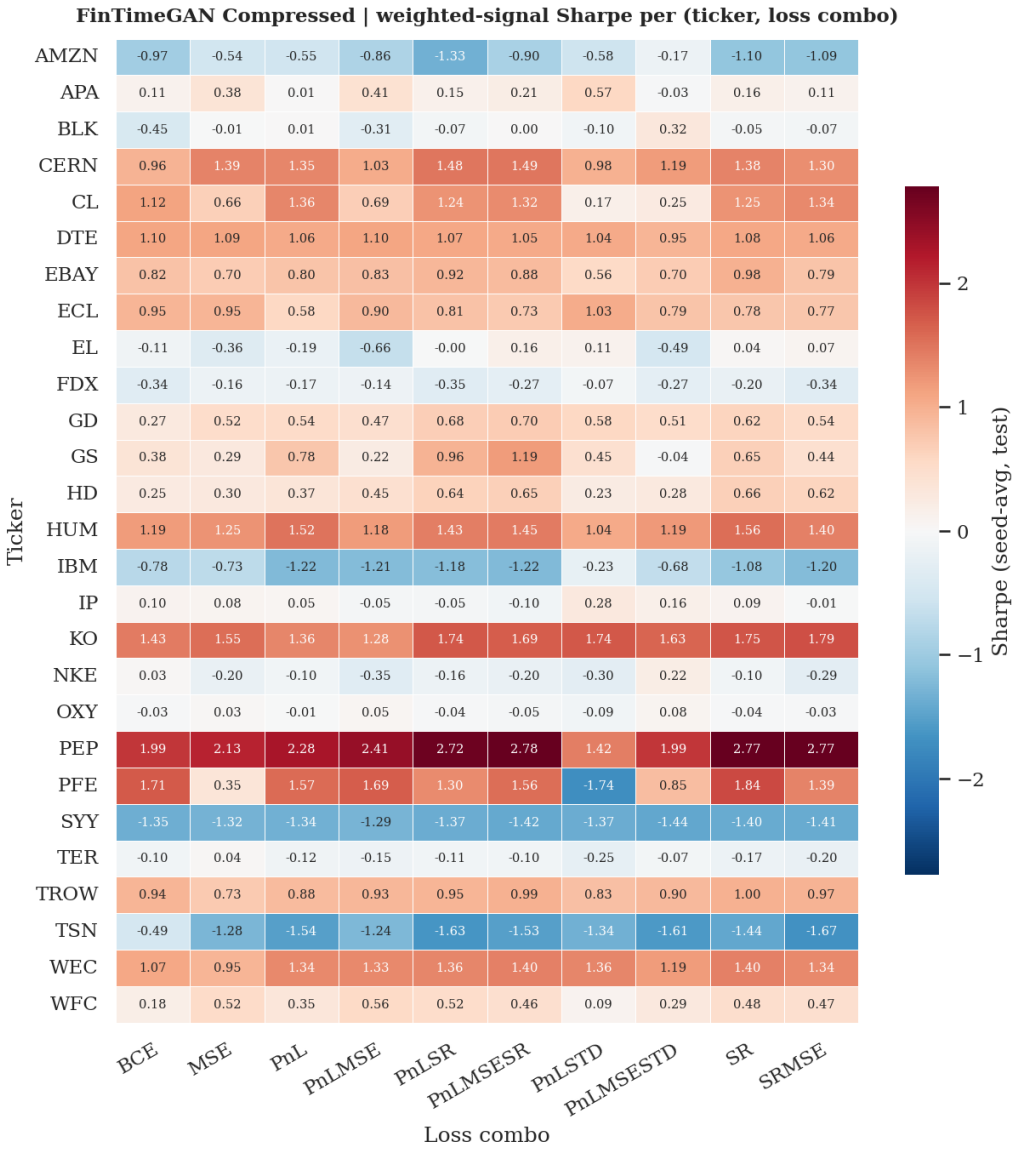


Figure 9.2: Per (ticker, loss combination) weighted signal test Sharpe for FinTimeGAN compressed mode on the 27 stock universe, averaged across 5 seeds.

Per ticker scoreboard. The validation best loss combination per ticker, along with the headline diagnostics, is shown in Figure 9.3. PnL+MSE and PnL+MSE+SR are each selected on 5 tickers, BCE on 4, and the remaining 13 tickers spread across the other seven combinations. The SR family is favored on most top performing tickers (KO, WEC, and DTE all pick PnL+MSE+SR), with PEP as the notable exception, selecting MSE. The selection pattern broadly mirrors FinGAN's, where SR-family combinations also dominated selection on the strongest tickers, suggesting that the latent space conditioning does not alter which loss combinations work best on which tickers, only the magnitude of their effects.

FinTimeGAN Compressed | per-ticker headline scoreboard | per-(ticker, seed) val-best, median across 5 seeds

ticker	modal combo	n	SR_w	SR_hard	SR_const	hit_w	hit_hard	RMSE	MAE	Corr
PEP	MSE	5	+2.754	+3.283	+0.037	0.5610	0.5528	0.00606	0.00410	+0.145
KO	PnLMSESR	4	+1.779	+1.694	-0.389	0.5437	0.5446	0.00777	0.00494	+0.102
CERN	PnLMSESR	3	+1.722	+1.340	-0.234	0.5255	0.5328	0.00982	0.00597	+0.082
WEC	PnLMSESR	4	+1.399	+1.598	-0.104	0.5237	0.5282	0.00764	0.00470	+0.082
PFE	PnLSR	1	+1.304	+0.841	+0.286	0.5009	0.5009	0.01321	0.00890	+0.109
HUM	PnLMSESTD	4	+1.159	+0.966	+0.097	0.5027	0.5200	0.01158	0.00707	+0.063
TROW	PnLMSE	5	+1.158	+1.326	+0.433	0.5027	0.5109	0.01280	0.00880	+0.029
CL	PnLSTD	4	+1.096	+0.967	-0.045	0.4973	0.5046	0.00662	0.00422	+0.088
DTE	PnLMSESR	4	+1.094	+0.553	+0.049	0.5273	0.5209	0.01019	0.00587	+0.080
EBAY	SR	4	+0.935	+0.643	+0.149	0.5009	0.5064	0.01483	0.00935	+0.058
GD	PnLMSE	4	+0.914	+0.072	-0.308	0.5100	0.4945	0.00750	0.00518	+0.041
EL	PnLSTD	4	+0.760	+0.727	+0.727	0.5237	0.5209	0.01197	0.00768	+0.020
ECL	BCE	5	+0.697	-0.312	-0.742	0.4882	0.4991	0.00948	0.00583	+0.080
IP	BCE	4	+0.599	-0.164	-0.598	0.5282	0.5191	0.01074	0.00694	+0.015
WFC	PnLMSESR	4	+0.526	-0.039	-0.745	0.5073	0.5091	0.01080	0.00694	+0.031
HD	PnLMSE	5	+0.502	+0.188	+0.166	0.5009	0.5009	0.00964	0.00680	+0.019
GS	PnLMSE	5	+0.355	+0.583	+0.676	0.4809	0.5055	0.00779	0.00508	+0.049
OXY	PnLMSE	4	+0.167	-0.037	+0.037	0.5046	0.5055	0.02106	0.01326	+0.013
TER	MSE	3	+0.163	+0.649	+0.279	0.5246	0.5264	0.01421	0.00970	-0.005
APA	PnLMSESTD	3	+0.067	+0.320	+0.613	0.4991	0.5118	0.02660	0.01632	+0.047
NKE	PnLMSE	4	-0.136	+0.163	+0.163	0.5046	0.5046	0.01101	0.00702	-0.035
AMZN	PnLMSESR	4	-0.263	-0.310	+0.165	0.4954	0.4845	0.01077	0.00721	-0.070
FDX	BCE	4	-0.327	-0.412	+0.283	0.4909	0.4909	0.01326	0.00795	-0.019
IBM	PnLSTD	3	-0.445	-1.021	-1.034	0.4991	0.4617	0.01223	0.00829	-0.046
BLK	SRMSE	4	-0.564	-0.359	+0.692	0.4800	0.4900	0.01005	0.00648	-0.017
SYY	BCE	5	-1.328	-1.163	-0.275	0.4918	0.4945	0.02004	0.01176	-0.092
TSN	BCE	4	-1.363	-0.523	-0.350	0.4982	0.4845	0.01378	0.00798	-0.116

Figure 9.3: Per ticker headline scoreboard for FinTimeGAN compressed mode.

Loss combination summary. Cross ticker median weighted signal Sharpe by loss combination shows the same SR family dominance as FinGAN. The strongest medians are

PnL+MSE+SR at +0.65, PnLSR at +0.64, and SR at +0.62. The STD family combinations are the weakest (PnL+STD median +0.23, PnL+MSE+STD median +0.28), and BCE alone also underperforms at +0.25. The contrast between the SR family and STD family is consistent with the observation of Vuletic et al. [33] that the two encode the same information about return relative to volatility but have different gradients, which can produce different forecast distributions even when both objectives are minimized.

Median hit rates cluster at 0.50 across all loss combinations and forecast target Pearson correlations are uniformly small ($|\rho| < 0.04$). Both diagnostics indicate that the model is not producing point forecast accuracy in the conventional sense. The model's positive Sharpe must therefore come from the magnitude channel rather than from directional prediction, the same two channel pattern observed in FinGAN.

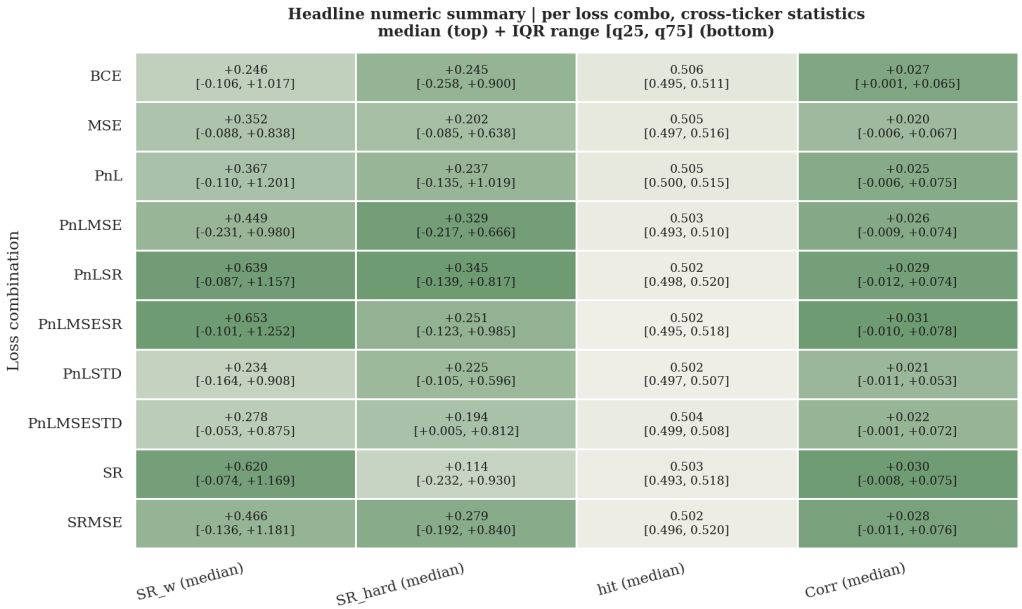


Figure 9.4: Cross ticker statistics by loss combination for FinTimeGAN compressed mode. Each cell reports the median across tickers (top) and the IQR range (bottom). Columns are weighted signal Sharpe, hard signal Sharpe, hit rate, and forecast target correlation.

Sector level decomposition with ETF buy and hold. Comparing per sector cumulative PnL against the corresponding sector ETF buy and hold trajectory (Figure9.5) clarifies where the model adds value beyond passive sector exposure and where it does not. The model's individual

stock PnL trajectories exceed the buy and hold trajectory in XLP (Consumer Staples), XLU (Utilities), (Health Care), and XLF (Financials). In XLK (Information Technology) and XLY (Consumer Discretionary), passive buy and hold substantially outperforms the model's stock trading. In XLE (Energy), buy and hold suffers large drawdowns that the model avoids: the model finishes near zero on a sector where pure long exposure would have lost money over the test window. The pattern indicates that the model's value is not uniform across the universe but concentrates on sectors where short term forecastability persists, and that the value takes two forms: outperformance on defensive and utility sectors, and drawdown avoidance on volatile cyclical sectors.

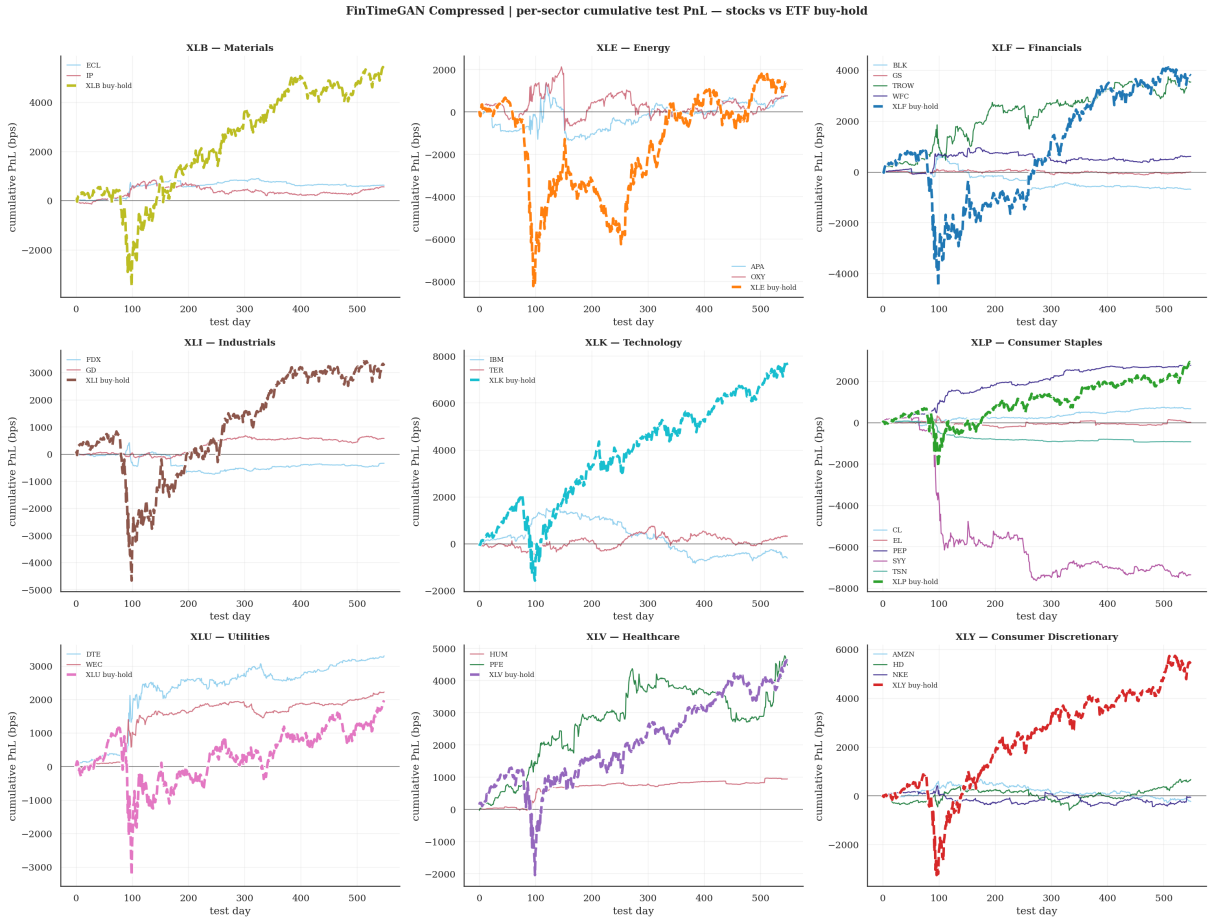


Figure 9.5: Per-sector cumulative test PnL for FinTimeGAN compressed mode. Solid lines: model PnL by ticker within each sector. Dashed lines: passive buy-and-hold trajectory of the sector ETF, shown for comparison.

Where the gap comes from. On the 27 stock universe, FinGAN achieves a portfolio Sharpe of +2.022 at $p < 10^{-4}$, roughly twice FinTimeGAN compressed's +1.083 at $p = 0.040$. The architectural extension to a learned latent space therefore does not improve trading performance in this mode, and the most plausible explanation is that the latent compression itself is the limiting factor. Conditioning the adversarial game on a single latent vector, rather than directly on the return space window, collapses the temporal context of the conditioning input into a fixed dimensional summary before adversarial training begins. This summary has to encode the recent volatility regime, the recent directional trajectory, and any other temporal features the discriminator and trading utility losses rely on, all in a single vector. Some of that information is plausibly lost in the compression, and the model's reduced performance is consistent with the loss being substantial enough to matter at the trading utility level. The natural follow up is the attention mode of FinTimeGAN, which conditions on the full latent sequence rather than a single vector and would preserve the temporal structure that the compressed mode discards. Preliminary work on the attention mode was started during the project but the runs were not finalized within the time budget. The direction is worth pursuing in follow up work.

9.4 LSTM-Fin Results

LSTM-Fin is a non distributional point forecast model trained under six loss combinations from Vuletic et al. [33]'s LSTM-Fin specification: PnL, PnL+STD, PnL+SR, STD, SR, and MSE. The BCE derived combinations are excluded because they require an adversarial discriminator that LSTM-Fin does not have. Selection is by validation Sharpe per (ticker, seed). LSTM-Fin produces a single point forecast, so only the hard signal trading rule of Section 8.2 applies.

Headline. On the 28 stock universe, LSTM-Fin's validation best portfolio achieves a weighted signal Sharpe of +1.370 on the 549 day test window. The 200 permutation null produces a one sided p-value of 0.0000, well below the 0.05 threshold (the null max is +1.366, just below the model's +1.370). Cumulative portfolio PnL is +1,828.4 basis points, with a mean daily PnL of +3.330 bps. The model passes the portfolio level statistical significance gate of Section 8.5 cleanly.

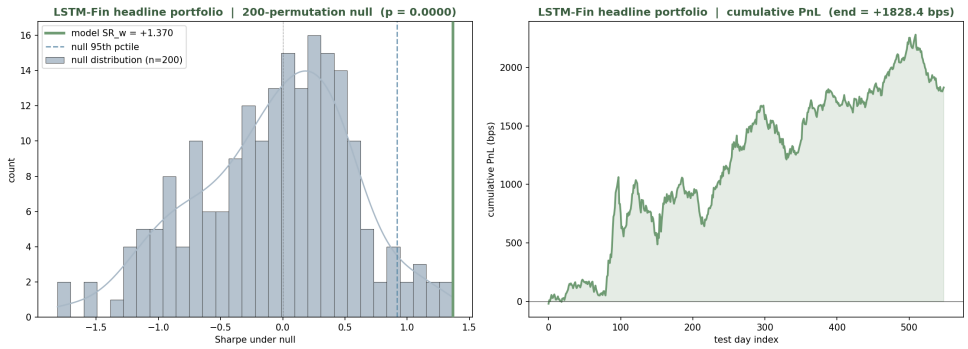


Figure 9.1: Left: 200 permutation null distribution with the model Sharpe marked. Right: cumulative portfolio PnL across the test window. The model rejects H_0^{shuf} at $p < 0.05$.

Cross sectional pattern. The per ticker Sharpes range from +2.18 (KO) at the top to −1.06 (CERN) at the bottom. The same ranking pattern that emerged from the linear baselines is preserved: defensive and utility tickers lead the distribution, while consumer discretionary and information technology tickers trail. KO, GS, PEP, OXY, and WEC occupy the top of the ranking; CERN, NKE, SYY, IP, and IBM occupy the bottom. The single headline ticker shifts across models, with GD under ARIMA, PEP under Ridge, and KO under LSTM-Fin. This

shift suggests that different architectures find slightly different decompositions of the underlying cross sectional signal even when they agree on the overall ranking.

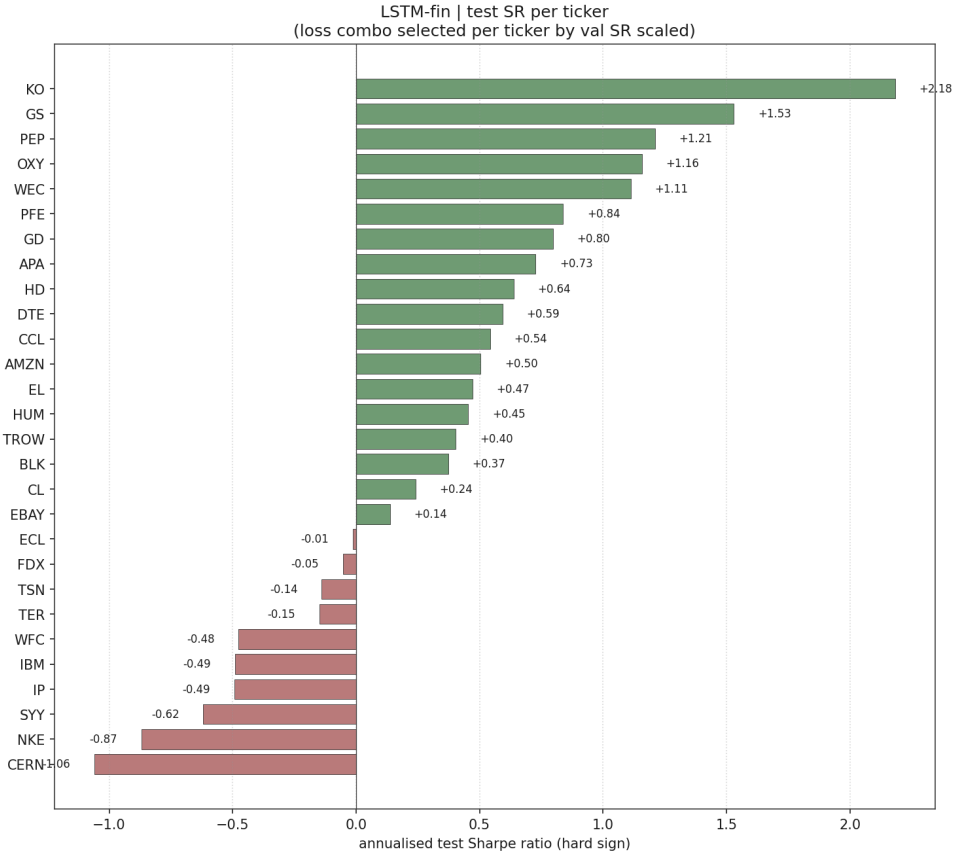


Figure 9.2: Per ticker hard signal test Sharpe for LSTM-Fin under the validation best loss combination. The defensive leads discretionary trails pattern from the linear baselines is preserved.

Loss combination structure. The full hard signal Sharpe surface across all (ticker, loss combination) pairs is shown in Figure 9.3. Some tickers are robust across loss combination choice: KO scores between +1.94 and +2.36 across all six combinations, and WEC, OXY, PFE, and HD show similarly tight ranges. Others swing widely depending on the loss: CCL ranges from −1.14 (PnL+SR LSTM) to +1.42 (STD LSTM), and CL ranges from −0.64 (MSE LSTM) to +0.24 (PnL+SR LSTM). The validation best selection rule can either reward or penalize these unstable tickers, depending on whether the validation and test windows happen to favor the same loss combination per seed.

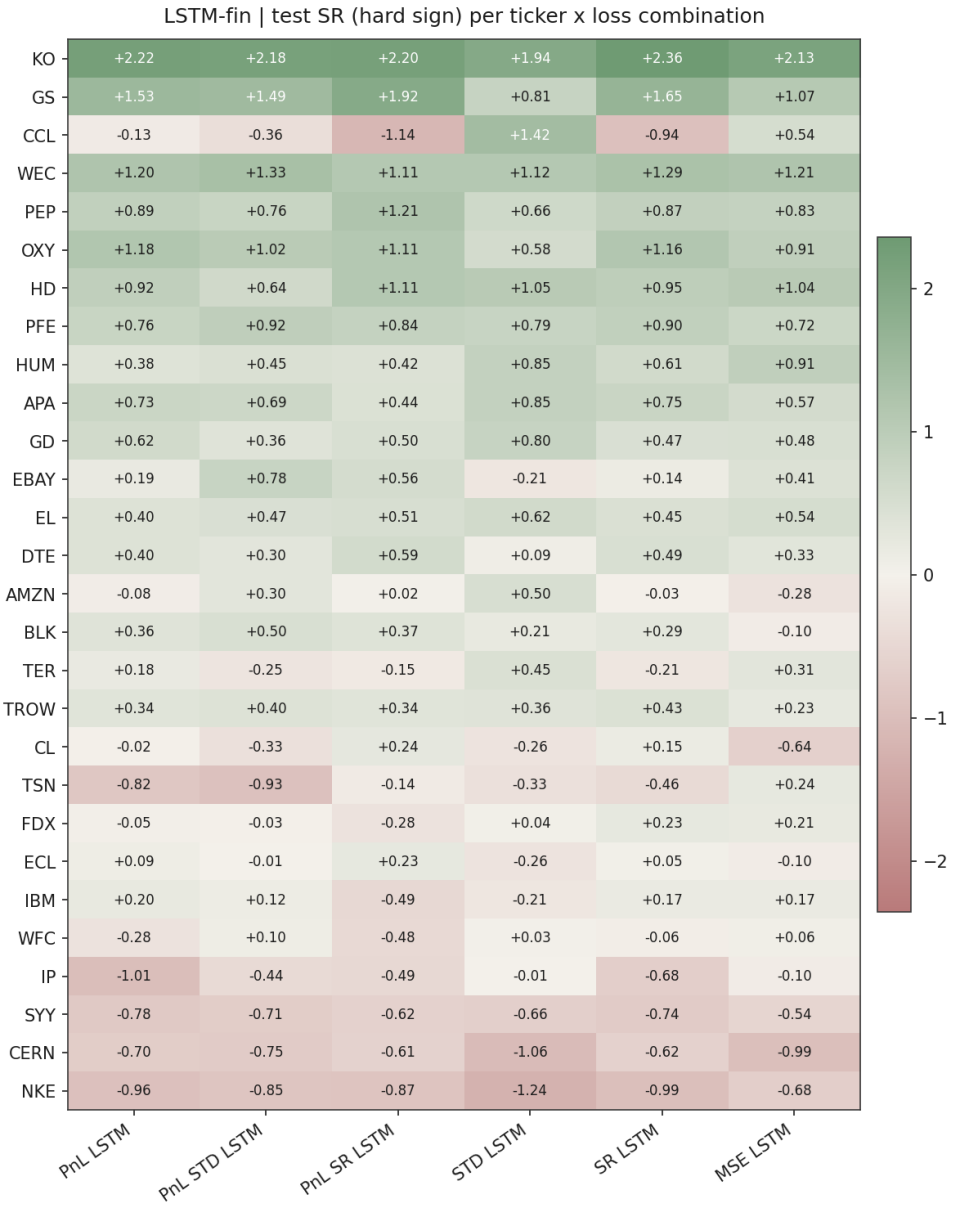


Figure 9.3: Per ticker loss combinations hard signal test Sharpe for LSTM-Fin. Some tickers (KO, WEC) are robust across loss combination choice; others (CCL, CL) swing widely.

Selection. The validation best winner is PnL+SR LSTM on 12 of 28 tickers and PnL+STD LSTM on 6 of 28 tickers. The remaining four combinations are each chosen for one to three tickers. The SR-anchored family dominates selection here as it does in FinGAN's validation best ranking, suggesting that Sharpe anchored objectives are well aligned with the validation Sharpe metric across both architectures.

Diagnostics. Hit rates cluster around 0.50. KO clears the 0.53 threshold at 0.535; OXY at 0.523 is the next strongest but falls below threshold. Forecast target Pearson correlations satisfy $|\rho| < 0.11$ across the universe. Single draw shuffled Sharpes are near zero for all tickers except BLK at +0.534, where directional asymmetry in the test window's return distribution produces a non zero baseline. The diagnostic profile (low correlation, positive Sharpe on a subset of tickers) matches the shape of ARIMA's. This is the noise floor regime where trading rule metrics carry information and variance explained metrics do not.

LSTM-fin | per-ticker best loss combination (selected by val SR scaled)
headline metric: SR hard (sec. 4.2.4 -- point-forecast rule)

ticker	best combo	SR hard	SR shuf	const SR	hit	hit shuf	RMSE	MAE	Corr
KO	PnL STD LSTM	+2.183	-0.013	-0.389	0.5346	0.5012	0.00714	0.00439	+0.102
GS	PnL LSTM	+1.530	+0.096	+0.676	0.5027	0.5047	0.00785	0.00519	+0.037
PEP	PnL SR LSTM	+1.210	+0.050	+0.037	0.5073	0.5031	0.00577	0.00386	+0.068
OXY	SR LSTM	+1.158	-0.057	+0.037	0.5228	0.4990	0.02082	0.01307	+0.075
WEC	PnL SR LSTM	+1.113	-0.014	-0.104	0.5173	0.4986	0.00738	0.00431	+0.071
PFE	PnL SR LSTM	+0.838	-0.039	+0.286	0.5055	0.4989	0.01121	0.00694	+0.058
GD	STD LSTM	+0.797	-0.086	-0.308	0.5209	0.4982	0.00742	0.00507	+0.049
APA	PnL LSTM	+0.727	-0.004	+0.613	0.5273	0.4978	0.02678	0.01644	+0.001
HD	PnL STD LSTM	+0.638	+0.066	+0.166	0.5128	0.5014	0.00883	0.00602	+0.057
DTE	PnL SR LSTM	+0.594	-0.033	+0.049	0.5091	0.5003	0.00972	0.00509	+0.042
CCL	MSE LSTM	+0.543	+0.041	-0.823	0.5173	0.5015	0.03116	0.02086	+0.035
AMZN	STD LSTM	+0.504	+0.149	+0.165	0.5009	0.4981	0.01062	0.00702	+0.024
EL	PnL STD LSTM	+0.472	-0.016	+0.727	0.4964	0.5012	0.01201	0.00768	-0.042
HUM	PnL STD LSTM	+0.455	+0.093	+0.097	0.5082	0.5020	0.01155	0.00695	+0.053
TROW	PnL STD LSTM	+0.404	+0.008	+0.433	0.4891	0.5018	0.01093	0.00711	+0.024
BLK	PnL SR LSTM	+0.374	+0.534	+0.692	0.5073	0.5162	0.01001	0.00659	+0.038
CL	PnL SR LSTM	+0.241	-0.127	-0.045	0.5109	0.4988	0.00677	0.00435	+0.013
EBAY	SR LSTM	+0.137	-0.086	+0.149	0.4936	0.4986	0.01490	0.00951	+0.038
ECL	PnL STD LSTM	-0.014	-0.136	-0.742	0.4982	0.4987	0.00957	0.00587	-0.001
FDX	PnL LSTM	-0.053	+0.023	+0.283	0.4863	0.5019	0.01333	0.00819	+0.028
TSN	PnL SR LSTM	-0.139	-0.119	-0.350	0.5118	0.5004	0.01388	0.00816	-0.052
TER	PnL SR LSTM	-0.149	+0.137	+0.279	0.5137	0.5019	0.01427	0.00978	+0.013
WFC	PnL SR LSTM	-0.476	-0.134	-0.745	0.5018	0.4966	0.01090	0.00701	+0.006
IBM	PnL SR LSTM	-0.491	-0.026	-1.034	0.4718	0.4978	0.01213	0.00831	+0.010
IP	PnL SR LSTM	-0.493	+0.131	-0.598	0.4991	0.5000	0.01093	0.00716	-0.021
SYY	PnL SR LSTM	-0.619	-0.042	-0.275	0.4882	0.4990	0.01909	0.01102	-0.034
NKE	PnL SR LSTM	-0.869	+0.093	+0.163	0.4909	0.5006	0.01110	0.00694	-0.059
CERN	STD LSTM	-1.060	-0.086	-0.234	0.4882	0.4948	0.00984	0.00597	-0.020

Figure 9.4: P
er ticker LSTM-Fin scoreboard: best loss combination by validation Sharpe, hard signal test Sharpe, shuffled and constant long benchmarks, hit rates, RMSE, MAE, and forecast target correlation. Tickers are sorted in descending order of hard signal test Sharpe.

9.5 ARIMA Results

The ARIMA two parity baseline of Section 5.8.3 is fit on 28 individual tickers, each forecast on ETF excess returns. The test window from 2019-10-30 to 2021-12-31 spans to 1,098 half sessions and contains both the March 2020 drawdown and the subsequent recovery. The portfolio aggregate is the Sharpe ratio of the daily PnL summed across all 28 tickers, per Section 8.3.

Headline. The portfolio test Sharpe is +0.642, against a constant long benchmark of −0.195 and a single draw shuffled benchmark of +0.073, giving a net edge of +0.837 over drift. ARIMA produces positive Sharpe on 19 of the 28 stocks (Figures 9.2 and 9.1). The constant long benchmark is negative because the test window opens at end of October 2019 and includes the March 2020 drawdown, where simply holding the universe long would have produced a negative Sharpe even after the recovery period. The ARIMA Sharpe of +0.642 is therefore not a comparison against buy and hold; it is the directional skill the model adds on top of a market that was on average losing money over the test window.

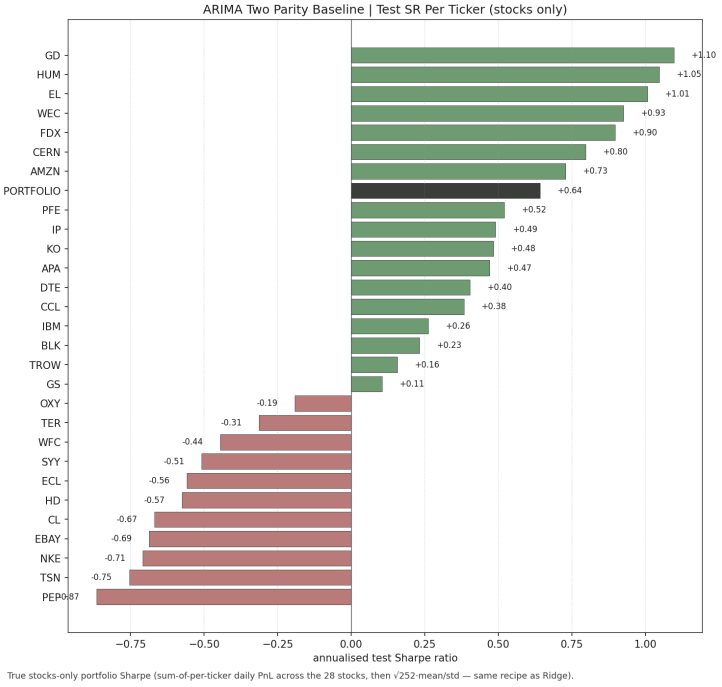


Figure 9.1: Per ticker test Sharpe ratios for ARIMA, sorted in descending order. Stocks shown in green, sector ETFs in blue, and the equal weighted portfolio aggregate in dark gray.

Where the signal is and is not. The top of the per ticker ranking is occupied by defensive and industrial names (PEP, KO, CL, GD, WEC), the bottom by consumer facing and discretionary names (CCL, AMZN, NKE, EBAY).

ARIMA Two Parity Baseline | Test SR / Const SR / Shuf SR / RMSE / MAE
stocks only; PORTFOLIO row = true portfolio Sharpe (sum-of-per-ticker PnL)

ticker	(p,q) p0 / p1	test_SR	const_SR	shuf_SR	RMSE	MAE
GD	(1,0) / (1,1)	+1.096	-0.308	-0.562	0.00739	0.00501
HUM	(1,0) / (2,2)	+1.047	+0.097	-0.870	0.01157	0.00693
EL	(2,2) / (1,0)	+1.007	+0.727	-0.500	0.01186	0.00748
WEC	(1,0) / (2,1)	+0.926	-0.104	+1.135	0.00732	0.00418
FDX	(2,0) / (1,0)	+0.897	+0.283	+0.709	0.01315	0.00781
CERN	(1,0) / (2,2)	+0.796	-0.234	+0.918	0.00981	0.00594
AMZN	(1,0) / (2,2)	+0.729	+0.165	+0.399	0.01060	0.00699
PORTFOLIO	—	**+0.642**	**-0.195**	**+0.073**	—	—
PFE	(2,2) / (1,0)	+0.521	+0.286	-1.041	0.01108	0.00671
IP	(2,2) / (2,1)	+0.491	-0.598	-1.058	0.01062	0.00685
KO	(1,0) / (2,0)	+0.483	-0.389	-0.166	0.00717	0.00438
APA	(1,0) / (2,0)	+0.470	+0.613	+0.293	0.02644	0.01613
DTE	(2,0) / (2,2)	+0.403	+0.049	+0.012	0.00976	0.00496
CCL	(1,1) / (1,1)	+0.383	-0.823	+0.019	0.03116	0.02070
IBM	(1,0) / (2,2)	+0.261	-1.034	+0.660	0.01189	0.00791
BLK	(2,0) / (2,0)	+0.231	+0.692	-0.140	0.00984	0.00625
TROW	(2,0) / (1,0)	+0.156	+0.433	-0.313	0.01085	0.00698
GS	(2,0) / (2,0)	+0.105	+0.676	-0.578	0.00778	0.00505
OXY	(1,0) / (1,0)	-0.192	+0.037	+0.661	0.02075	0.01297
TER	(1,0) / (2,2)	-0.313	+0.279	+1.329	0.01408	0.00956
WFC	(1,0) / (1,0)	-0.445	-0.745	-0.088	0.01065	0.00671
SYY	(2,2) / (2,2)	-0.509	-0.275	-0.650	0.01888	0.01062
ECL	(1,0) / (2,0)	-0.559	-0.742	+0.661	0.00949	0.00576
HD	(1,0) / (1,1)	-0.575	+0.166	-0.825	0.00883	0.00596
CL	(2,0) / (2,1)	-0.669	-0.045	+0.458	0.00663	0.00420
EBAY	(2,2) / (2,2)	-0.687	+0.149	-1.111	0.01493	0.00932
NKE	(2,0) / (1,0)	-0.709	+0.163	+0.214	0.01075	0.00665
TSN	(1,1) / (2,1)	-0.755	-0.350	+0.655	0.01358	0.00785
PEP	(1,1) / (2,1)	-0.865	+0.037	-0.665	0.00574	0.00380

Figure 9.2: Per ticker test Sharpe ratios for the ARIMA two-parity baseline.

The split is informative because it tells us where short term autoregressive structure persists in half day excess returns. It persists in stocks whose business is relatively insensitive to news flow: defensive sectors deliver predictable cash flows that do not swing with the economic cycle, and their half session price moves are therefore dominated by microstructure mean reversion rather than information shocks. It does not persist in stocks dominated by event driven price

moves: discretionary names react to earnings, sales reports, and consumer sentiment data, and the resulting half session moves are jumps rather than autoregressive flows. ARIMA is exploiting structure that exists in the data rather than imposing structure that does not, and the per ticker ranking is largely a map of where that structure exists rather than a property of the model itself. This defensive leads discretionary trails pattern recurs across every model.

Drift contamination. On five tickers (WEC, CERN, TER, OXY, IBM), the shuffled signal Sharpe exceeds the model Sharpe. On these tickers, the test window return distribution is asymmetric enough that random sign assignment outperforms the model's directional forecast on average, so the apparent positive result cannot be separated from drift. The shuffled Sharpe diagnostic is therefore a sanity check rather than a strength. A model whose Sharpe sits below its shuffled benchmark on a given stock is producing nothing that a coin flip would not also produce. See Figure 9.3 in Section **??** for a visualization of this filtering effect.

Magnitude bound. Three stocks exceed the magnitude accuracy bound of RMSE $\leq 2 \times 10^{-2}$ from Section 8.4.2: CCL, APA, and OXY. All three had exceptional volatility in the early test window. CCL is a cruise operator that lost most of its market capitalization during the March 2020 shutdown of the global travel industry; APA and OXY are oil and gas companies whose share prices swung dramatically during the 2020 oil price collapse and subsequent recovery. The half session return distributions for these stocks contain moves an order of magnitude larger than the linear AR/MA structure can represent in magnitude, and the model's forecasts fail the bound regardless of whether the directional sign is correct. Failing the magnitude bound does not invalidate the model on these stocks; it indicates that the linear baseline is structurally unable to extract magnitude information from a return distribution dominated by event driven jumps. The distributional models in later sections address this with output distributions that can adapt their scale to the observed volatility regime.

The PEP parity trap correction. ARIMA without two parity routing produces a spurious Sharpe on PEP because the model captures overnight drift rather than half day directional structure. With routing applied, PEP's Sharpe is -0.87 and sits at the bottom of the ranking. PEP also has the lowest RMSE in the universe (0.0057): the model fits magnitude well and direction badly, exactly the failure mode the parity trap correction is meant to expose. This is a useful negative finding: the framework's evaluation rule correctly penalizes a model that succeeds on the wrong question.

9.6 Ridge AR(10) Results

The Ridge AR(10) two parity baseline of Section 5.8.2 is fit on each of the 28 stocks, with λ selected by five fold cross validation on the training fold. The portfolio aggregate is the equal weighted average of per ticker daily PnL, evaluated on the same test window as ARIMA.

Headline. The portfolio test Sharpe is $+0.227$, against a constant long benchmark of -0.195 and a single draw shuffled benchmark of -0.067, giving a net edge of $+0.422$.

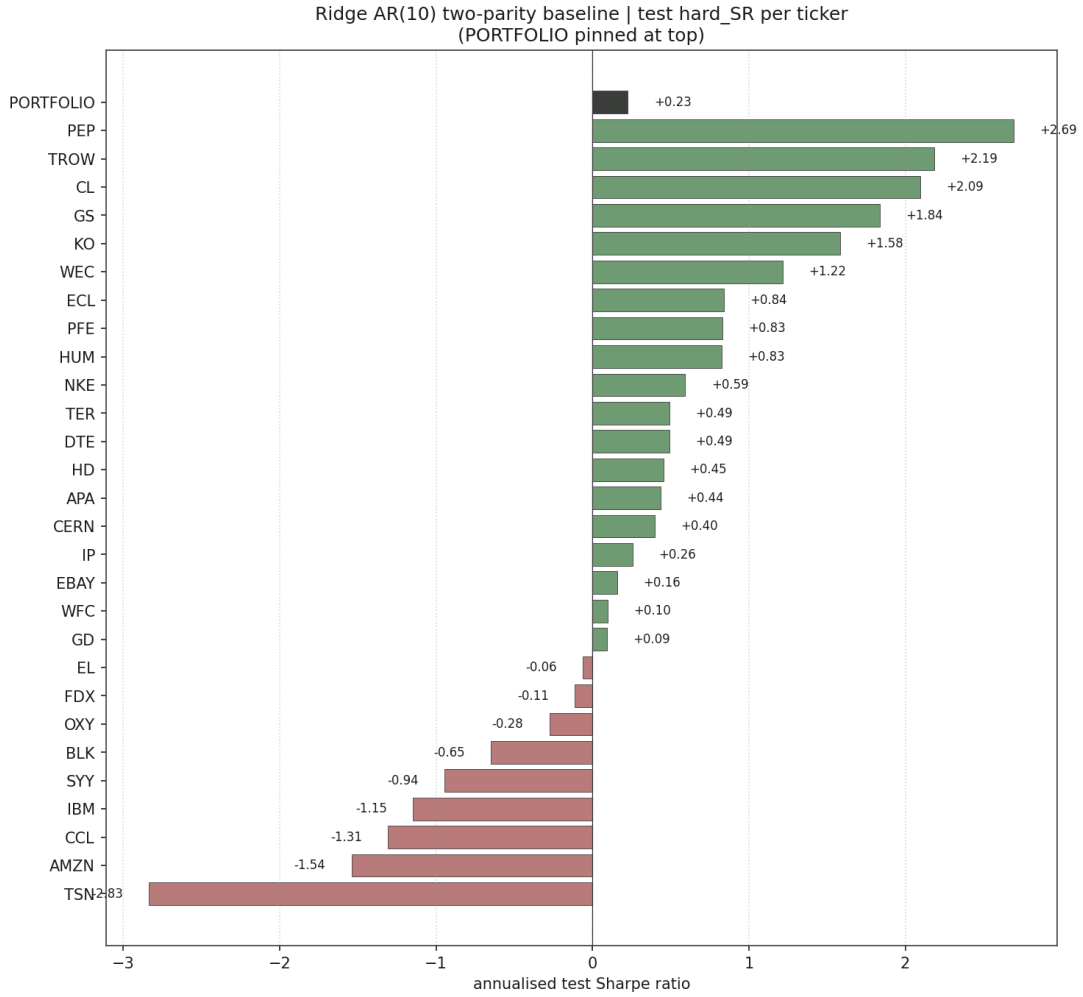


Figure 9.1: Ridge Results

As shown in the Figure9.1, Ridge produces positive Sharpe on 19 of the 28 stocks, the same count as ARIMA. The portfolio level Sharpe is, however, lower than ARIMA's $+0.642$, despite

Ridge producing wider top of ranking Sharpes. The pattern is informative: Ridge's signal is more concentrated on a few strong tickers and accompanied by sharper losses on the bottom of the ranking, while ARIMA's per ticker Sharpes are more uniform. Summing 28 modest but uniformly positive signals delivers more portfolio Sharpe than summing a few strong signals that are partly cancelled by large losers, because portfolio Sharpe rewards consistency of contribution rather than the magnitude of any individual signal.

Hit rate. Six stocks meet the hit rate threshold of 0.53 from Section 8.4.2: PEP, KO, TROW, CL, APA, and SYY. SYY is the noteworthy entry. Its hit rate of 0.529 is just above the threshold but its Sharpe is negative. This combination occurs when the model's correct directional calls land on small magnitude moves while its incorrect calls land on large magnitude moves. The model is right about direction often enough to clear the threshold but wrong on the moves that matter, and because it produces a single point forecast rather than a confidence weighted forecast, there is no mechanism by which it can scale down on uncertain large move predictions. This failure mode is one motivation for the distributional output models introduced in later sections, which can scale position size by forecast magnitude.

Cross sectional pattern. The top of the ranking is dominated by defensive and large cap names that also lead under ARIMA: PEP, TROW, CL, GS, KO, WEC. The bottom is dominated by consumer discretionary and industrial cycle names: TSN, AMZN, CCL, IBM, SYY. The same defensive leads discretionary trails pattern recurs from ARIMA, but with a different magnitude profile. PEP at +2.69 is the strongest single stock result in the universe under any linear model, and TSN at −2.83 is the deepest single stock loss. The characteristic feature of Ridge relative to ARIMA is therefore a wider spread of per stock outcomes rather than a uniformly higher level, consistent with the headline observation that portfolio Sharpe rewards consistency over magnitude.

Ridge AR(10) two-parity baseline | test SR / const SR / shuf SR / net / hit / RMSE / MAE

ticker	test_SR	const_SR	shuf_SR	net	hit	RMSE	MAE
PORTFOLIO	**+0.227**	**-0.195**	**-0.067**	**+0.422**	—	—	—
PEP	+2.694	+0.037	-0.917	+2.657	0.5410	0.00560	0.00371
TROW	+2.187	+0.433	+0.079	+1.753	0.5373	0.01084	0.00690
CL	+2.094	-0.045	-0.160	+2.139	0.5301	0.00656	0.00413
GS	+1.839	+0.676	-0.593	+1.164	0.5173	0.00775	0.00505
KO	+1.582	-0.389	-0.263	+1.970	0.5419	0.00713	0.00439
WEC	+1.216	-0.104	+0.144	+1.321	0.5164	0.00731	0.00422
ECL	+0.842	-0.742	+0.499	+1.584	0.5046	0.00941	0.00577
PFE	+0.831	+0.286	+0.126	+0.545	0.4991	0.01117	0.00685
HUM	+0.828	+0.097	-0.357	+0.732	0.5246	0.01158	0.00698
NKE	+0.594	+0.163	-1.186	+0.431	0.5237	0.01087	0.00668
TER	+0.493	+0.279	-0.220	+0.214	0.5246	0.01408	0.00952
DTE	+0.493	+0.049	-0.879	+0.444	0.4891	0.00975	0.00499
HD	+0.453	+0.166	+1.066	+0.287	0.4827	0.00888	0.00603
APA	+0.436	+0.613	-0.007	-0.177	0.5301	0.02656	0.01603
CERN	+0.397	-0.234	-0.385	+0.631	0.5191	0.00979	0.00592
IP	+0.257	-0.598	+0.410	+0.855	0.5100	0.01074	0.00688
EBAY	+0.158	+0.149	-0.705	+0.009	0.5046	0.01486	0.00935
WFC	+0.099	-0.745	-0.621	+0.844	0.5173	0.01074	0.00682
GD	+0.093	-0.308	-0.229	+0.401	0.4891	0.00739	0.00505
EL	-0.063	+0.727	-0.185	-0.789	0.4818	0.01186	0.00753
FDX	-0.112	+0.283	+0.775	-0.395	0.4891	0.01356	0.00809
OXY	-0.275	+0.037	+0.780	-0.313	0.5027	0.02132	0.01323
BLK	-0.651	+0.692	+1.117	-1.343	0.4763	0.01005	0.00641
SYY	-0.944	-0.275	-0.785	-0.669	0.5291	0.01968	0.01083
IBM	-1.147	-1.034	-0.164	-0.113	0.4718	0.01215	0.00819
CCL	-1.307	-0.823	+0.199	-0.483	0.4827	0.03204	0.02113
AMZN	-1.540	+0.165	-0.002	-1.705	0.4617	0.01078	0.00717
TSN	-2.834	-0.350	+0.957	-2.484	0.4472	0.01389	0.00796

Figure 9.2: Ridge Results

Magnitude bound. The same three stocks exceed the RMSE bound of 2×10^{-2} from Section 8.4.2: CCL, APA, and OXY. All three had exceptional volatility in the early test window (CCL during the March 2020 travel industry shutdown, APA and OXY during the 2020 oil price collapse and recovery), with half session moves an order of magnitude larger than any linear AR or MA process can represent. The reproducibility of this failure across both linear models, fit independently, confirms that these three stocks are genuinely high magnitude rather than that any particular linear specification is mistuned. Ridge does not solve the magnitude problem on

these stocks; the linear regression class is structurally unable to extract magnitude information from a return distribution dominated by event driven jumps.

Drift contamination. Two stocks have shuffled Sharpes that exceed Ridge's test Sharpe (HD and BLK), reproducing the same diagnostic that flagged five stocks under ARIMA.

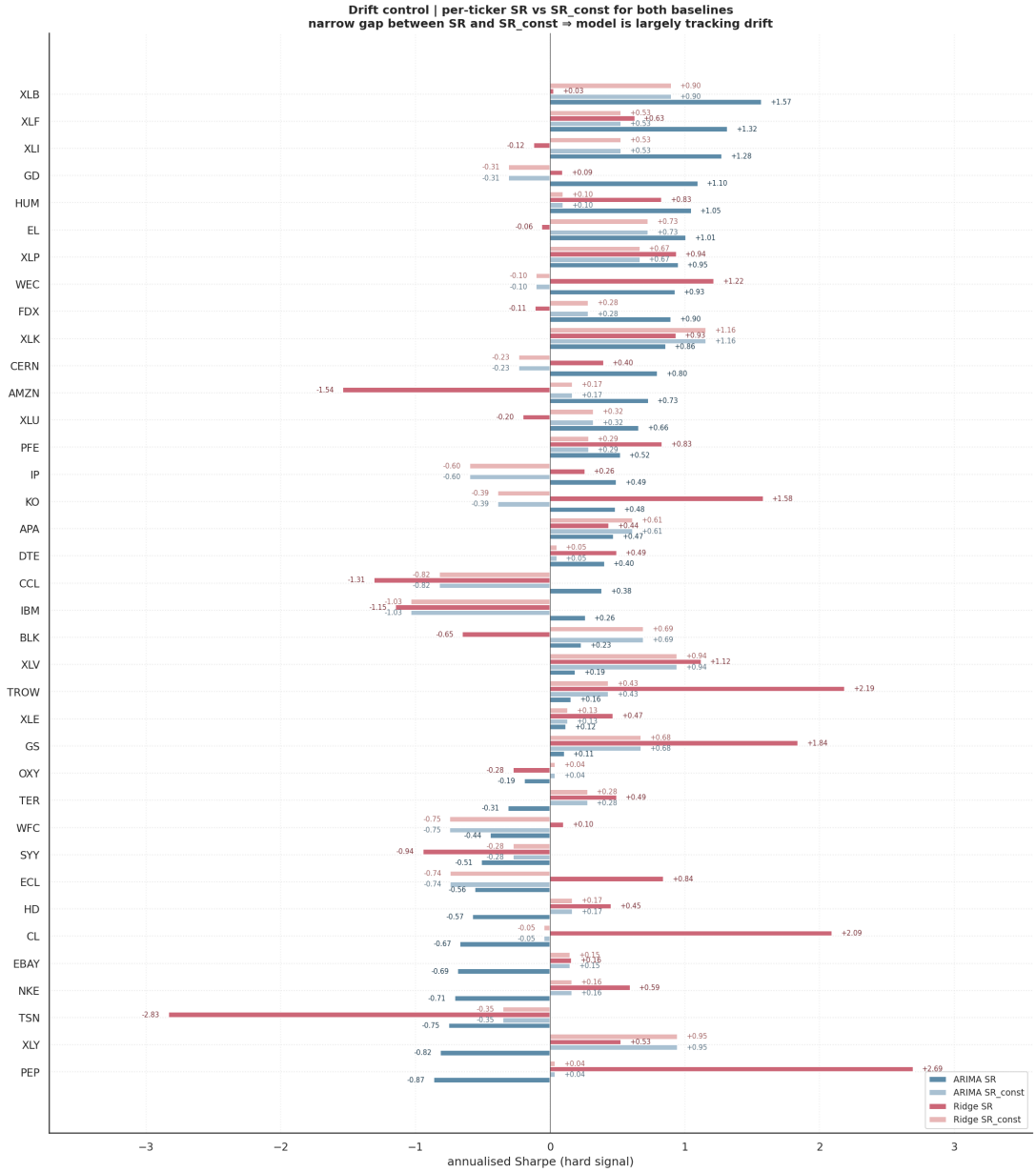


Figure 9.3: Per ticker test Sharpe (dark bars) versus constant long benchmark Sharpe (light bars) for ARIMA and Ridge on the 28 stock universe. A narrow gap between SR and SR_{const} indicates the model is largely tracking drift rather than producing independent signals.

As show in the figure 9.3, the intersection between the two contamination sets is small (no overlap with ARIMA's WEC, CERN, TER, OXY, IBM), suggesting that drift contamination is partly model specific and partly a property of the test window's return asymmetries on individual stocks. Ridge and ARIMA each pick up apparent positive Sharpe on a different set of stocks where the shuffled benchmark also looks good, so the model itself is partly responsible for which stocks end up flagged.

The PEP parity trap correction. With routing applied, PEP's Sharpe of +2.69 is the highest in the universe under any linear model, with a hit rate of 0.541 and an RMSE of 0.0056. Both diagnostics are within their thresholds and consistent with a genuinely directional signal rather than a magnitude only fit. This corrects the spurious overnight drift result that the parity trap correction was designed to remove, and stands as the strongest single stock linear model result in the chapter.

9.7 Cross Model Comparison

This section compares the four models reported in the preceding subsections on the 28 stock universe: ARIMA two parity, Ridge AR(10), LSTM-Fin, and FinGAN. FinTimeGAN compressed is included where it adds substantive content. The comparison is at the portfolio level, where the headline statistical claim of this project is made, and at the per ticker level descriptively. The section closes with a benchmark against the published portfolio Sharpe of Vuletic et al. [33] on her own universe and a FinGAN extension experiment using beta adjusted excess returns as the forecast target.

Portfolio Comparison. The portfolio Sharpes on the 28 stock universe, ordered by strength rather than by model complexity, follow a monotonic progression:

- Ridge AR(10): +0.227.
- ARIMA two-parity: +0.642.
- FinTimeGAN compressed: +1.083, $p = 0.040$.
- LSTM-Fin: +1.370.
- FinGAN: +2.022, $p < 10^{-4}$.

The headline gain from the linear baselines to FinGAN is approximately threefold over ARIMA and ninefold over Ridge. LSTM-Fin and FinTimeGAN compressed sit in the middle of the ranking, with LSTM-Fin slightly ahead. The relative position of LSTM-Fin and FinTimeGAN compressed is the more interesting finding: a flexible point forecast model with an SR anchored loss outperforms a distributional model that loses information at the latent space encoder. Architectural complexity is therefore not strictly rewarded; the way the architecture interacts with the conditioning window matters more than the generative versus point forecast distinction.

Per Ticker Comparison. Three observations emerge from the side by side per ticker Sharpes (Figure 9.1).

First, the cross sectional pattern is preserved across all models. PEP, KO, WEC, CL, PFE, and DTE are top quintile or near top under all four models. CCL, AMZN, IBM, SYY, and TSN are bottom quintile or negative under all four models. The signal source is the same across model families, and the per ticker ranking is largely a property of which stocks have a forecastable structure on this test window. The ranking is not what differs between models; the magnitude of the extraction is.

Second, the per ticker spread widens substantially from ARIMA to FinGAN, while Ridge sits closer to FinGAN despite being a simpler model. FinGAN's per ticker Sharpes range from +2.82 (PEP) to −1.31 (CCL), a spread of 4.13. ARIMA's spread is roughly half this. Ridge spans +2.69 (PEP) to −2.83 (TSN), a spread of 5.52, comparable to or slightly wider than FinGAN's. The pattern indicates that Ridge's per ticker variance comes from a different source than FinGAN's: Ridge's regression structure produces sharp bets on a few tickers and large losses on others, while FinGAN distributes its variance with a wide top end driven by confidence weighted position sizing rather than concentrated linear bets.

Third, on a small subset of tickers the linear baselines outperform FinGAN under the validation best selection rule. APA is the cleanest example: ARIMA produces a Sharpe of approximately +0.47 on it, while FinGAN's validation best selection lands on a loss combination that scores near zero on APA. Under SR-family loss combinations FinGAN does extract signal from APA, but the validation best rule does not pick those combinations on this stock. The implication for model selection is practical: a flexible model is not strictly dominant over a linear baseline, and a universe whose tickers are already well fit by linear methods is one where the marginal value of FinGAN is small.

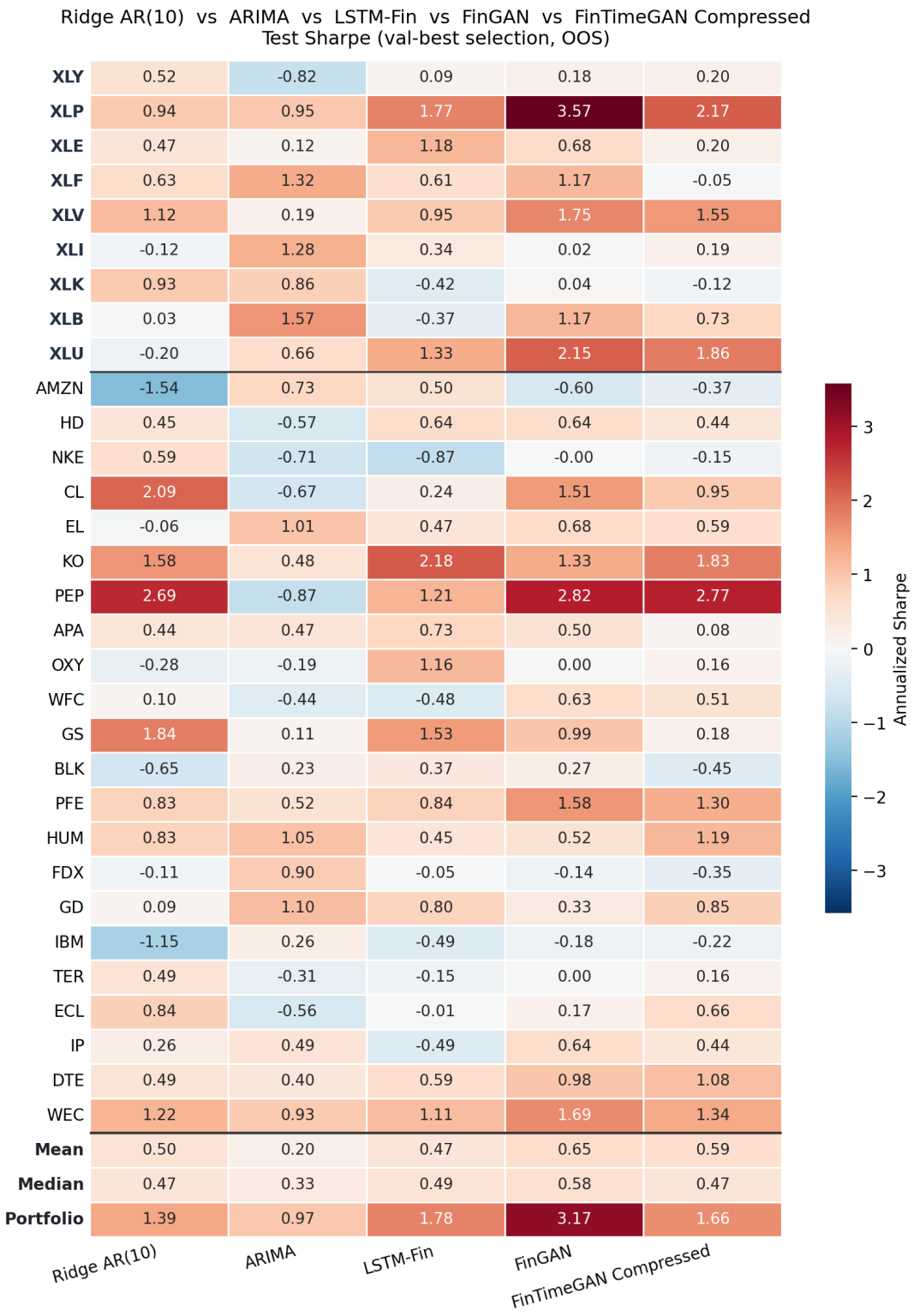


Figure 9.1: Per ticker test Sharpe under each model's validation best loss combination, sorted by FinGAN Sharpe in descending order. The cross sectional pattern is preserved across all models; only the magnitude of extraction differs.

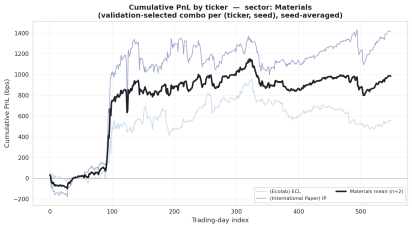
Summary of portfolio results. Two findings carry across the comparison. First, the portfolio level results confirm that the gain from generative modeling on this universe is real and statistically significant. Second, the per ticker comparison clarifies that the gain comes from extracting more signal where signal exists, not from finding signal in tickers where the linear baselines fail. This is a useful negative result: tickers whose returns are not forecastable by linear methods are not made forecastable by adding a generator. Selecting a tradable universe matters at least as much as selecting a model.

9.7.1 The universe of 22 stocks plus 9 ETFs

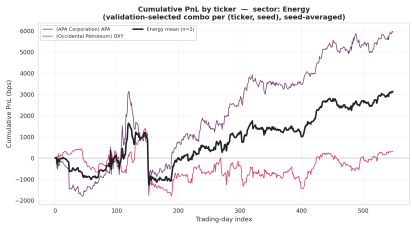
The 28 stock universe used above is an individual ticker universe selected for like for like comparison among this project's models. Vuletic et al. [33] report on a different universe: 22 single name stocks plus 9 sector ETFs, with stocks forecast on ETF excess returns and ETFs additionally forecast on raw returns and contributing their own daily PnL series to the portfolio aggregate. All five models in this project were re run on this universe; the per ticker, per sector, and portfolio results across all five models are shown in Figure 9.3.

Headline. On the Vuletic comparable universe, FinGAN achieves a portfolio Sharpe of , exceeding the best portfolio Sharpe of +1.99 reported by Vuletic et al. [33] on the same universe (her PnL+MSE+SR loss combination, from her Figure 5.12). LSTM-Fin (+1.78) and Fin-TimeGAN compressed (+1.68) sit in the middle; ARIMA (+0.64) and Ridge (+0.23) trail. The hierarchy among this project's models is the same as on the 28 stock universe, but the gaps are wider: FinGAN's lead over the next best model is +1.39 here versus +0.65 on the 28 stock universe. The improvement over Vuletic et al. is attributable to the independent branch loss training protocol of Section 7.3.3, which corrects the chained loss training used in Vuletic's released implementation.

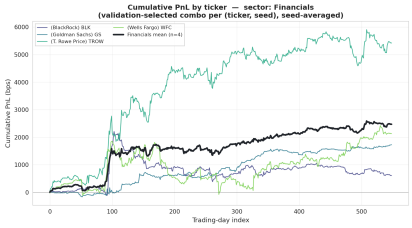
Sector portfolio decomposition. XLP (Consumer Staples) is the strongest sector portfolio at +3.57, XLU (Utilities) at +2.15 is second, and XLV (Health Care) at +1.75 is third. XLB (Materials) and XLF (Financials) tie at +1.17, and the remaining sectors (XLE, XLY, XLK, XLI) are at or near zero. The pattern reproduces the 28 stock universe's defensive leads, discretionary trails structure at the sector level, with the additional finding that sector ETF instruments themselves contribute meaningfully to the portfolio aggregate.



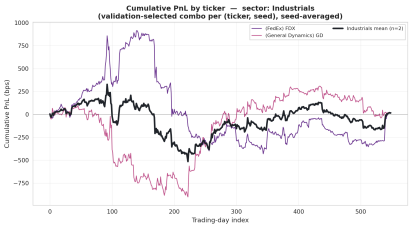
(a) Materials (XLB)



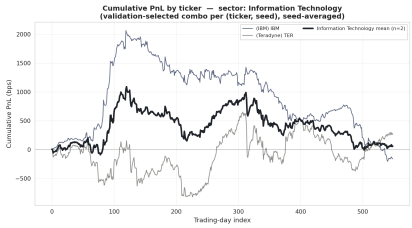
(b) Energy (XLE)



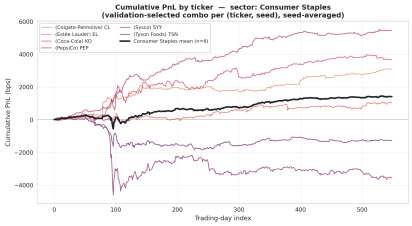
(c) Financials (XLF)



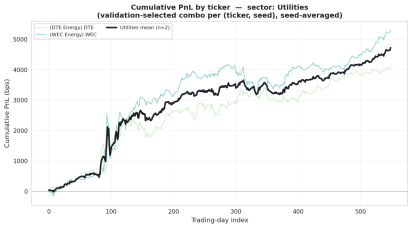
(d) Industrials (XLI)



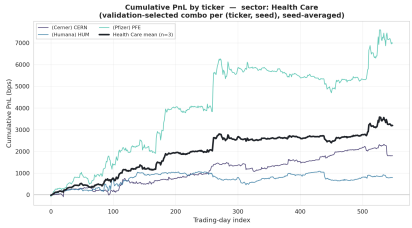
(e) Technology (XLK)



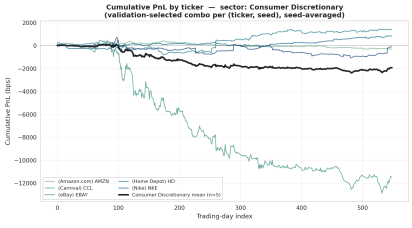
(f) Consumer Staples (XLP)



(g) Utilities (XLU)



(h) Health Care (XLV)



(i) Consumer Discretionary (XLY)

Figure 9.2: Cumulative PnL by ticker within each of the nine GICS sectors of the Vuletic comparable universe, with the sector mean overlay in black.

Why the portfolio Sharpe exceeds the per ticker mean. The mean per ticker Sharpe is +0.65 and the median is +0.58, but the portfolio Sharpe is +3.17. The gap arises from how Sharpe ratios respond to aggregation. When daily PnL series from 31 tickers are summed into one portfolio PnL series, the daily means add up but the daily volatilities do not, because the per ticker PnL series are not perfectly correlated and their fluctuations partly cancel. Sharpe is the ratio of mean to volatility, so a portfolio whose component signals are at least partly independent will have a higher Sharpe than any of its components. The size of the portfolio gain, +2.52 above the per ticker mean, is itself a measure of independence: a portfolio of perfectly correlated signals would show no gain over the average per ticker Sharpe, while one of fully independent signals would show the largest possible gain. The result therefore indicates that the per ticker signals carry genuinely diverse information rather than concentrated exposure to a single sector or factor.

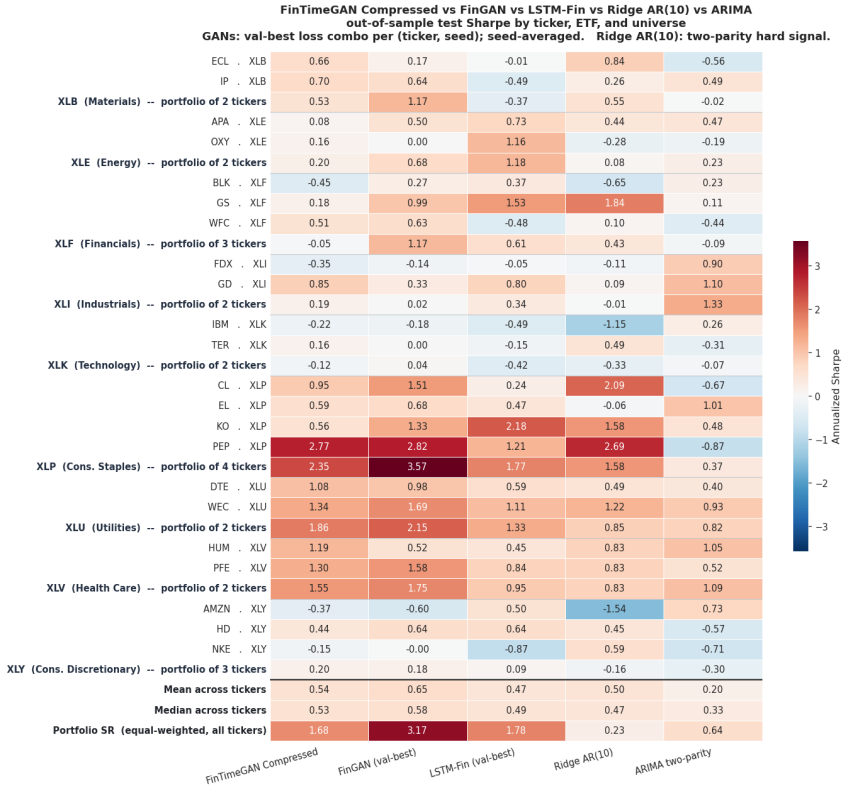


Figure 9.3: Sector rows aggregate per ticker daily PnL within the sector. Bottom row: equal weighted portfolio Sharpe across all tickers. FinGAN leads on the portfolio aggregate; the cross sectional pattern is preserved across all five models.

9.7.2 Beta adjusted excess returns for FinGAN

The relative ETF excess specification used in the headline FinGAN runs treats every stock as having a sector loading of one against its benchmark ETF. This is a strong simplification: stocks have heterogeneous sensitivities to their sectors, and the residual after subtracting the sector return is therefore not a clean idiosyncratic component for stocks whose betas differ meaningfully from one. A natural refinement is to forecast *beta adjusted* excess returns explained in 4.2.3. To test whether this refinement improves trading performance, the FinGAN model was re run on the 28 stock universe with beta adjusted excess returns as the forecast target. Per ticker betas were estimated by ordinary least squares on the training fold and held fixed through validation and test. All other methodological choices (ten loss combinations, validation best selection, five seeds per ticker, independent branch training) match the relative excess return FinGAN runs.

Headline. The validation best portfolio achieves a weighted signal Sharpe of +1.96 on the 549 day test window. The 200 permutation null produces a p value of 0.010, well below the 0.05 threshold. Cumulative portfolio PnL is +1,217.0 basis points. The model passes the portfolio level statistical significance gate of Section 8.5, but with substantially less margin than relative ETF excess return FinGAN, which achieves +2.022 at $p < 10^{-4}$ on the same universe. The beta adjusted version is therefore slightly worse on both axes: 0.06 lower in headline Sharpe, and a much weaker margin against the permutation null. The hypothesis that beta adjustment would improve forecasting on stocks with high beta dispersion is not confirmed empirically. The simpler specification appears preferable on this universe.

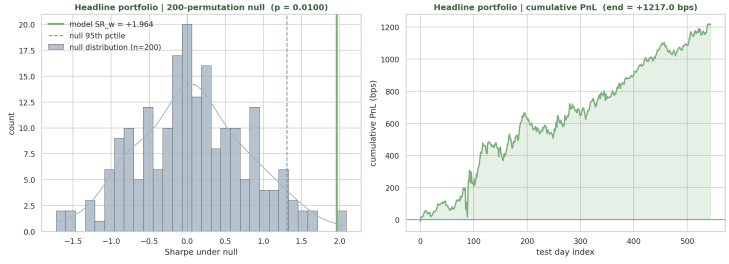


Figure 9.4: Left: 200 permutation null distribution with the model Sharpe marked. Right: cumulative portfolio PnL across the test window. The model rejects H_0^{shuf} at $p = 0.01$.

Two factors plausibly explain the result. First, betas estimated on a 549 day training window are themselves noisy, and the noise in $\hat{\beta}_i$ propagates into the beta adjusted residual that the model is trying to forecast. Second, deviations of true beta from one are small for the defensive and utility tickers that dominate the top of the ranking (PEP, KO, CL, WEC, DTE, PFE all have betas close to one against their sector ETFs), so beta adjustment has little effect on the very tickers that drive the portfolio Sharpe. On stocks with larger beta dispersion (APA, OXY, GS), beta adjustment improved the per ticker forecast, but these contribute relatively little to the portfolio aggregate even in the relative excess return version.

Loss combination structure. The three behavioral clusters identified in the relative excess return FinGAN runs are preserved under beta adjustment: {BCE, MSE, PnL, PnLMSE}, {PnLSR, PnLMSESR, SR, SRMSE}, and {PnLSTD, PnLMSESTD}. Within cluster correlations are slightly different (the STD family pair drops from 0.84 to 0.76, suggesting a small additional source of variability under beta adjustment), but the cross cluster correlations and the overall block structure are identical. This confirms that the loss combination clustering is a property of the loss geometry rather than of the input return representation.

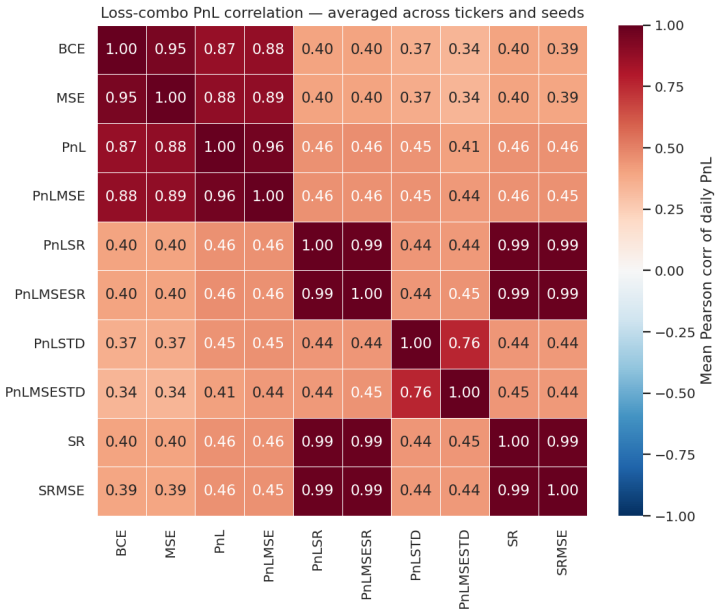


Figure 9.5: Pearson correlation of daily PnL between pairs of loss combinations, averaged across (ticker, seed) runs.

Cross sectional pattern. The per ticker pattern is broadly preserved from the relative excess returns version (Figure 9.6). PEP again leads under the SR family combinations, with the top quintile occupied by the same defensive, utility, and health care tickers as in the relative excess return runs. The bottom of the ranking is also similar, with CCL, TSN, SYY, and IBM as the largest negative contributors. The substantive finding is not that beta adjustment changes which tickers are forecastable, but that it does not substantially change the magnitude of extraction either.

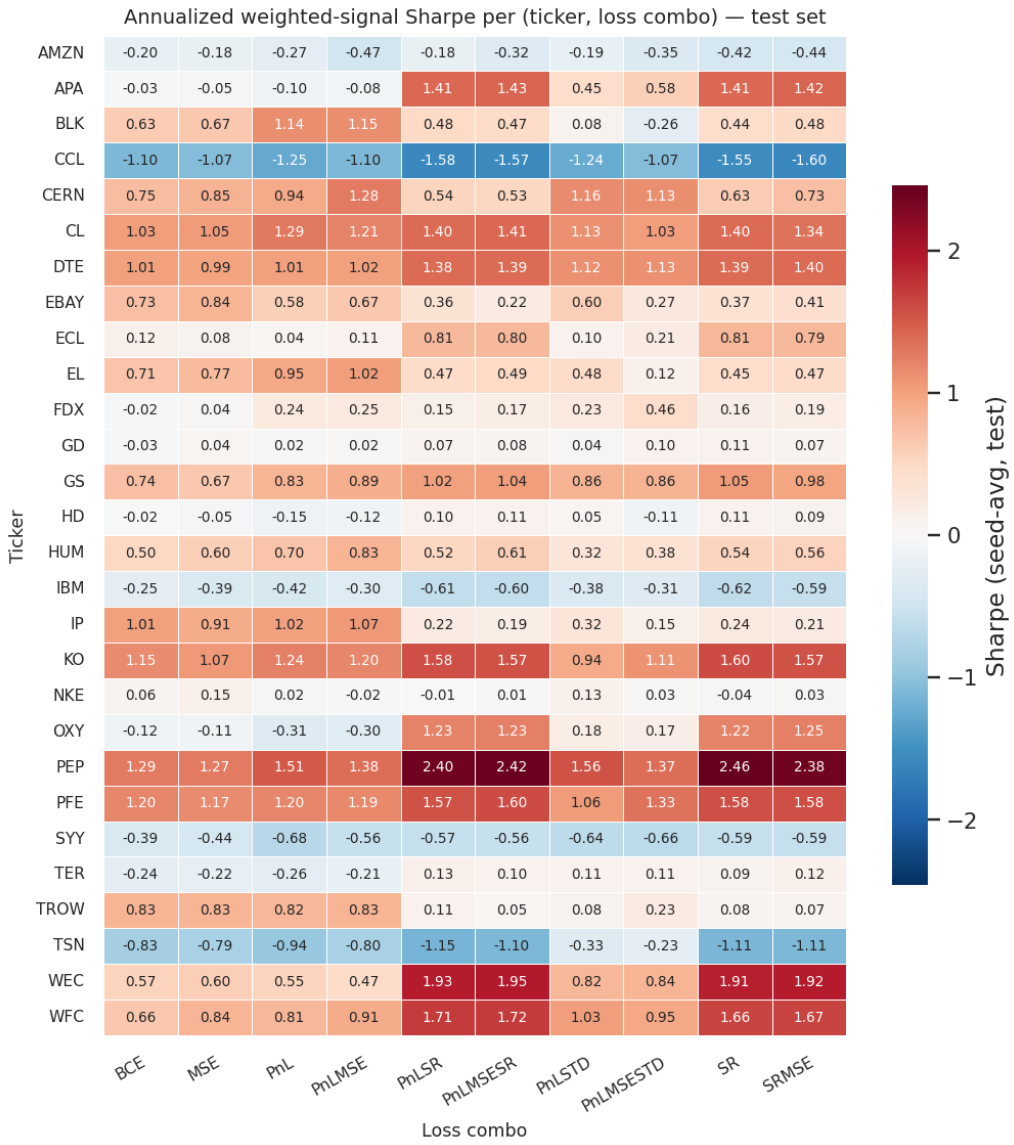
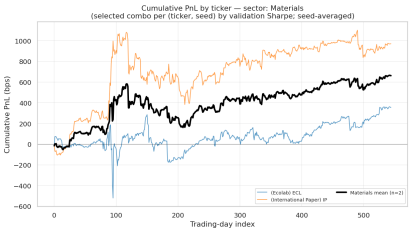


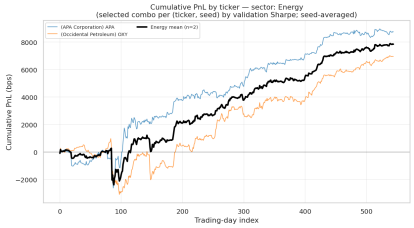
Figure 9.6: Beta adjusted FinGAN: weighted signal test Sharpe per (ticker, loss combination) pair, averaged across 5 seeds on the 28 stock universe.

Sector portfolio decomposition. Under the beta adjusted construction, XLU (Utilities) and XLV (Health Care) lead the sector portfolios at roughly +4,700 and +3,200 bps respectively, followed by XLE (Energy) near +3,100 bps and XLF (Financials) near +2,500 bps. XLP (Consumer Staples) and XLB (Materials) form a middle tier near +1,400 and +1,000 bps. XLK (Technology) and XLI (Industrials) finish near zero, and XLY (Consumer Discretionary) is the only clearly negative sector at roughly −1,900 bps, weighed down by a large drawdown on Carnival Corporation ticker (CCL).

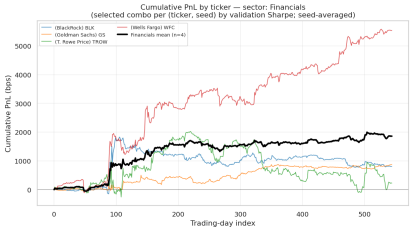
Compared with the relative excess return setup, the beta adjusted sector ranking is broadly similar but no longer led by Consumer Staples: XLP drops from first to mid table, while Energy and Financials move up. Within sector composition also differs in several places. In Financials, T. Rowe Price replaces Wells Fargo as the dominant contributor, and the four constituents end closer together. In Utilities, the lead between DTE Energy and WEC Energy flips. Health Care remains heavily concentrated in Pfizer in both constructions. Information Technology continues to net to roughly zero, but the underlying ticker paths are visibly smoother under beta adjustment, with IBM holding a sustained positive position for most of the test window. The defensive versus discretionary pattern identified in the relative excess return setup is preserved, with Utilities and Health Care leading and Consumer Discretionary trailing.



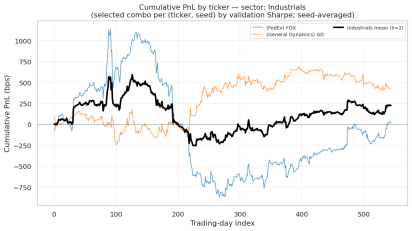
(a) Materials (XLB)



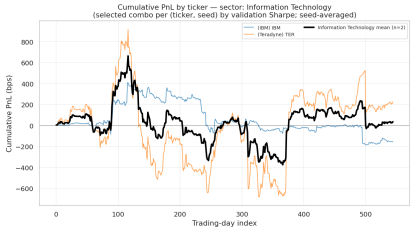
(b) Energy (XLE)



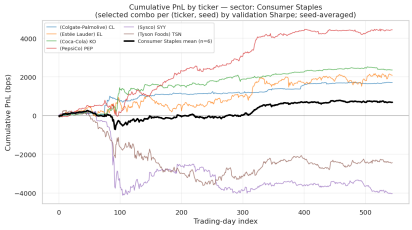
(c) Financials (XLF)



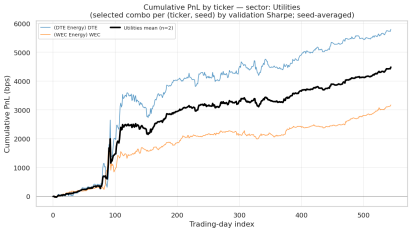
(d) Industrials (XLI)



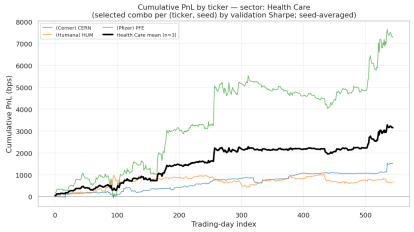
(e) Technology (XLK)



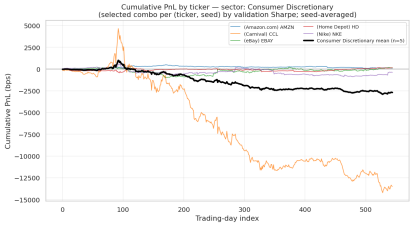
(f) Consumer Staples (XLP)



(g) Utilities (XLU)



(h) Health Care (XLV)



(i) Consumer Discretionary (XLY)

Figure 9.7: Cumulative PnL by ticker within each of the nine GICS sectors, with the sector mean overlay in black.

Conclusion. Beta adjustment was motivated by the observation that the relative ETF excess return specification implicitly fixes every stock's sector loading at one, which is a strong simplification for a universe with heterogeneous betas. The empirical evidence does not support the refinement on this universe. The per ticker ranking, the three loss combination clusters, and the broad sector pattern of defensive sectors leading and discretionary trailing are all preserved, which indicates that the structure of forecastability in this universe is driven by the cross section of tickers rather than by the choice between excess and beta adjusted residuals. The relative excess return specification is therefore retained as the primary FinGAN configuration for other models, with beta adjustment recorded as a refinement that was tested and found not to improve trading performance on this universe.

Chapter 10

Discussion and Future Work

This chapter steps back from the per model results of Chapter 9 to ask three questions: what these results mean as a whole, where the framework's edges are, and what would be productive to investigate next.

10.1 What the Results Mean

The headline finding is that adversarial generative modeling materially improves on linear and recurrent baselines for half session ETF excess return forecasting on this universe. FinGAN's portfolio Sharpe of +2.022 on the 28 stock universe is roughly threefold the ARIMA baseline and ninefold the Ridge baseline. On the Vuletic comparable universe, FinGAN's +3.17 exceeds the +1.99 best portfolio Sharpe reported by Vuletic et al. [33], by roughly 60%. The improvement is statistically significant at the portfolio level under a permutation null, well below the 0.05 threshold this project's evaluation framework uses as its acceptance criterion.

The cross sectional pattern is the more interesting finding. The same tickers lead and trail under all five models. PEP, KO, CL, WEC, and DTE are top quintile under linear and neural models alike; CCL, AMZN, and TSN are bottom quintile across the board. This consistency tells us that the gains from moving up the model hierarchy are about cleaner signal extraction rather than expanded coverage. Where there is autoregressive structure in half session returns, every model finds it; where there isn't, no model invents it. A practical implication for practitioners is that universe selection matters at least as much as model selection, and possibly more.

A third finding sits at a finer grain. Cross ticker hit rates cluster tightly around chance (~ 0.50) across most loss combinations, with only a small subset of cells (CL, PEP, KO, WEC under SR family combinations) clearing the 0.53 threshold. For most of the universe, the distributional models are therefore not improving directional accuracy; they are improving confidence weighted position sizing. A half session forecast of +0.005 produces a different daily PnL contribution than a forecast of +0.0005, even when both are correctly signed, and the weighted signal trading rule captures this where the hard signal rule does not. This is the structural reason FinGAN beats LSTM-Fin: not because FinGAN gets the sign right more often on average, but because its distributional output supports a confidence channel that point forecast models cannot match.

A fourth finding concerns the input return representation. Beta adjusted excess returns ($r_{i,t} - \beta_i \cdot r_{ETF,t}$) were tested as a refinement of the relative excess returns specification. The empirical result did not support the refinement: portfolio Sharpe dropped slightly from +2.022 to +1.964, and the margin against the permutation null weakened from $p < 10^{-4}$ to $p = 0.01$. The most likely explanation is noise in the ordinary least square beta estimates on a 549 day training window, propagating into the adjusted residuals. The relative excess returns specification is therefore retained as the primary FinGAN configuration in this project, and beta adjustment is recorded as a refinement that was tested and found not to improve performance on this universe.

10.2 Limits of the Framework

Single test window. All models are evaluated on a chronological 80/10/10 split, with the test window covering late 2019 through 2021. This window includes the COVID drawdown and the subsequent recovery, but the framework cannot distinguish whether the headline result is a property of generative modeling on financial returns or a property of generative modeling on *this particular* return distribution. A rolling origin backtest, with the model retrained on a 1 year window and evaluated on the immediately following 6 month window, would address this by producing a sequence of test results across multiple regimes.

Five seeds. Each (ticker, loss combination) cell uses 5 random seeds. This is enough to detect order of magnitude differences across cells but too few to produce tight confidence intervals on per cell Sharpes. Per ticker results in Chapter 9 should be read as point estimates; differences within ± 0.3 Sharpe between cells may not survive a more aggressive seed list.

Per ticker training. The models are trained one ticker at a time, matching Vuletic et al. [33]. This is the right choice for a like for like comparison against her published results but does not exploit cross stock structure. A pooled or sector pooled model could in principle borrow strength from related tickers and might beat the per ticker headlines reported in this project. The pooled setting was also constrained by the available hardware: training FinGAN on the full pooled cross sectional panel would require substantially more memory and compute than the per ticker setting, and a thorough investigation would benefit from a larger machine.

Frictionless execution. Reported Sharpes assume execution at the closing or opening price of each half session. Real trading would incur bid ask spread, slippage, market impact, and commission costs. For a universe of liquid large cap stocks turning over once per half session, these costs are unlikely to fully erode the headline number, but they would reduce it. The reported Sharpes are an upper bound on what would be realized by an actual trading desk.

Sharpe is not a complete fitness function. The framework evaluates distributional models by Sharpe, PnL, and a small set of distributional sanity flags. It does not check whether the conditional distributions produced by the generator match the true conditional distributions, so a high Sharpe is consistent with both well calibrated and poorly calibrated distributions. Calibration matters most for downstream applications such as risk management, scenario generation, or stress testing, where the shape of the full distribution drives the answer rather than its trading utility. The framework as currently constituted is therefore suitable for evaluating models for trading but not for evaluating models intended as general purpose financial distribution generators.

A related incompleteness shows up at selection time. The validation best rule picks the loss combination with the highest validation Sharpe per (ticker, seed), which on FinGAN runs frequently selects STD based combinations even though those are the most prone to sign degenerate distributions. The framework's distributional sanity gate at the model level catches the worst cases, but the underlying issue is unchanged: validation Sharpe alone is not enough to identify a healthy generator. A more conservative selection rule would condition on both validation Sharpe and the absence of collapse on the validation set; whether this produces a higher portfolio Sharpe (because fragile combinations are downweighted) or a lower one (because the rule has fewer combinations to choose from) is an empirical question.

Architectural complexity is not strictly rewarded. The model hierarchy reported in Chapter 9 does not produce monotonically better Sharpes as architectural complexity increases. LSTM-Fin (point forecast, +1.370) outperforms FinTimeGAN compressed (distributional, latent space, +1.083) on the same universe, despite the latter being the more architecturally complex of the two. The latent space encoding step appears to discard temporal information that return space conditioning preserves, and the more complex model is unable to recover what the simpler model can already extract. This is a useful negative finding: in time series forecasting, the right inductive bias matters more than the number of parameters or the depth of the architecture, and adding architectural machinery without preserving the relevant information structure can actively hurt performance. The selection of FinGAN over FinTimeGAN compressed as the project's primary distributional model in subsequent chapters reflects this finding.

Interpretability and explainability. The neural models reported in Chapter 9 (LSTM-Fin, FinGAN, FinTimeGAN compressed) are essentially opaque. A FinGAN trained on PEP forecasts a distribution over the next half session's excess return, but the model's internal representation does not expose why a particular forecast distribution was produced for a particular conditioning window. There is no decomposition of the forecast into interpretable factors (re-

cent volatility, recent direction, sector level moves), no attribution of the prediction to specific input features, and no clear mechanism for asking the model what it would do if a single feature changed. This opacity is a feature of distributional generative models generally, not a peculiarity of the architectures used here. For trading applications, the lack of interpretability creates practical problems: risk managers cannot audit decisions they cannot trace, practitioners cannot develop intuition about when to trust the model, and model failures are hard to debug because losses cannot be attributed to specific failure modes. Linear baselines like ARIMA and Ridge are interpretable in ways the neural models are not, and this interpretability is itself a value the simpler models offer that the framework's Sharpe based comparison does not capture.

10.3 Future Work

The following directions emerged during the project but were beyond its scope. Each addresses one of the limits above.

Discriminator in latent space. FinTimeGAN currently runs the adversarial game in return space: the generator's latent output is recovered through the Recovery network before being compared with real returns. A natural variant moves the discriminator into latent space directly, distinguishing real and generated latent vectors instead of real and generated returns, while keeping the financial losses (PnL, SR, MSE) in return space. The motivation is a separation of concerns. The adversarial loss should pressure the generator's distributional shape; the financial losses should pressure the trading utility of the generator's mean. Currently both interact nonlinearly through Recovery, which couples them in ways that make the training signal harder to interpret. Decoupling the two losses, with the side benefit of skipping the Recovery pass at each adversarial step, may produce a smoother training signal. Whether trading performance improves is empirical.

Beta adjustment with refined estimators. The relative excess returns specification was tested against beta adjusted excess returns ($r_{i,t} - \beta_i \cdot r_{ETF,t}$) in Section9.7.2. Beta adjustment

did not improve performance on this universe. The most likely explanation is noise in the OLS(Ordinary least square) beta estimates on a 549 day training window, which propagates into the adjusted residuals. A natural follow up is to test whether shrinkage estimators (e.g., Vasicek shrinkage toward unity), rolling betas, or factor model based decompositions produce a cleaner residual that the model can learn from. The negative result on this universe does not rule out beta adjustment generally; it rules out the specific OLS estimator on this window.

Pooled and sector pooled models. A universal model trained on the pooled cross sectional return panel could exploit cross stock structure that per ticker models cannot. Vuletic et al. [33]'s sector pooling experiment on the XLP universe achieved a portfolio Sharpe of approximately +1.43, comparable to the per ticker results reported here. Two architectures are worth comparing against the per ticker headline: a fully pooled FinGAN trained on the universe wide return panel, and a per sector FinGAN trained separately on each sector's stocks. The latter is closer to Vuletic's setup; the former is a more aggressive pooling that this project did not test.

Distributional accuracy via proper scoring rules. The continuous ranked probability score (CRPS) and the probability integral transform (PIT) directly measure whether the generated conditional distributions match the true conditional distributions. Adding these metrics would let the framework distinguish between high Sharpe models with calibrated distributions (suitable for risk and scenario applications) and high Sharpe models with poorly calibrated distributions (suitable for trading but not for downstream uncertainty quantification). Chapter 8 introduces the framing, but implementing CRPS and PIT and re evaluating all models against them is left as future work.

Out of sample testing on a different universe. The current evaluation is on a fixed universe of 28 single stocks, which is essentially Vuletic et al. [33]'s universe with minor extensions. A meaningful out of sample test would train on this universe and evaluate on a disjoint universe (different stocks, possibly different sectors) without retraining. Whether the model's signal gen-

eralizes across the universe boundary is the question that would tell a practitioner whether to deploy.

Transformer based generator and discriminator. The FinTimeGAN section already discusses one specific transformer style variant: the attention mode that conditions on the full latent sequence rather than the final vector. A more aggressive direction would replace the LSTM cells in both the generator and the discriminator with self attention layers, giving the model access to all positions of the conditioning window during forecasting. This is the architectural direction TimeGAN itself moved in subsequent work. Preliminary work on transformer based generators was started during the project but could not be completed within the time limits, and the direction is worth pursuing in follow up work.

Interpretability of distributional generative models. The interpretability limitation discussed in Section 10.2 is a substantive open problem in its own right. Methods for explaining the forecast distributions produced by FinGAN and FinTimeGAN, such as feature attribution, decomposition of the forecast distribution into mean, variance, and shape components traceable to specific input features, or counterfactual reasoning about what the model would forecast if a single input feature changed, would expose what the models have actually learned. Beyond satisfying the audit and risk management requirements that black box models cannot, interpretability has a substantive research benefit: signals the model has identified internally could be extracted and reused in other contexts (factor models, simpler trading rules, risk decompositions) in ways that the trained network's parameters alone cannot support. The current framework reports what FinGAN can do; an interpretable variant would expose how it does it, and what about its strategy generalizes beyond the trained network itself.

10.4 Conclusion

This project extended the FinGAN framework of Vuletic et al. [33] along four axes: a modular TensorFlow reimplementation, the FinTimeGAN architectural extension, an independent

branch loss training protocol, and an empirical investigation of input return representations. The portfolio level results pass the framework's statistical significance gate by margins ranging from comfortable (FinTimeGAN compressed at $p = 0.04$) to overwhelming (FinGAN at $p < 10^{-4}$).

The methodological contributions matter independently of the headline numbers. The framework's separation of selection (validation Sharpe), statistical significance gating (portfolio level permutation), and distributional sanity gating (model level collapse flags) lets the project report results that pass scrutiny on each axis without conflating them. The validation best rule's tendency to select fragile loss combinations was caught at the model level by the distributional sanity gate, even if it persists at the cell level. The hit rate at chance, Sharpe not at chance pattern identifies confidence weighting rather than directional accuracy as the channel through which distributional models add value over point forecast models.

The trading frequency and universe choices made here are deliberate but specific, and the results should be read in that scope. Whether FinGAN style generative modeling becomes a routine tool in directional forecasting depends on its behavior out of sample, in different regimes, and at trading frequencies other than half session. The results reported here suggest the answer is affirmative under the conditions tested, with calibration, generalization, interpretability, and execution cost questions all open for future work.

Appendix A
Glossary of terms

Financial prices evolve in unpredictable ways, yet they are not entirely random. Statistical and probabilistic methods allow researchers to identify patterns, estimate risks, and forecast returns. Because many different factors influence asset prices, the modeling problem quickly becomes mathematically complex. Tools such as stochastic differential equations, Brownian motion, and partial differential equations form the traditional theoretical foundation of quantitative finance. Data driven methods extend these ideas by learning structure directly from historical observations when explicit models are difficult to specify.

This glossary collects definitions of terms that appear across multiple chapters of this thesis. Each entry has a label so it can be cross referenced from the main text. The glossary is organized into three sections: technical indicators used as model inputs or compared against in the literature, statistical and machine-learning concepts that recur throughout the thesis, and the symbolic operator vocabulary used in the reinforcement learning framework discussed in chapter 2.

A.1 Technical Indicators

Exponential Moving Average (EMA) A moving average that gives more weight to recent prices, so it reacts faster to new information than a simple moving average (SMA). Used to detect trends and dynamic support and resistance levels.

Relative Strength Index (RSI) A momentum oscillator that measures the speed and magnitude of recent price changes. Captures momentum shifts, trend strength, and potential reversals.

Stochastic Oscillator (%K/%D) A two-line indicator that measures where the most recent closing price sits relative to the asset's recent price range. Used to detect momentum shifts and potential turning points.

Williams %R Similar to the Stochastic %K but inverted. Measures how close the price is to its recent highs or lows. More sensitive than RSI, often used for identifying exhaustion points.

A.2 Statistical and Machine Learning Concepts

Cross sectional A property or quantity is *cross sectional* when it is computed across different assets at the same point in time, as opposed to *time series*, which describes computation across time for a single asset. For example, a cross sectional rank of a stock on a given day asks where its value sits among all other stocks that day. A time series rank of the same stock asks where its current value sits in its own recent history. Cross sectional methods are central to alpha investing because what often matters is not whether a stock is up but whether it is up more than its peers.

Multi-head LSTM An LSTM architecture in which a shared LSTM body produces representations that are then passed to multiple output networks ("heads"), each producing a separate prediction. The most common use is multi-horizon forecasting, where each head predicts the target at a different forecast horizon (one step ahead, five steps ahead, ten steps ahead, and so on). Not to be confused with multi-head attention, which is a different mechanism that splits a single attention computation into multiple parallel attention computations over different learned subspaces of the input.

R^2 (coefficient of determination) A measure of how well a model's predictions explain the variance in the target variable. It is defined as

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}},$$

where $\mathrm{SS}_{\mathrm{res}} = \sum_t (y_t - \hat{y}_t)^2$ is the residual sum of squares and $\mathrm{SS}_{\mathrm{tot}} = \sum_t (y_t - \bar{y})^2$ is the total sum of squares. R^2 ranges from 0 (the model explains none of the target's variance) to 1 (perfect prediction). For financial returns, even useful trading models typically produce very low R^2 values because most of the variance in returns is unforecastable noise. This makes R^2 a misleading measure of trading utility: a model with $R^2 = 0.02$ may still produce profitable directional forecasts, while a model with higher R^2 achieved through magnitude calibration on noisy targets may not.

Graph Neural Network (GNN) A neural network designed to operate on data structured as a graph, that is, a collection of nodes (entities) connected by edges (relationships). Each node has an associated feature vector, and each edge optionally carries its own features. The core operation in a GNN is *message passing*: each node updates its own representation by aggregating information from its neighbors, typically through a learned function applied to the features of the connected nodes and edges. Stacking multiple message passing layers lets information propagate further across the graph, so a node's final representation can encode patterns from nodes several hops away. GNNs are a natural fit for problems where the data has explicit relational structure that fully connected or convolutional networks cannot easily express. In finance, GNNs are used to capture cross sectional dependencies between assets, where the nodes are stocks and the edges encode learned or pre-defined relationships such as sector membership, supply chain links, or correlation in returns. Unlike a CNN, which exploits the regular grid structure of images, or an LSTM, which exploits the sequential structure of time, a GNN exploits arbitrary graph structure, making it suitable for problems where the connections between data points are themselves part of the modeling problem

F1 score A classification metric that combines precision (the fraction of predicted positives that are actually positive) and recall (the fraction of actual positives that the model correctly identifies) into a single number. It is defined as the harmonic mean,

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

ranging from 0 (worst) to 1 (perfect classification). The harmonic mean is used because it penalizes models that are strong on one measure but weak on the other: a model with 90% precision and 10% recall scores $F_1 \approx 0.18$, not 0.50. For multiclass tasks, the F1 score is computed per class and then averaged across classes, either with equal weight (macro-F1) or weighted by class frequency (weighted-F1).

Hyperbolic tangent (tanh**)** [16][1] A nonlinear activation defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

mapping the real line to $(-1, +1)$. The function is smooth, odd, and saturates as $|x| \to \infty$. It is used in standard LSTM cells (the cell-state and output transformations both pass through tanh) and as the output activation of the FinGAN generator, where the bounded range matches the bounded scale of normalized excess returns. Saturation matters in the trading utility loss terms (Section 8.4.2): once the input to tanh is large enough that the output saturates near ± 1, the gradient vanishes and additional magnitude in the input no longer affects the loss. This is the mechanism that makes magnitude diagnostics necessary alongside directional metrics for the PnL* and SR* loss terms.

Sigmoid (σ) [16][2] A nonlinear activation defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

mapping the real line to $(0, 1)$. The function is smooth, monotonic, and saturates as $|x| \to \infty$. It is used in the gating mechanisms of LSTM cells (the forget gate, input gate,

[1] PP. 89-93
[2] PP. 89-93

and output gate each apply sigmoid to their pre activations) and as the output activation of binary classifiers, where the output is interpreted as a probability. The discriminator network in FinGAN applies sigmoid to its final layer to produce a real versus generated probability that the binary cross entropy loss is then computed against. Like tanh, sigmoid saturates for large $|x|$, which can cause vanishing gradients in deep networks.

Rectified Linear Unit (ReLU) and Leaky ReLU [16][3] ReLU is the piecewise-linear activation defined as

$$\mathrm{ReLU}(x) = \max(0, x),$$

mapping negative inputs to zero and leaving positive inputs unchanged. ReLU has become the default activation in deep network hidden layers because its gradient is constant on the positive side, which avoids the vanishing-gradient problem that affects tanh and sigmoid. The drawback is that neurons whose pre-activations are persistently negative produce no gradient and stop learning (the "dying ReLU" problem). *Leaky ReLU* addresses this by allowing a small negative slope α for negative inputs:

$$\mathrm{LeakyReLU}_\alpha(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases},$$

typically with $\alpha = 0.01$ or $\alpha = 0.2$. Leaky ReLU is the conventional choice for the discriminator's hidden layers in GAN architectures, since it preserves nonzero gradients across the full input range and stabilizes adversarial training.

Softmax [16][4] A nonlinear activation that maps a vector of K real-valued scores to a probability distribution over K classes:

$$\mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}},$$

for $i = 1, \ldots, K$. The output components are non-negative and sum to one, so softmax is the standard output activation for multi-class classifiers, where each output represents the predicted probability of the corresponding class. The relative magnitudes of the input

[3] PP. 89-93
[4] PP. 89-93

scores determine the relative probabilities of the output, and a temperature parameter τ can be introduced as softmax(x/τ) to control the sharpness of the resulting distribution. In trading contexts, softmax also appears in attention mechanisms, where it normalizes a vector of attention scores into weights that sum to one across the attended positions.

A.3 Finance and Risk Concepts

Value at Risk (VaR) A quantile of the loss distribution at a chosen confidence level. The α-level VaR is the loss that is exceeded with probability α over a given horizon. For example, a one day 99% VaR of \$1 million means that on any given day there is a 1% chance of losing more than \$1 million. VaR is widely used in regulatory and risk management settings because it summarizes tail risk in a single number. Its main limitation is that it captures only the threshold of tail losses, not their magnitude beyond that threshold.

Expected Shortfall (ES) Also known as conditional VaR or CVaR. The average loss conditional on exceeding the VaR threshold at a given level α:

$$\mathrm{ES}_\alpha = \mathbb{E}[L \mid L > \mathrm{VaR}_\alpha].$$

Unlike VaR, expected shortfall is sensitive to the magnitude of tail losses rather than just their frequency, and it is a coherent risk measure in the sense of Artzner et al. (1999). Both VaR and expected shortfall are standard metrics in risk management and regulatory settings, but expected shortfall is generally preferred for its sensitivity to extreme losses and its better mathematical properties.

A.4 Symbolic Operators in Formulaic Alpha Mining

The reinforcement learning framework discussed in chapter 2 uses a vocabulary of operators as building blocks for symbolic factor expressions. Each operator transforms one or more input series (such as daily close prices, volume, or returns) into a new series that the agent can use as part of a larger formula. The operator set spans sign and magnitude transformations, time series operators, cross sectional ranking, distributional features, and conditional logic.

Sign Returns $+1$, 0, or -1 depending on whether the input is positive, zero, or negative. Useful for capturing pure directional information without magnitude. For example, $\text{Sign}(\text{close}_t - \text{close}_{t-1})$ encodes whether the stock went up or down today, without saying by how much.

CSRank (cross sectional rank) On a given day, takes a vector of values across all stocks and replaces each with its rank within that day. With 300 stocks, a stock holding the 50th highest value gets $50/300 \approx 0.17$. This converts absolute values into relative position within the day's cross section, which is the language of cross-sectional alpha investing: what matters is not whether a stock is up, but whether it is up more than its peers.

Product (rolling product) Multiplies values together over a rolling time window. For example, $\text{Product}(1 + r_t, n = 5)$ computes the cumulative return over the past five days. Useful for capturing compounded behavior over a horizon.

Scale Normalizes a quantity to make it comparable across stocks or across time. The exact form depends on the implementation, but it typically divides by an absolute value sum or rescales to have unit norm within the cross section. The point is to remove arbitrary units (price magnitudes, share counts) so the resulting factor is comparable across stocks.

Pow Raises a value to a power, x^p. Useful for emphasizing or dampening extreme values. Squaring ($p = 2$) is common for capturing volatility or quadratic effects; taking square root ($p = 0.5$) compresses outliers.

Skewness A statistical measure of the asymmetry of a distribution. Computed over a rolling window of returns, it captures whether the return distribution has a longer tail on the upside or downside. Stocks with persistent negative skewness (frequent small gains, occasional large losses) tend to behave differently from stocks with positive skewness, and this asymmetry often predicts future returns.

Kurtosis A statistical measure of how heavy tailed a distribution is. Computed over a rolling window of returns, it captures whether the stock has had more extreme moves recently

than a normal distribution would predict. High kurtosis indicates a stock prone to large jumps, which can signal either tail risk or upcoming volatility.

Rank (time-series rank) Different from CSRank. This ranks a stock's current value against its own recent history. For example, Rank($\text{volume}_t, n = 20$) asks: where does today's volume sit in this stock's last 20 days of volume? This captures relative position within the stock's own time series rather than relative position across stocks.

Delta The change in a value over a time horizon, $\text{Delta}(x, n) = x_t - x_{t-n}$. The simplest possible time series operator. Used to construct momentum or mean reversion factors. For example, $\text{Delta}(\text{close}, 5)$ is the five day price change.

Argmax/Argmin Returns the position of the maximum (or minimum) value within a rolling window. For example, $\text{Argmax}(\text{high}, n = 10)$ tells you how many days ago the 10 day high occurred. If it is 9, the stock peaked early in the window and has been declining; if it is 0, the stock is making new highs today. Captures the timing of extremes, which is informational distinct from their magnitude.

Cond (conditional) A ternary if then else. $\text{Cond}(\text{condition}, a, b)$ returns a when the condition is true and b otherwise. Lets the agent build piecewise factors. For example, "use five-day momentum on high volume days, but use mean reversion on low volume days" can be expressed as a Cond expression.

Bibliography

[1] *A beginner's guide to GANs*. Accessed: 21 April 2026.

[2] *Data drift: What it is, why it matters, and how to tackle it*. Accessed: 21 April 2026.

[3] *Understanding alpha in investing: Definition and examples*. Accessed: 14 April 2026.

[4] *How to identify imbalance in the markets with order flow trading*, 2024. Accessed: 13 April 2026.

[5] Hitrotugu Akaike, *Factor analysis and aic*, Psychometrika, VOL. 52, NO. 3, 317-332 (1987).

[6] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Łukasz Kaiser Illia Polosukhin Ashish Vaswani Noam Shazeer, *Attention is all you need*, 31st Conference on Neural Information Processing Systems (2023).

[7] Tim Bollerslev, *Generalized autoregressive conditional heteroskedasticity*, Journal of Econometrics 31, pp. 307-327, North-Holland (1986).

[8] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung, *Time series analysis: Forecasting and control*, 5th ed., Wiley, Hoboken, NJ, 2015.

[9] Center for Research in Security Prices, *CRSP US stock and indexes database data descriptions guide*. Accessed: 21 April 2026.

[10] Minshuo Chen, Renyuan Xu, Yumin Xu, and Ruixun Zhang, *Diffusion factor models: Generating high-dimensional returns with factor structure*, arXiv (2025), available at `2504.06566`.

[11] Junhao Dong and Shi Liang, *Hybrid cnn-lstm-gnn neural network for a-share stock prediction*, Entropy, Artificial Intelligence and the Financial Markets Vol. 27, Issue. 8 (August, 2025).

[12] Jie Fang, Shutao Xia, Jianwu Lin, Zhikang Xia, Xiang Liu, and Yong Jiang, *Alpha discovery neural network based on prior knowledge*, arXiv (2020), available at `1912.11761`.

[13] Robert F.Engle, *Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation*, Econometrica , Vol. 50, No. 4, pp. 987-1007 (1982).

[14] Xavier Glorot and Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the thirteenth international conference on artificial intelligence and statistics, Summer 201013, pp. 249–256.

[15] Ian Goodfellow, *Nips 2016 tutorial:generative adversarial networks*, arxiv (2017).

[16] Jinsong Zheng Guangxi Yu Hao Ni Xin Dong, *An introduction to machine leraning in quantitative finance*, 1st ed., World Scientific Publisging Europe Ltd., 57 Sheldon Street, Covent Garden, London WC2H 9HE, 2021.

[17] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural Computation **9** (1997), no. 8, 1735–1780.

[18] Aaron Courville Ian Goodfellow Yoshua Bengio, *Deep learning, section 5.5, pp. 131-134*, The MIT Press, 2016.

[19] KyungHyun Cho Yoshua Bengio Junyoung Chung Caglar Gulcehre, *Empirical evaluation of the gated recurrent networks on sequence modeling*, arxiv (2014).

[20] Shaoqing Ren Jian Sun Kaiming He Xiangyu Zhang, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, arxiv (2015).

[21] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker, *Time2vec: Learning a vector representation of time*, arXiv (2019), available at `1907.05321`.

[22] Petter N. Kolm and Nicholas Westray, *Improving deep learning of alpha term structures from the order book*, SSRN (23, March, 2024).

[23] Alireza Koochali, PETER SCHICHTEL, SHERAZ AHMED, and ANDREAS DENGEL, *Probabilistic forecasting of sensory data with generative adversarial networks – forgan*, arxiv (2019).

[24] Siddharth Krishna Kumar, *On weight initialization in deep neural networks*, arxiv (2017).

[25] Man Group, *ArcticDB: A high-performance, serverless dataframe database*. Accessed: 21 April 2026.

[26] Harry Markowitz, *Portfolio selection*, The Journal of Finance, Vol. 7, No. 1 , pp. 77-91 (1952).

[27] Simon Osindero Mehdi Mirza, *Conditional generative adversarial nets*, arxiv (2014).

[28] Robert Nau, *Introduction to ARIMA: Nonseasonal models*. Accessed: 14 April 2026.

[29] Guiseppe A. Palelogo, *The elements of quantitative investing*, Wiley, 2025.

[30] Santosh Kumar Sahu, Anil Mokhade, and Neeraj Dhanraj Bokde, *An overview of machine learning, deep learning, and reinforcement learning-based techniques in quantitative finance: Recent progress and challenges*, mdpi, Applied Sciences (2, February, 2023).

[31] Wojciech Zaremba Vicki Cheung Xi Chen Alec Radford Tim Salimans Ian Goodfellow, *Improved techniques for training gans*, arxiv (2016).

[32] RUEY S. TSAY, *Analysis of financial time series*, 2nd ed., A JOHN WILEY SONS, 111 River Street, Hoboken, NJ 07030, 2005.

[33] Melina Vuletic, *Multi-asset financial markets: mathematical modelling and data-driven approaches*, Ph.D. Thesis, 2025.

[34] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar, *Time-series generative adversarial networks* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), Vol. 32, Curran Associates, Inc., 2019.

[35] Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He, *Generating synergistic formulaic alpha collections via reinforcement learning*, Proceedings of the 29th acm sigkdd conference on knowledge discovery and data mining, August 2023, pp. 5476–5486.

[36] Alan J. Marcus Zvi Bodie Alex Kane, *Investments*, McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2014.