Fair Market Value & Chance to Fill Models for OTC Fixed Income Markets (Confidential)

Nathaniel Powell {nathan@deepmm.com}
Founder & CEO, Deep Market Making, Inc.

August 31, 2026

Contents

1 Preface 4
1.1 Our Mission . . . 4
1.1.1 Core Belief . . . 4
1.1.2 How Deep Learning Will Allow Traders to Think More Deeply . . . 5
1.2 Author's Founding Story . . . 6

2 Fair Market Value Introduction 11

3 Fair Market Value Model Competitors 13

4 Fair Market Value Model Use Cases 13

5 Fair Market Value (FMV) User Interface 14

6 API 15

7 Accuracy Translation to Risk Adjusted P&L 15
7.1 Case Study: Dealer/Market Maker . . . 20
7.2 Case Study: Hedge Fund Trader with Competing Real Time Signal . . . 21
7.3 Case Study: Hedge Fund Trader with Competing End of Day Signal . . . 21

8 Chance to Fill Introduction 22
8.1 Finding Opportunities to Improve Quotes . . . 22
8.2 The Chance to Fill Definition . . . 23
8.3 CTF Model Related Work . . . 24

9 FMV Model Design 25

10 CTF Model Design Proposal 25

11 CTF Use Cases 26
11.1 Pricing the Quote . . . 26
11.2 Modeling the State Transitions for Back-Testing . . . 26
11.3 Selecting the Optimal Quote using a CTF Model . . . 26
11.4 Optimality of the Objective Function . . . 29
11.4.1 Setup and Key Concepts . . . 30
11.4.2 Intuition . . . 30
11.4.3 Formal Proof . . . 30
11.4.4 Concrete Example . . . 30
11.5 Proof that Improving FMV Accuracy Improves Dealer Profitability . . . 31
11.5.1 Setup and Key Concepts . . . 31
11.5.2 Intuition . . . 31
11.5.3 Formal Proof (Continuous Case) . . . 31
11.5.4 Concrete Numerical Example . . . 32
11.6 Accuracy in Mid-Level vs. CTF Function Function . . . 32
11.6.1 Setup and Distinction . . . 32
11.6.2 Intuition . . . 33
11.6.3 Formal Proof for Mid-Level Improvement (Not Guaranteed) . . . 33
11.6.4 Formal Proof for CTF Accuracy Improvement (Guaranteed) . . . 34
11.6.5 Concrete Numerical Example . . . 35
11.6.6 Conclusion . . . 35
11.7 Reference Dealer's Objective Function . . . 36
11.8 Simple Formal Objective Function . . . 37
11.8.1 User Parameters of the Objective Function . . . 38
11.8.2 Volatility Measures . . . 39
11.9 Objective Function User Interface (UI) . . . 39
11.9.1 χ Client Rating Slider . . . 40
11.9.2 γ Client Conversion Slider . . . 40
11.9.3 μ Market Share Bonus Slider . . . 40
11.9.4 η Risk Slider . . . 40
11.9.5 ζ Long/Short Bias Slider . . . 41
11.9.6 o Further Adjustment Slider . . . 41
11.10 Incorporating Predictive Alpha Signals for Post-Trade Carry . . . 41
11.10.1 Simulation Design . . . 42
11.10.2 Results . . . 42
11.11 Optimizing Objective Function Parameters with Multiobjective Optimization . . . 43

12 Markets 46
12.1 US Treasuries . . . 47
12.2 European Government Bonds . . . 47
12.3 US Corporate Bonds . . . 47

12.4 European Corporate Bonds . . . 48

13 Current FMV Model Data 48
13.1 Features . . . 48
13.2 RFQ Fields . . . 48
13.2.1 Recent Trade Features . . . 49
13.3 Labels . . . 49

14 CTF Model Data 49
14.1 RFQ Fields . . . 50
14.2 Fields Useful for Predicting the CTF . . . 50

15 Modeling U.S. Credit FMV 50
15.1 FMV Model's Loss Function: MAE . . . 51
15.2 Predicting the Probability Density of the FMV Median Estimates . . . 53

16 Modeling the CTF 53
16.1 Unknowns . . . 53
16.2 Historical $p_{c,i}$ and $p_{r,i}$ Values for EGBs . . . 55
16.3 Calculating $c(a, s)$ from the Distributions of $p_{c,i}$ and $p_{r,i}$ . . . 55
16.4 The Maximum Likelihood Estimation Model . . . 56
16.5 Market Likelihood Functions . . . 57
16.6 Estimating the Distributions . . . 57

17 Modeling the c.d.f. Functions Directly to Model the CTF 59
17.1 Introduction . . . 59
17.2 Model Definition . . . 60
17.3 Back-testing . . . 61

18 Data Relevant for FMV & CTF 62
18.1 Base Features for the CTF Model . . . 62
18.2 US Credit FINRA TRACE Features for FMV & CTF . . . 64
18.2.1 FINRA TRACE Fields . . . 64
18.3 US Corporate Bond Feature Transformations . . . 70
18.3.1 Log . . . 70
18.3.2 Log Return . . . 70
18.3.3 Embedding . . . 71
18.3.4 Relative Time Embedding . . . 71

19 Machine Learning Architecture Choices for the FMV & CTF Models 72
19.1 Why Neural Networks . . . 72
19.2 Other Relevant Common Advanced Deep Learning Algorithms . . . 74
19.2.1 Recurrent Neural Networks . . . 74
19.2.2 Transformers/Attention Networks . . . 76

20 Feature Importance for the FMV and CTF Models 79
20.1 Introduction . . . 79
20.1.1 Using Background Sample . . . 79
20.1.2 Percent Change in Regression Output . . . 80
20.2 Enhancing Evaluated Pricing Engines and Recommended Quoting Systems for Traders . . . 80
20.2.1 Sell-Side Traders . . . 80
20.2.2 Buy-Side Traders . . . 81
20.2.3 UI Display for Ranking Features . . . 81
20.2.4 Advanced UI Enabled by SHAP Feature Importance . . . 82
20.3 Calculating SHAP Feature Importance Values . . . 84
20.4 Important Properties of DeepSHAP Values . . . 85
20.4.1 Local Accuracy . . . 85
20.4.2 Missingness . . . 85
20.4.3 Consistency . . . 85
20.4.4 Conclusion . . . 86

References 86

A Objective Function Alpha Parameter Python Simulation Code 89

1 Preface

1.1 Our Mission

Our mission is to build the AI with the most accurate predictions of the probability of future events.

We started with capital markets, so much of the forthcoming discussion is regarding how to elevate traders, but this future prediction capability can be applied to many other industries and use cases.

1.1.1 Core Belief

Everyone on Earth who wants to will be able to raise their mental understanding of their world, relationships, art, science, emotions, personal fulfillment, with some effort, but standing on the shoulders of various AI systems. How this works is rooted in how our human mind works, and likewise, how modern artificial Neural Networks (NN) function. For traders, we want to help elevate their ability to think strategically, better seeing the forest from the trees. In other words, spending less time analyzing raw market data, and more time thinking strategically, talking to clients, and ultimately profiting as a trader because the AI is helping perform the low-level processing of the market data, surfacing the relevant information when and where it's needed. Out of the machine learning paradigms available today, only deep learning has the capability to learn from

traders how to surface the right information, as all other ML algorithms are just too simple to even begin to approach the mental complexity of the mind of a trader.

1.1.2 How Deep Learning Will Allow Traders to Think More Deeply

Different tissues in the brain developed during different stages of evolution. The parts of our brain we share in common with reptiles include, among others, the emotional centers like the amygdala, and the cerebellum, which helps us to subconsciously plan our movement by solving differential equations [2], and these parts originally evolved hundreds of millions of years ago. The mammalian tissue, the neocortex, is the largest portion of our human brains, and is much larger than in other great apes. The reason the size of the neocortex makes such a large difference in our intelligence versus other animals is that it allows much deeper layers, which allow for much more abstract reasoning with high-level concepts than other animals can achieve.

The neocortex is made up of about 300 MM pattern recognizing units, arranged in a vast network of parallel hierarchies. Each pattern recognizing unit has about 100 output neurons that light up in different codes depending on which pattern has been recognized (each unit can recognize many different patterns). Different parts of your brain may be responsible for processing different types of information, but the remarkable thing is that every pattern recognizer is functionally equivalent. The reason the neocortex ends up being organized in certain ways is simply because of how inputs and other brain anatomy is wired to it. For example, the visual cortex is in the back of your head typically because your eyes are connected to the back of the neocortex. [18]

Pattern recognizers which happen to be near sensory inputs like our optic nerves recognize low-level patterns – such as lines. The output of the these pattern recognizers are fed to the next layer in the hierarchy to recognize a slightly more abstract concept – such as shapes. [7] Then parts of faces, for example, then whole faces, then expressions, then a high level concept like "this person is happy." Then this might feed into your understanding of a social situation, such as "Mike pulled a funny prank on his buddy and is giddy." Since there are many high level concepts that you can recognize, there are many such hierarchies that all share lower level pattern recognizers with each other.

As alluded to earlier, the limit to how abstract, high level our thinking can become is limited to the number of layers in our concept hierarchies our brains are capable of wiring. Therefore, if we want to elevate our thinking even higher, we need mental prosthetics of one kind or another. In the past those prosthetics often took the form of other people; in the financial world you might have a team of analysts, associates, junior traders, etc. who can help you process low level information so that you can think of the big picture. The problem with this model is that you can only scale it so far before it becomes prohibitively expensive.

You might be thinking – what about all of the quant finance models, can't they be a mental prosthetic as well? Yes – they can be and they are to a very small degree. The problem lies in the lack of complexity of these models, and the profound and untrue assumptions they make about the markets. For example, simple economic models often assume that there is perfect competition, or that the markets have already reached

equilibrium, because those assumptions allow a simple equation to function. But there is a reason that we don't hand high-notional OTC trading over to simple formulas and heuristics – they are just too simplistic and would too easily be taken advantage of, unlike the typical human trader who is constantly learning and evolving, with a truly giant 10 to 100 trillion synapse neural network in his head helping him understand the market complexity.

However, now we have a new option for scaling our teams' information processing capabilities, with the advent of robust deep learning models. Similarly to biological neural networks, the most effective neural networks are very deep, with many layers, and their processing capability is increasing exponentially year after year through improving hardware efficiency, software efficiency, and larger amounts of capital being devoted to training the largest networks. So now every member of your credit team, including the analysts, can now stand on top of many more layers of analysis performed by deep neural networks, which will allow you as a leader to see the forest from the trees at a true 10,000 foot perspective, whereas now you may be stuck below the ridge under a canopy thinking about raw incoming, publicly available data points, rather than focusing on the unique information you have in your possession which will allow you and your team to outperform your competitors.

Just as the evolution of our larger brains allowed us to develop abstract reasoning, mathematics, science, art, and music, all of which our great ape cousins are incapable of, think of the abilities you will be able to invest in as a team if you have a much deeper understanding of your market environment at all times, as Ray Kurzweil, the futurist, explains in his many speeches regarding his (accurate) predictions of the future of technology. At Deep MM our machine learning algorithms primarily learn from the traders via the market data they generate, and although these models are still tiny compared to the mind of a human trader, they are at least orders of magnitude more complex than the traditional ML models employed by our competitors as of this writing.

1.2 Author's Founding Story

Most of my 16-year career has been in the technology industry at companies like Amazon and Bing@Microsoft, where I had my longest stint of over 6 years. Most of my tenure at Bing was in the Monetization group, responsible for the final placement of ads on the search results page. While I was there, our small group of less than ten people made A/B tested changes worth over $2 billion in ad revenue per year. Most of that money was gained by making various changes to the UI, but for most of my time in the group I was the only data scientist or ML researcher and I managed to squeeze an additional over $100 million in revenue out of the ads, without on net hurting the user experience. These changes I made were not obvious changes, but were significant when aggregated over many pages. They were things like using machine learning to find when are the most opportune times to make the ads bigger for queries (some of which we had never seen before) where the user was more likely to click on a helpful ad, and make them less prominent for queries where ads are almost never clicked on and only serve to distract the user from websites they want to visit. My biggest contribution to revenue, however, was

in designing a continuous A/B testing system that would continually monitor metrics like ad clicks, revenue, etc. on the individual components of the ad interface. (There were various elements of ads that you could click on such as ratings, click to call, etc.). The year that this system was activated, it quickly surfaced a component of the ads that was actually decreasing the ad clicks such that we were losing about $60 million a year (which was surprising since most ad elements increased ad click-through).

My largest technical contribution occurred towards the end of my tenure at Microsoft; I created a deep-learning based method to translate user queries like "convert fifteen and a half dollars to euros" into a machine query like "USD:EUR:15.50" so that the answer could be looked up and directly displayed to the user. This framework is still being used at Bing last time I checked, but is likely soon to be supplanted because of Bing's partnership with Open AI. This type of large language model will help us build interfaces to the FMV and CTF models discussed in this paper that will be even more intuitive and human like than the graphical user interfaces we have been building.

I helped the experimentation team create additional tools such as one that allowed our engineers to find query examples that helped to explain the metrics they were seeing on an A/B test scorecard.

I worked with Dr. Susan Athey, the chief consulting economist of Microsoft at the time, on a patent application that improved the way that we ranked ads so that they reward advertisers for truth telling – in other words favoring ads that tell users what they can expect to find should they click on the ad, rather than those annoying ads that misrepresent the landing page, resulting in a quick click-back by the user. From Dr. Athey I got my causal inference bug, which will be discussed in the Chance to Fill sections of this paper, particularly in section 10.

Some of my lessons learned I took with me from the technology industry, and Micorosoft in particular, I gained some take-aways I cherish:

1. The lessons learned observing about 1,000 experiments and from Ronny Kohavi who ran our experimentation group which enabled our little group to run hundreds of A/B tests ("flights") on our users per year:
   (a) Most ideas brilliant people have fail to help the business metrics we care about
   (b) Most organizations rely on the highest paid person's opinion for decision making (a "HiPPO"). Data-driven organizations run an experiment to make a decision.
   (c) Most people don't know how to design experiments correctly – creating a centralized, reliable framework or library helps mitigate this problem
   (d) Observational or retrospective A/B tests or studies are generally garbage
   (e) When top leaders decided to "go rogue" and ship features that the metrics clearly showed were bad – it doesn't end well.
2. Causal inference is key to making ML perform well in the wild

3. Deep Learning performs orders of magnitude better than previously-developed machine learning techniques

When leaders launch features wherein the A/B test shows it's bad for our users and/or revenue in the hopes that the numbers will change in the long-term because it's such an important or good feature, it pretty much never works out long-term. This is what happened, for example, with Bing's partnership with Facebook to give search social features.

On deep learning, I was taught about deep learning by leaders in Microsoft Research (MSR), who were some of the pioneers in deep learning when it was first developed. Perhaps because I was exposed to this reality very early in the history of deep learning, so that by the time I moved to the banking industry in 2017, I already had supreme confidence that deep learning can solve previously unsolved problems in banking just as it had in other industries, a belief very few shared at the time, and few seem to share even to this day, as all of our competitors still seem to be built on obsolete machine learning techniques.

That confidence didn't just come from hearing other people at Microsoft talk about how great deep learning was and seeing how hundreds of millions of dollars if not billions were being invested in deep learning-related initiatives at Microsoft, both company wide, but also within Bing and MSR, but more importantly from learning and understanding how deep learning is different than traditional machine learning approaches which are only capable of solving very simple problems compared to the possibilities enabled by deep learning, and it primarily comes down to how deep learning can process hierarchical information through having many interconnected layers similar to the human brain as discussed above in section 1.1.2.

Given my experience working with Dr. Athey, and my personal interest in economics, particularly Austrian economics, and because I thought bonuses were often larger in the financial industry than in the technology industry (the year I was key to Microsoft making $60 MM they gave me a $30k bonus), I had started interviewing with various New York financial firms. I originally imagined that I would work at a high frequency trading firm like Renaissance Technologies or Two Sigma, but all of the firms that I interviewed with were not in the market for a machine learning applied researcher like myself, and were not particularly impressed given my lack of low-level C++ skills. This obviously was partially explained by how machine learning might not be the greatest fit for HFT, it's a much better fit for medium-frequency trading or human traders, given the high computational requirements of large models, but I think more importantly it was a symptom of how far behind the financial industry overall has been in its adoption of AI.

More than a year after I started interviewing in New York I landed with an avant-guard technology group at JP Morgan led by Gene Fernandez, who later became the CTO of FactSet. After he left the group shortly after I arrived, it seemed to lack the political power needed to enact the change necessary to adopt AI, as the group was split up and my smaller group was assigned to a leader who didn't seem to know anything about AI or ML. Fortunately, Dr. Geoffrey Zweig allowed me to transfer to his group

after he said he was impressed by my presentation regarding applications of deep learning and deep reinforcement learning to finance.

Dr. Zweig is a former leader at Microsoft Research before he joined JP Morgan. The last year he was at MSR his name was on over 100 publications. He was a great mentor for our attempts to apply machine learning to some problems. Unfortunately, the bank ultimately was not ready to utilize his great skills, and he went to Facebook New York after his first year. Our team later became the Machine Learning Center of Excellence (ML CoE). Before Dr. Zweig left, with his support I came up with my own Long-Short Term Memory (LSTM)-based deep learning design for pricing U.S. corporate bonds using FINRA TRACE data, and implemented the first prototype. This model eventually went on to be twice as accurate for High Yield (HY) bonds than the Kalman-filter approach developed by the Quantitative Research (QR) team, and four times more accurate than Bloomberg's benchmark real-time pricer for HY bonds, as well as significantly more accurate for Investment Grade (IG) bonds as well. This model later went to production.

While part of the ML CoE, I also worked with a quant trader to research my ideas around using a Chance-to-Fill (CTF) model combined with an objective function to price offers, which I prototyped and showed that it would be a huge potential benefit to the trading desk's metrics such as spread capture (short term P&L), market share / fill rate, etc. Since leaving JP Morgan I have created a completely new design for the CTF approach to market making introduced in section 8.

Around this time Dr. Chak Wong joined the ML CoE as the main Managing Director reporting to Dr. Lidia Mangu (who had gone on to lead the ML CoE after Dr. Zweig left), and Dr. Wong became the leader of all of the quantitative side of the ML CoE (there was also a natural language processing group). Dr. Wong was a good mentor as well, who persuaded me to do a better job documenting my ideas and fleshing them out before implementing them (for example as I have in this paper). Dr. Wong's view on the solution to market making in credit trading was to implement it using reinforcement learning, a shared passion for me as well.

Reinforcement learning is a subfield in AI / ML which is unsupervised learning. In supervised learning, we give the models labels which are the expected outputs we want the model to learn to produce (given its inputs). In reinforcement learning, the model tries to maximize the reward as it takes actions in an environment. Reinforcement learning has shown promise by dominating in games. The AlphaGo [32] system developed by DeepMind, for example, was the first computer system to beat the world champion of the game of Go. More recently, ChatGPT by Open AI, was improved for chat functionality through applications of reinforcement learning. So in summary, reinforcement learning needs:

1. Environment which is updated by the agent's actions and other updates to the environment's state (such as from moves by the other Go player)

2. Rewards returned by an environment during an episode of interaction between the agent and the environment.

Dr. Wong's view, as I understood it, and as discussed widely in JPM's conference interactions / recruiting lighting talks at their booth, etc., is that the reward for a reinforcement learning over-the-counter (OTC) credit trading quoting system should be long-term risk-adjusted P&L. So this P&L would encompass all aspects of the trading life cycle, including the full change in the book's mark-to-market, trading P&L, etc. Of course additional constraints could be added to such a system like risk limits, other guardrails, etc. So under Dr. Wong's model, risk-adjusted P&L would be the reward for the agent.

Then for the environment in which the reinforcement learning agent is trained, it wouldn't be necessarily sufficient to train a reinforcement learning agent on just historical data. For one thing, this historical data will not react to the agent's actions, and another problem is that the historical data may end up biasing the model's actions towards strategies that worked well in the past but don't translate to the new environment because financial time series data is notoriously non-stationary in its distribution. The solution then would be to create a highly realistic simulation environment that would generate many outlier events, be unbiased according to market conditions by design, and still have the stylistic features of real market data, such as no more than very short-term autocorrelation, stochastic process attributes, etc.

There are two (in my view) showstopping problems with this approach relative to alternatives, in going to production with this approach:

1. Trading desks are not going to be happy with a static reward objective function. Trading desks are constantly changing their preferences. And the problem with a reinforcement learning agent is that if the reward function changes, the model needs to be retrained, re-evaluated, and deployed, which could take a long time, certainly not fast enough to react quickly to changing market conditions and thus desk preferences.

2. The realism of any kind of simulated environment for reinforcement learning agents in financial markets is still very much an open area of research, one where a reinforcement learning agent could learn to take advantage of any weaknesses or patterns present in the simulation environment which will not generalize well to the real world.

For these reasons we have taken a different approach. In our CTF sections, we discuss how our system allows trading desks to constantly change their objectives and preferences and have the quoting system instantly adjust for those changes, while still taking advantage of our AI analyst system which is able to not only factor in large amounts of data for every recommendation, but also explain its decisions.

We also present the Fair Market Value (FMV) model, which assists in creating a base price level off of which to use in quoting decisions or decisions on the buy-side.

Another point of disagreement between Dr. Wong and myself that also helped lead to my exit from JP Morgan, is that Dr. Wong seemed to care nothing about empirical results. If I approached him with promising results for a particular model he would just

shrug his shoulders and say "I don't know." He wanted the model to be mathematically "proven," which is the typical quantitative research approach in finance. The big problem with this approach is that deep learning, as a method to model various types of data, was proven far superior to mathematically "proven" methods many years before researchers began to grasp why mathematically. Also – for a *particular* dataset, it's not really possible to prove that a particular deep learning architecture will have some error bounds or meet other objectives, as there are too many idiosyncrasies and frankly fine details in the real-world data. You can prove it's a good model by assuming a particularly simplistic data generation process, but if you're going to pretend that the data generation process is that simple, then you could also pretend that a simplistic model will be the best idea – and that is in fact what the quantitative researchers have done at our competitors like MarketAxess, TradeWeb, TruMid, and probably Bloomberg as well with their new IBVal model. And this is why it turns out our fair market value (FMV) model is far more accurate than our competitors' (see our separate accuracy report for details), as we have focused on incrementally improving our empirical results over the past three years of Deep MM's existence, *even though we use only a subset of the data sources to which our competitors have access.*

2 Fair Market Value Introduction

The U.S. corporate bond market is mostly an over the counter (OTC) market, wherein there are a limited number of dealers / market makers who receive and place voice or electronic requests for quotes (RFQs). The U.S. credit market is not generally an exchange traded market place because of a lack of liquidity in terms of the number of trades per issue. There are many bonds which don't trade at all on a given market day, particularly in the high yield space.

In over-the-counter (OTC) markets, a client sends a request for quote (RFQ) to one or more dealers through various OTC electronic platforms or over the phone to buy or sell the instrument, and dealers choose to respond with a quote at which they are each willing to buy or sell. The client will then choose to trade one or zero of the quotes.

Both the sell-side dealer and the buy-side trader come to a view of the fair market value of a particular issue of interest; the dealer must have this view in order to create a quote according to their objectives. If the dealer wishes to quote aggressively in order to win the trade, they need to know where their hypothetical trade stands compared to the current mid market and buy-sell spread. If they wish to quote in a way that ensures a high P&L should they get filled, they need to know that their quote is favorable to the dealer relative to current market conditions. Likewise, the buy side trader needs a view of the fair market value in order to decide whether they should accept the best quote they received from the dealer(s), negotiate for a better price, or wait.

When an OTC sell-side or buy-side U.S. credit trader has a desire to buy or sell a specific bond, they are usually trying to achieve a price relative to the current mid level and the buy/sell spread of the issue. There are many ways that the trader can calculate the mid level price and spread, including looking at the most recent prints for the same

bond or related bonds, using a traditional quant-finance mathematical model, an in-house machine learning approach, or a vendor product for End of Day (EOD) prices or real-time pricing signals such as the MarketAxess Composite+ (CP+) or Bloomberg's benchmark price.

The author designed and implemented the very first prototype of the Deep Bond Pricing (DBP) model that went into production at JP Morgan around 2020. At JP Morgan, we implemented a Long Short Term Memory (LSTM)-based approach, similar to extensive published research about using LSTM's for time series data, such as [10], [23], [31], [17], [37]. This work is from the Machine Learning Center of Excellence, an applied research group serving all lines of businesses for JP Morgan, including the Corporate and Investment Bank (CIB). This DBP model was found to be much more accurate than the Kalman Filter approach JP Morgan had previously taken for both high yield and investment grade bonds. The model we present in this paper is an entirely new neural network implemented by Deep Market Making, Inc., which incorporates design elements not invented at the time the author was employed by JP Morgan.

One difficulty with OTC trade price modeling compared with traditional linear time series models such as auto-regressive (AR), ARIMA, etc., is the lack of clarity about how to use information from different *types* of trades to map to the price of a particular *target* trade. The Kalman filter approach solves this problem by explicitly defining linear transformations from a particular observation to a state space, and then another set of transformations which map from the state space to a pricing estimate for each particular type of RFQ, such as buy, sell, or dealer RFQs of various quantity buckets. These filters are typically calibrated to a set of training data, similar to machine learning approaches.

The main advantage of neural network approaches compared with a Kalman Filter approach is that while the Kalman Filters are structurally linear models, meaning any non-linear features must be specifically designed, neural networks are inherently non-linear. Neural networks that are wide [15] or deep enough [13] can theoretically be designed to approximate any function, including between fixed income OTC time series data and the current fair market value, if we can find a way to transform the data such that the assumption that the data generation process is stationary (meaning that the distribution of the training samples and out-of-sample data are the same). A cursory search of the literature reveals no such claim that Kalman filters are universal function approximators.

How does this accuracy improvement translate to the bottom line? A more accurate price estimate reduces the uncertainty about whether the quoted price is good or bad relative to the market. Reducing uncertainty about the market price reduces the variance of the return/P&L when executing any trading strategy based on this pricing information, because the difference between a trader's expectations of the market price and the actual market price is one of the sources of uncertainty in P&L of the trade. If the trader is measuring their risk-adjusted returns using the Sharpe ratio, then their Sharpe ratio will be increased by reducing the standard deviation of the P&L caused by pricing uncertainty. See section 7 for more discussion about measuring the impact of an improvement in pricing accuracy in terms of risk-adjusted P&L.

The model uses attributes of a proposed RFQ as well as recent transactions from the same bond and related bonds (with the same issuer, industry, sector, credit rating), along with all of their attributes, to estimate the price. The model is trained using a variant of stochastic gradient descent on GPU hardware. The loss function is based on mean absolute error (MAE) rather than mean squared error (MSE) as MAE results in the model estimating the median of the trades with the specified attributes, which yields a more realistic, smoother curve. It takes about 1-2 weeks to train the model on a single GPU, and we will scale this up to a GPU cluster as we continue to increase the model's design sophistication.

For a buy-side trader, it may be enough to know the fair-market value which is defined as our model-estimation of the median price of a trade given the current state of the market and the properties of the trade's RFQ. However, for a market maker, they must optimize their objective function, which means they need to know the probability that their counter-party will accept their quote in response to an RFQ. The fair market model is currently implemented and is rolling out to production, and our next product after FMV will be the chance to fill (CTF) model also discussed in this paper.

3 Fair Market Value Model Competitors

We see as competitors any provider of a pricing signal for U.S. (and in the future European and other region) corporate bonds. This includes companies like MarketAxess, ICE-CEP, Bloomberg that all provide real-time signals, providers of EOD prices like EDI (exchange-data.com), as well as providers of TRACE intraday or EOD prints with no pricing model.

To learn about the accuracy of our model compared to competitors, please contact us for a demo.

Our model has a greater degree of customization ability than many of our competitors. Our model is capable of not only considering the size and side of the transaction, but other attributes of the RFQ as described under the question below about the inputs for the model.

4 Fair Market Value Model Use Cases

There are at a minimum the following possible use cases for the fair market value model:

1. Real-time updates of credit curves
2. Increasing the risk adjusted P&L such as sharpe ratio through reducing the standard deviations of the return.
3. Knowledge is power! The model will be helpful in negotiating a better price with counterparties in a voice trade.



Figure 1: User interface of the fair market value model

4. Can be helpful in deciding not only what price to trade at but when is an opportune time to trade.

5. Fully automated trading on the buy or sell - side. Could be an input for an algorithmic trading system. On the sell side, for example, it identifies the current buy/sell spread against which the market maker can measure their spread capture, typically one of the most important KPIs for any dealer.

6. Intraday algorithmic portfolio risk analysis, hedging, and rebalancing (part of which needs accurate pricing for all bonds in a portfolio).

7. Portfolio trading needs an accurate price when calculating the intraday mark-to-market of all of the bonds in the group of bonds to be exchanged.

8. Once we start calculating the empirical distribution by dropout-sampling from our already trained model, traders will have some sense of the probability of a given trade arriving at different price levels.

5 Fair Market Value (FMV) User Interface

Figure 1 is a screenshot of the issuer-level interface that we can demo when we meet. This UI was implemented by Brian Adams and Oleksandr Oliinyk and designed by Dung Tien Le. We also have another view of the model which shows a real-time credit curve for an issuer, credit rating, etc, and how it has changed over time, and more.

```
{
    "fmv_stream": {
        "figi":"BBG00JXGN2L3", "size":"10", "trade_count": 20
    }
}
```

Listing 1: JSON example of the client message to our websocket server

The bond-level view shows the FINRA TRACE-reported trades (triangles) with the model's predictions for buy, sell, and dealer-to-dealer trades shown as solid lines. On the upper left there are the current price estimates from the model. In the table on the left each of the recent trades show the actual price along with the model's predicted price. This UI sits on top of a JSON Secure Websocket connection API. This is currently a prototype that we hope to have in production within the next couple of months. We are collocating in the NYC area, with a direct connection to the FINRA TRACE multi-cast. We can also install an appliance on-prem if desired.

6 API

The client can make a secure websocket connection to our server and send JSON formatted like that shown in listing 1.

Sending this message begins a continuous stream of FINRA trace print "trade" objects which include our model's customized prediction for them which allows an accuracy comparison to be made by the client. Also in the stream are model predictions which are customized to the size of the RFQ specified in the client message for the particular bond specified. An example of this continuous stream is shown in listing 2.

We are also working on implementing another API stream which can send in real-time a credit curve and credit curve delta.

7 Accuracy Translation to Risk Adjusted P&L

As of this updated in Q4 2025, given the validation we've received from multiple large sell-side banks, we are more accurate than our top competitors for trades larger than 150,000 in our universe of bonds (both in IG and HY).. This comparison can be done by using each of the competing signals across a historical, out-of-sample (for our model) time period to create an estimated price for each of the trade prints in the FINRA TRACE Enhanced Historic Data. Then, we can compute the MSE, RMSE, and MAE as follows:

```
{"model_price": [{
    "date": "2020-01-01 04:13:28",
    "prices": {
        "B": 104.633100212487,
        "S": 104.84089007315279,
        "D": 104.74209533718468
    }
}]}
{"trade": [{
    "record_num": 53,
    "report_date": "2020-01-01 08:05:45",
    "execution_date": "2020-01-01 08:05:44",
    "side": "B",
    "quantity": 60000,
    "price": 104.408,
    "model_price": 104.63141310301663,
    "yield": 3.902168
}]}
{"trade": [{
    "record_num": 54,
    "report_date": "2020-01-01 08:05:45",
    "execution_date": "2020-01-01 08:05:45",
    "side": "S",
    "quantity": 60000,
    "price": 104.47,
    "model_price": 104.91399872389675,
    "yield": 3.890543
}]}
{"model_price": [{
    "date": "2020-01-01 11:54:18",
    "prices": {
        "B": 104.5624468108341,
        "S": 104.7317530741617,
        "D": 104.66185359616577
    }
}]}
```


Listing 2: JSON example of the client message to our websocket server

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1}$$

Where y is the actual trade print i's price and $\hat{y}$ is the signal's estimated price, for all n trade prints in the out-of-sample data set. The RMSE is the square root of the MSE:

$$RMSE = \sqrt{MSE} \tag{2}$$

And MAE is the following:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{3}$$

If one were to compare the accuracy gain with that of one of our competitors, such as a real-time or EOD model, how do we quantify the benefit of our model in practice? A naive approach is to make the simple assumption that traders or trading algorithms will manage to capture some percentage of the accuracy gain in improved spread capture. Spread capture is a Key Performance Indicator (KPI) which indicates, particularly for market makers but also other relatively high-frequency traders, how much of the buy/sell spread the trader has been able to capture in immediate, albeit in a mark-to-market sense when entering a position, P&L. If the trader captures 100% of the spread, under our definition of a fair market value as the median trade price the trader was able to enter a position at the buy price and exit the position at the sell price. If the trader is able to achieve 200% spread capture, then they buy at a price twice the distance from the buy/sell midpoint than the buy price, and also sell at twice the distance from the midpoint than the sell price.

Before using our improved model, we could assume that the trader is achieving a spread capture c, and with our model a capture of:

$$c' = (MAE - MAE')c \tag{4}$$

where MAE is the MAE of the signal used before adopting the Deep MM model, and MAE' is the MAE of the Deep MM model, and c' is the spread capture P&L metric after switching to using our model. Note that this assumes the improvement is directly proportional to the absolute error reduction; however, it's unrealistic to assume that the trader will be able to capture all of the pricing accuracy gain in terms of P&L. For example, a retail trader with a very small trade going through a second-tier opaque broker may not see an immediate improvement in pricing even if they know what the price is for someone with better access to the market. Therefore, we propose some haircut h between 0.0 and 1.0 which will reduce the gain. For example, an assumption of a 10% haircut ($h = 0.1$).

$$c' = h(MAE - MAE')c \tag{5}$$

However, this choice of $h = 0.1$ is arbitrary. A more serious problem is that spread capture is usually measured against a pricing signal's estimate of the midpoint, as well as buy/sell spread. If we're changing from one pricing signal to another more accurate signal, the spread capture between the two metrics are not exactly comparable. You could measure both against the better Deep MM signal, but even Deep MM's signal is not perfect.

A different approach is to look at the improved signal's effect on the standard deviation σ of the spread capture part of the trader's P&L. This is because a trader is likely not optimizing for absolute P&L, but rather a risk-adjusted P&L. A common way of measuring risk-adjusted P&L performance is the Sharpe Ratio r_s:

$$r_s = \frac{r - r_f}{\sigma} \tag{6}$$

r is the expected return, the return on investment, for our purposes a single trade:

$$r = \mathbb{E}[\frac{v_{t+1} - v_t}{v_t}] \tag{7}$$

σ is the standard deviation of the return on investment:

$$r = \sqrt{VAR(\frac{v_{t+1} - v_t}{v_t})} \tag{8}$$

where v_t is the value of our portfolio invested/at risk in each trade, and v_{t+1} is the mark to market value of the portfolio after the trade is completed. We assume that there is no change in bias between the existing model signal with regards to the trader's behavior, because once acclimated to any signal we assume the trader will adjust to any such bias. Therefore we assume that r does not change when we go from one pricing signal to another.

r_f is the risk-free return, or the interest rate over an identical time period for an investment vehicle considered to have zero risk, typically a short term U.S. Treasury Bond. We will define the counterparts r'_s, and σ' as the equivalent values if traders were to use our FMV model, rather than the competing signal. We assume that r is not affected by the change to our model, although if there is an improvement in r'_s relative to r_s, then obviously the trader could choose to convert some or all of these gains into in an improved r by allowing them to execute riskier trades with higher mean rewards. But for now we will focus on the improvement in r_s, which, if it could be improved significantly, would be transformative for the firm in question. We will also focus on the spread capture component of r and r_s, so when we say r in the subsequent discussion in this section, we mean the returns due to spread capture.

We can solve for Δr_s, which we define as the percent change in the Sharpe ratio r_s going from a competing model to our FMV model, as follows:

$$r_s = \frac{r - r_f}{\sigma}, r'_s = \frac{r - r_f}{\sigma'} \tag{9}$$

$$\Delta r_s = \frac{r'_s - r_s}{r_s} = \frac{\frac{r-r_f}{\sigma'} - \frac{r-r_f}{\sigma}}{\frac{r-r_f}{\sigma}} = \frac{\frac{1}{\sigma'} - \frac{1}{\sigma}}{\frac{1}{\sigma}} = \frac{\sigma}{\sigma'} - 1 = \frac{\sigma}{\sigma'} - \frac{\sigma'}{\sigma'} \tag{10}$$

$$\Delta r_s = \Delta\sigma = \frac{\sigma - \sigma'}{\sigma'} \tag{11}$$

Equation 11 implies that if we can improve σ by $x\%$, then we can also improve the Sharpe ratio r_s by $x\%$.

We define S_0 as $v_{t+1} - v_t$ for a single trade. Assume that our spread capture P&L random variable S_0 for a single trade is composed of the sum of two random variables A and B:

$$S_0 = v_{t+1} - v_t \tag{12}$$

$$S_0 = A + B \tag{13}$$

A represents the difference between the pricing signal for a trade used by the trader and the actual (but unknown to the trader) fair market value for the bond and RFQ in question. For some trades, A will be positive and sometimes it will be negative. B represents other sources of spread-capture P&L, such as how well the trader timed the transaction, negotiated the trade, etc.

Assuming for simplicity that A and B are independent, then:

$$VAR(\frac{v_{t+1} - v_t}{v_t}) = VAR(\frac{S_0}{v_t}) = \frac{1}{v_t} VAR(S_0) \tag{14}$$

$$VAR(S_0) = VAR(A) + VAR(B) \tag{15}$$

$$VAR(\frac{v_{t+1} - v_t}{v_t}) = \frac{1}{v_t}(VAR(A) + VAR(B)) \tag{16}$$

If we want to discuss not in terms of just one trade but across a group of trades over a *longer* time period we will call T and $T+1$, then we define the random variable S as the spread capture $P\&L$ across our time period transition from T to $T+1$ composed of the sum of m independent trials/trades of S_0. S is the sum of the spread capture when using a competing pricing signal, whereas S' is the equivalent when using our FMV model pricing signal. We can likewise redefine our σ variables as the standard deviation of Δv_T across this entire time period.

$$S = \sum_{i=1}^{m} S_0 = mS_0 \tag{17}$$

$$\mathbb{E}[S] = m\mathbb{E}[S_0] \tag{18}$$

$$VAR(S) = mVAR(S_0) = m(VAR(A) + VAR(B)) \tag{19}$$

$$VAR(S) = m(MSE + VAR(B)) \tag{20}$$

$$VAR(S') = m(MSE' + VAR(B)) \tag{21}$$

$$\sigma = \sqrt{\frac{VAR(S)}{v_T}} = \sqrt{\frac{m(VAR(A) + VAR(B))}{v_T}} = \sqrt{\frac{m(MSE + VAR(B))}{v_T}} \tag{22}$$

$$\sigma' = \sqrt{\frac{VAR(S')}{v_T}} = \sqrt{\frac{m(VAR(A') + VAR(B))}{v_T}} = \sqrt{\frac{m(MSE' + VAR(B))}{v_T}} \tag{23}$$

If we plug in the σ and σ' into equation 11, we get:

$$\Delta r_s = \frac{\sqrt{\frac{m(MSE+VAR(B))}{v_t}} - \sqrt{\frac{m(MSE'+VAR(B))}{v_T}}}{\sqrt{\frac{m(MSE'+VAR(B))}{v_T}}} \tag{24}$$

$$\Delta r_s = \frac{\sqrt{MSE + VAR(B)} - \sqrt{MSE' + VAR(B)}}{\sqrt{MSE' + VAR(B)}} \tag{25}$$

Notice we have shown that m and v_T cancels out in equation 25, so whether we're talking about the percentage increase in the risk-adjusted P&L for a single trade or for a set of trades over a set time period, the percentage increase ΔS will be the same. Additionally, the percentage increase in Sharpe ratio r_s will be the same whether we're drawing a large or small amount of capital v_T to trade with. Therefore, we can convert our improvement from MSE to MSE' to improved P&L assuming a VAR(B) for a particular trader by simply plugging in those numbers into equation 25 as we do below in our case studies.

7.1 Case Study: Dealer/Market Maker

Our Deep MM FMV model has a pricing percentage MSE out of sample of approximately 0.812. Suppose that we're competing with a real-time signal with a pricing percentage of MSE of 1.16 for a market maker with other sources of spread capture standard deviation of 20 basis points, then we calculate Δr_s as follows:

$$\Delta r_s = \frac{\sqrt{MSE + VAR(B)} - \sqrt{MSE' + VAR(B)}}{\sqrt{MSE' + VAR(B)}} \tag{26}$$

$$= \frac{\sqrt{1.16/10,000 + 0.002^2} - \sqrt{0.812/10,000 + 0.002^2}}{\sqrt{0.812/10,000 + 0.002^2}} \tag{27}$$

$$= 16\% \tag{28}$$

We divide the MSE by 10,000 because mean squared error's unit in the case of bond prices is percentage squared, so instead of dividing by 100 as we normally do when converting a bond price percentage into a raw number, we have to divide by 100^2, which is 10,000.

Given that, and the assumptions and simplifications we've made, using our model would improve spread capture risk adjusted P&L/Sharpe ratio for this hypothetical dealer by 16%. This illustrates the large impact on a dealer's business from having a significantly improved fair market value pricing model.

7.2 Case Study: Hedge Fund Trader with Competing Real Time Signal

In this case, we do similar calculations as for the dealer in section 7.1. The only difference in the underlying assumptions is that the hedge fund trader, as they have fewer resources than a dealer, will have a standard deviation in other causes of variance in the spread capture of 35 basis points instead of 20. Therefore:

$$\Delta r_s = \frac{\sqrt{1.16/10,000 + 0.0035^2} - \sqrt{0.812/10,000 + 0.0035^2}}{\sqrt{0.812/10,000 + 0.0035^2}} \tag{29}$$

$$= 15\% \tag{30}$$

So although there is a slightly lower impact on a percentage basis of spread capture risk adjusted P&L, it is still a very significant one assuming sufficient trading volume for the hedge fund trader such that the spread capture/execution cost P&L is a high enough percentage of the overall firm's ROI.

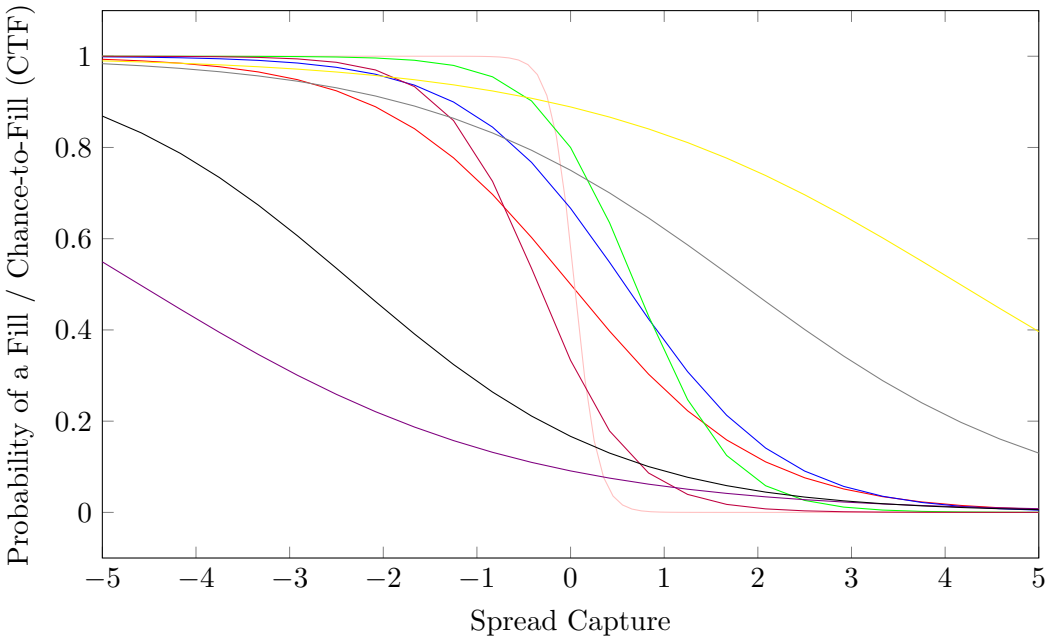
7.3 Case Study: Hedge Fund Trader with Competing End of Day Signal

If we do the same calculation as in section 7.2, but we instead are comparing with an EOD signal with 1.5 in price percentage mean squared error, a number obtained from comparing our model against the most recently traded price as a proxy for an EOD signal, then we obtain the following:

$$\Delta r_s = \frac{\sqrt{1.5/10,000 + 0.0035^2} - \sqrt{0.812/10,000 + 0.0035^2}}{\sqrt{0.812/10,000 + 0.0035^2}} \tag{31}$$

$$= 24\% \tag{32}$$

Figure 2: Hypothetical chance-to-fill (CTF) curves for different RFQs



8 Chance to Fill Introduction

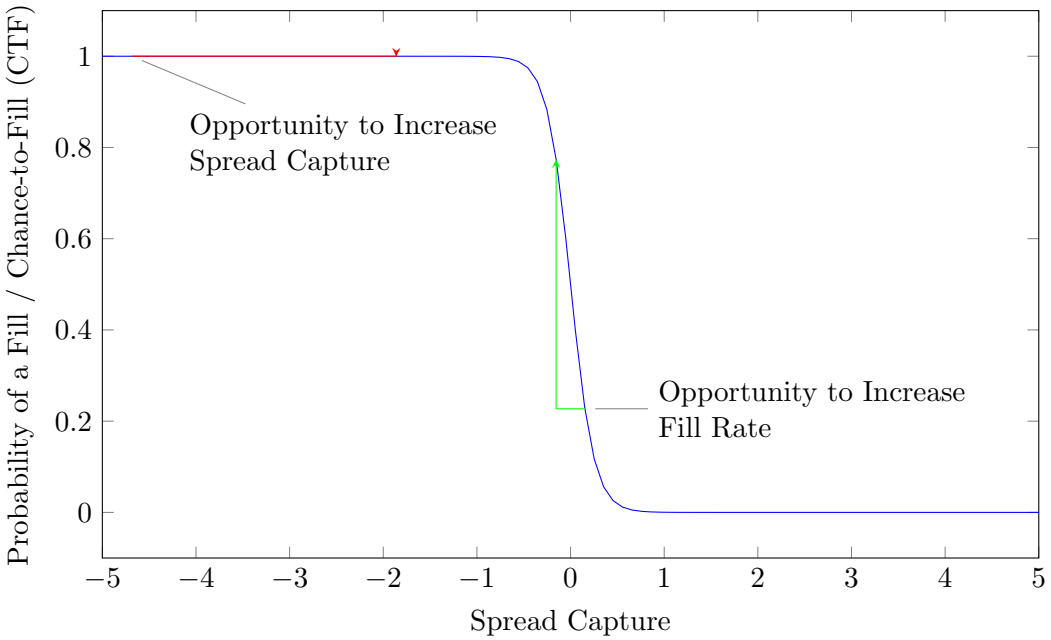
8.1 Finding Opportunities to Improve Quotes

As shown in figure 2, there will be a different CTF curve generated by the CTF model for each RFQ, and the shape of that curve will vary widely according to the market for the particular bond, what is known about the client, the number of dealers participating in the RFQ auction, etc. The shape of the curve will affect when a quote should be adjusted up or down in order to trade off cover-expense and trading volume.

Figure 3 shows a hypothetical RFQ's CTF curve, and shows how two different quotes could be improved. The one on the top right is a situation where the dealer is buying, and their base quote is high enough that the CTF curve is very flat at that quote, which means that their cover expense on this RFQ could be improved quite a bit by adjusting the quote lower, allowing a lower cover expense while at the same time decreasing the probability of a fill only slightly. This is shown by the red arrow. The green arrow shows an improvement in the quote that could be made in the other direction. Since the base quote from the dealer was at a point which is very steep on the CTF curve, there is an opportunity to sacrifice a slight increase in cover expense to gain a large increase fill rate, which if done in aggregate for many RFQs with base quotes in similar situations, would result in large trading volume at a slight cost in cover expense. We propose that the quotes are adjusted in this way depending on the situation for the RFQ such that in aggregate, the dealer will have both improved cover expense and trading volume overall.

By improving the way opportunities to adjust quotes are identified using machine

Figure 3: Hypothetical chance-to-fill (CTF) Curve for a single RFQ



learning, it is our goal to improve the overall aggregate trade-off curve between cover expense and trading volume, as illustrated in figure 4, by going from the solid blue curve for v1.0 to the v2.0 curve shown as a dotted line curve. How much of an improvement can be made is unknown until Deep MM has a chance to access the associated RFQ and market data, train the CTF model, and run back-test simulations. On this graph we show a hypothetical operating point A for a dealer, and how they could move to point B were the dealer to choose only to improve their cover expense. However, if they would like to improve both metrics, they could sacrifice some of the improvement in cover expense in order to also receive an increase in trading volume by moving to point C.

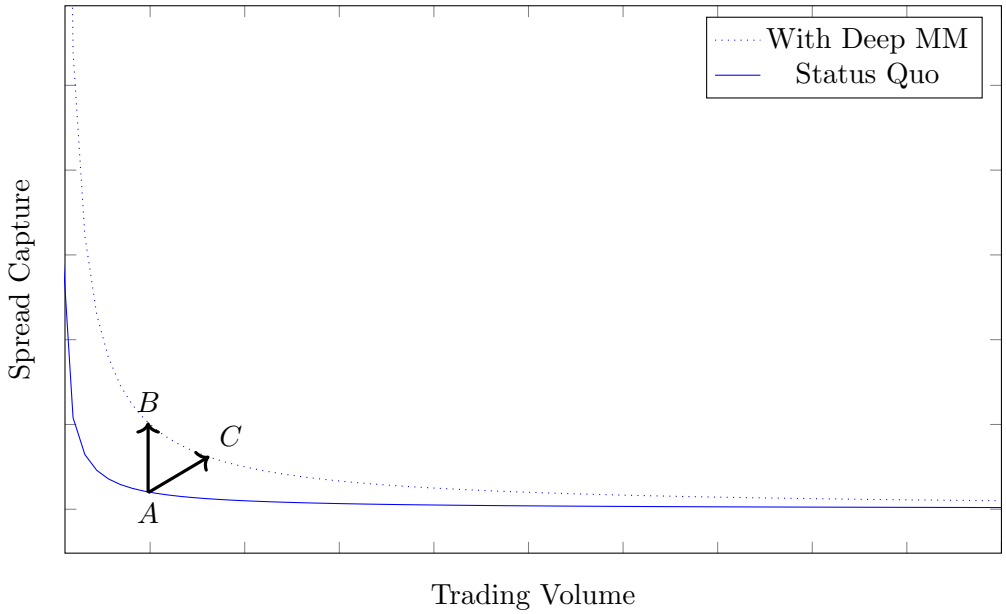
8.2 The Chance to Fill Definition

$$c(a, s_i) = Pr[Y_i^a = 1|s_i] = E[Y_i^a|s_i] \tag{33}$$

Y_i is the random variable representing whether the client traded r_i. $Y_i = 1$ represents the outcome that the client trades r_i, and $Y_i = 0$ is the outcome that the client did not trade r_i. Y_i^a represents the *counterfactual* outcome that the client *would have* filled r_i had the reference dealer quoted a instead of q_i, the quote which was sent to the client as recorded in the reference dealer's log.

For the purposes of this paper the reference dealer is the institution for which we have visibility into its historical quotes and the respective information received from the OTC platforms given the final state of the trade. For example, if you use data collected

Figure 4: Conceptual goal for volume / cover expense curve improvement



by bank ABC in the course of their market making fixed income business to calibrate a CTF model, then bank ABC is the reference dealer.

s_i is an observation of signals relative to the state of the world related to the value of the bond trade at the time the dealer issues the quote a, which includes:

- The properties of r_i
- The dealer's existing inventory
- The reference dealer's observed state of the client
- The reference dealer's observed state of the bond market

8.3 CTF Model Related Work

The authors are aware of multiple implementations at large banks which take a simplified approach to modeling CTF that often ignores the biases introduced by many scenarios where the outcome is unknown by the reference dealer. For example, in the U.S. Treasuries market, the reference dealer doesn't know the winning price for RFQs which were not by the reference dealer. Therefore, it is unknown at what price the reference dealer *would have* won the RFQ, making accurate back-testing and optimization of trading strategies difficult. Furthermore, if a model is trained only on the cases where the outcome is known, then there are biases introduced by not adjusting for common confounding causal factors of both the dealer's quote and the quotes of the other dealers

as well as the client's final decision. A solution is to model the causal underlying variables such as the best competing dealer's price and the client's reserve price using applied missing data analysis [8] techniques such as maximum likelihood estimation (MLE) [9].

9 FMV Model Design

The model we have produced is a deep neural network trained using a variant of stochastic gradient descent called adadelta using Keras & Tensorflow. Different deep neural network architectures can be useful for OTC time series data. There are papers on the use of LSTMs, and other useful architectures include transformers, and convolutional neural networks. There are model design, training, and implementation details tailored for OTC financial time series data that we are keeping as trade secrets. Much hard work went to building the data generator that has to efficiently gather all of the related transactions for each randomly generated minibatch, which required code optimizations in Cython as well as high-level computer science algorithmic design. Even with these efficiencies and training with a sophisticated GPU, the model takes over 1.5 weeks to converge and has approximately 700,000 parameters. The regression model's loss function is mean absolute error, which is shown in the literature to find an optimal solution wherein the model produces the median prediction rather than the mean/expected value, as is the case with mean squared error.

10 CTF Model Design Proposal

Whereas the FMV model has already been implemented for U.S. Corporate Bonds, here we propose a CTF model which extends the functionality of the FMV into a CTF model useful for market makers to optimize their objectives as we will discuss formally in this paper.

We take a similar approach to [9], except instead of calibrating constant parameters for the distributions of the underlying causal variables (such as the best competing dealer's price and the client's reserve price), we use machine learning to map between s_i and the parameters of the distribution. Rather than relying on static distributions of our underlying variables, we allow the distributions to be dynamically adjusted according to the changes in the reference dealer's observations of the client's state, the market state, and the properties of the RFQ.

A side benefit of using machine learning is that such model's universal function approximation ability allows the easy addition of new features or signals the reference dealer believes will improve the accuracy of the CTF model, often by simply appending the new signals to the input feature vector and retraining the model. Care must be taken when adding features to respect Independent and Identically Distributed (IID) assumptions of most function approximation techniques regarding the data samples. Features may be normalized in various ways to make the data more compatible with such IID assumptions. For example, the authors have found it helpful to normalize the price levels of various bonds to make time series data over time effectively stationary

when subsets of the time series are sampled from different time periods within the overall data set.

11 CTF Use Cases

There are two main use cases for the CTF model by a reference dealer:

1. Pricing the quote
2. Modeling the state transitions for back-testing

11.1 Pricing the Quote

For automated market making, as we discuss in the next section, if the reference dealer is able to define an objective function, that when optimized the dealer is confident their short and long-term objectives are maximized, a CTF model can easily derive an optimal pricing policy by optimizing the objective function we will show later in equation and in our discussion. If we optimize in terms of the quoted price the product of the CTF and the objective function, then we have an optimal policy. Note: the authors propose taking this work further and implementing a full reinforcement-learning solution, but that research is beyond the scope of this proposal.

The CTF can also be used as a reference signal by voice traders, who gain such benefits as being better informed about the pricing level they need to maintain their quota or having a certain probability of hedging an existing position with the trade.

11.2 Modeling the State Transitions for Back-Testing

In order to back-test an automated market making system, the reference dealer typically must test a *different* policy than the one which was followed historically. This means again that there must be a model of the CTF in order to sample from that probability to generate a series of simulated realized trades. If this CTF model is inaccurate or has systemic biases, than the resulting metrics generated by the back-test will also be inaccurate and/or biased.

11.3 Selecting the Optimal Quote using a CTF Model

We define $Q(a, s_i)$ as the historical long-term profit if at state s_i the dealer quotes a and the rest of the trades stay the same as was recorded in the historical log. $Q_\pi(a, s_i)$ is the same definition except that quoting policy π is followed for all subsequent RFQs. Those readers who are familiar with the reinforcement learning literature will recognize the Q notation. For further reading on this broader subject see [35]. Here we use Q to represent an assumed objective function which represents the reference dealer's expected utility given the price quoted a and s_i. Such an objective function will usually combine terms which reward and are numerical proxies for various typical objectives of a dealer. For market makers, such an objective function is often written in terms of:

- Short term P&L/initial margin
- Various types of risk such as net risk and gross risk
- Market share

If the historical Q is known, then the optimal quote assuming the historical policy is unchanged a_i^h is:

$$a_i^h = argmax_a Q(a, s_i) \tag{34}$$

However, although as stated a_i^h is the optimal quote for RFQ r_i assuming the historical actions are followed in the future, this is not the optimal *policy* π^*, because we would need to produce the optimal quote for each $r_i \in R$, the set of all RFQs in a given trading episode/window, under the assumption that π^* is followed for subsequent RFQs.

A policy is normally defined as a probability density function $\pi(a|s)$, wherein for continuous action spaces:

$$\int_{a=min(A)}^{a=max(A)} \pi(a|s)da = 1 \tag{35}$$

Whereas for discrete action spaces:

$$\sum_{a \in A} \pi(a|s) = 1 \tag{36}$$

Whether to assume a discrete or continuous quoting system is a design choice which will be discussed later.

For some reinforcement learning environments, a deterministic policy is optimal. For deterministic optimal policy π^*, there is an optimal action a_i^* associated with each s_i.

$$\pi^*(a = a_i^*|s_i) = 1 \tag{37}$$
$$\pi^*(a \neq a_i^*|s_i) = 0 \tag{38}$$

Although there are some environments where a stochastic policy is optimal, and therefore there is not a unique optimal a_i^* defined for each s_i, we make the simplifying assumption for our analysis that the market making problem's optimal policy is deterministic. We therefore assume that a_i^* is uniquely optimal, and thus:

$$a_i^* = argmax_a Q^*(a, s_i) \tag{39}$$

$Q^*(a, s_i)$ is the long term risk-adjusted profit if from state s_i action a is followed and the optimal policy π^* is followed for the rest of the trading episode. This optimal Q^* function is defined by the bellman equation:

$$Q^*(a, s) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s') \tag{40}$$

S is the set of all possible states. $V^*(s')$ is the value function of the optimal policy which represents the expected value over a of the Q^* function in the subsequent state s' and is defined as:

$$V^*(s') = E_{a\sim\pi^*}[Q^s(a|s')] = \sum_{a\in A} \pi^*(a|s')Q^*(a,s') \tag{41}$$

$P^a_{ss'}$ is the state transition probability. It is the probability that the state will transition from s to s' after taking action a.

$\gamma \in [0,1]$ is a discount factor that specifies how much to discount future rewards compared to present rewards. For a financial problem such as this one, it would make sense to use the real return rate of a risk-free asset (accounting for the expected level of inflation). Since real returns are negative as of late, this would not be well defined in the reinforcement learning literature, so a reasonable alternative is to set γ to 1.0 and only use this bellman equation for episodic settings where the total reward is bounded.

Depending on the type of problem, there are different strategies for using the bellman equation to solve for Q^* and thus the optimal policy π^*. For problems with a state size under 1 million or so, it is reasonable to take a tabular approach such as dynamic programming or value iteration. For problems such as the OTC market where the set of states S is practically infinite, one might attempt to use some of the popular techniques in model-free reinforcement learning such as Q learning [26], Proximal Policy Optimation (PPO) [29], Trust Region Policy Optimization (TRPO) [28], Asynchronous Advantage Actor-Critic (A3C) [25], Deep Deterministic Policy Gradient [20], Rainbow [14], etc. However, these techniques rely on a simulation of $P^a_{ss'}$, the state transition probability density function conditional on s and a. Many of the applications where reinforcement learning has been successful have perfect or near perfect simulations for the problem. For example, some early successes for Deep Q Networks (DQN) were Atari games. Since the game is cheap to run in an emulation environment on a computer, it was an easy simulation to create. Similarly, for board games like chess and go where the state transitions are known and deterministic, a perfect simulation of the game can be created and an agent can learn a super-human strategy for self-play.

However, for trading episodes wherein we only have available historical data, the state transition probability density function is unknown, but could be approximated by modeling the data generation process of every element of the state as defined above. If we had such a data generation process model, it would effectively be a simulation environment in which we could train a model-free reinforcement learning agent such as those listed above, or alternatively we could use the data generation model to perform planning in a model-based reinforcement learning algorithm, or even a hybrid approach similar to AlphaZero [33]. It goes without saying that creating a realistic simulation of the market including the different price paths is a complex problem, which is beyond the scope of this paper, but even if we don't have $P^a_{ss'}$, we can still approximate the optimal policy by approximating Q^* as $\hat{Q}$, which is a direct model of the value of different actions that come from an understanding of the problem at hand. This approximation is estimated with respect to a given policy for both managing the bond book and the hedging

book. For example, $\hat{Q}$ could be decomposed as follows (where $V(s)$ is an approximation of the relative profitability of the state s):

$$\hat{Q}(a, s_i) = c(a, s_i)V(s_i^{Y^a=1}) + [1 - c(a, s_i)]V(s_i^{Y^a=0}) \tag{42}$$
$$= c(a, s_i)[V(s_i^{Y^a=1}) - V(s_i)] + V(s_i) \tag{43}$$

$s_i^{Y^a=x}$ is defined as the state starting at s_i and transitioned to the new state portfolio should the outcome $Y_i^a = x$ (the RFQ is filled), given the quote a, where $x \in \{0, 1\}$. It is assumed $s_i^{Y^a=0} = s_i$, because if the client does not fill the quote, there is no change to the dealer's state and no change to the client's state that wouldn't have happened had the client filled the quote (excluding the few cases where the client cancels because none of the dealers quotes are sufficient), and again we assume the market state does not significantly change beyond what would have happened had the client filled the dealer's quote instead of another dealer's (again assuming away the effect of cancellations).

The approximate optimal action is defined similarly to 39 as:

$$\hat{a}_i = argmax_a \hat{Q}(a, s_i) = argmax_a c(a, s_i)[V(s_i^{Y^a=1}) - V(s_i)] + V(s_i) \tag{44}$$
$$= argmax_a c(a, s_i)[V(s_i^{Y^a=1}) - V(s_i)] \tag{45}$$

The simplification in the above equation comes because the $V(s_i)$ is constant with respect to the quote a. Let us simplify this equation further by defining:

$$\delta V(s, a) = V(s^{Y^a=1}) - V(s) \tag{46}$$

So therefore:

$$\hat{a}_i = argmax_a c(a, s_i)\delta V(s_i, a) \tag{47}$$

In other words, if we have a good chance to fill model and an objective function, we can derive the optimal action, the RFQ quote price, using equation 47. Therefore, we have shown that having a chance to fill model is an important component for deriving the OTC market making policy of a dealer.

In the next section we have a formal proof that the chance to fill and the objective function can encapsulate everything that is knowable by the dealer pre-trade towards optimizing the quote, and then we prove that improving the accuracy of the estimate of the chance to fill probability will improve the profitability (as defined by the objective function).

11.4 Optimality of the Objective Function

We now prove that all pre-trade knowable information regarding the chance to fill for a given RFQ auction and the future commercial value to the company of getting filled at a quote level a can be encapsulated into $c(a)$ or $\delta V(a)$, and therefore $a = \arg\max_a c(a) \cdot \delta V(a)$ represents the optimal quote.

11.4.1 Setup and Key Concepts

Consider a single RFQ auction. The dealer must choose a quote level a from a set of possible quotes (discrete or continuous). The outcome is binary: the trade is either filled (with probability $c(a)$) or not (with probability $1 - c(a)$).

The CTF $c(a)$ encapsulates all pre-trade knowable factors affecting the probability of fill, such as client reserve prices, competing dealer behavior, market conditions, RFQ characteristics (e.g., size, bond liquidity), and historical data patterns. Any knowable pre-trade information that influences whether the quote wins the auction is reflected in $c(a)$, as it is the conditional probability given all observable features.

The objective $\delta V(a)$ represents the expected net present value to the company if the trade is filled at a. This includes immediate spread capture, expected post-trade carry (from alpha signals), risk adjustments, client relationship value, market share bonuses, inventory effects, and any other future commercial impacts (e.g., hedging costs, regulatory implications). All pre-trade knowable factors affecting the value conditional on fill are incorporated into $\delta V(a)$.

Assume the dealer is risk-neutral or that risk is already priced into $\delta V(a)$ (e.g., via utility adjustments). The expected profit for quoting at a is $c(a) \cdot \delta V(a) + (1 - c(a)) \cdot 0 = c(a) \cdot \delta V(a)$, assuming no value if unfilled.

11.4.2 Intuition

Any pre-trade information must affect either the likelihood of fill (into $c(a)$) or the value if filled (into $\delta V(a)$). The product $c(a) \cdot \delta V(a)$ is the expected value, and maximizing it yields the optimal quote under standard decision theory.

11.4.3 Formal Proof

Let $\mathcal{I}$ be the set of all pre-trade knowable information. The true expected profit is $\mathbb{E}[\pi \mid \mathcal{I}, a] = \Pr(\text{fill} \mid \mathcal{I}, a) \cdot \mathbb{E}[\pi \mid \text{fill}, \mathcal{I}, a]$.

By definition, $c(a) = \Pr(\text{fill} \mid \mathcal{I}, a)$, so it encapsulates all $\mathcal{I}$ affecting fill probability.

Similarly, $\delta V(a) = \mathbb{E}[\pi \mid \text{fill}, \mathcal{I}, a]$, encapsulating all $\mathcal{I}$ affecting value if filled.

Thus, $\mathbb{E}[\pi \mid \mathcal{I}, a] = c(a) \cdot \Delta V(a)$.

The optimal quote is

$$a^* = \arg\max_a \mathbb{E}[\pi \mid \mathcal{I}, a] = \arg\max_a c(a) \cdot \Delta V(a),$$

as it maximizes expected profit over all possible a.

If any factor in $\mathcal{I}$ is omitted, it would bias $c(a)$ or $\delta V(a)$, leading to suboptimal a^*. Including all $\mathcal{I}$ ensures optimality.

11.4.4 Concrete Example

Suppose a client rating affects both fill probability (higher-rated clients more likely to fill) and value (repeat business). The rating adjusts $c(a)$ upward for likely fills and $\delta V(a)$ for long-term value, fully captured in the product.

11.5 Proof that Improving FMV Accuracy Improves Dealer Profitability

We now provide a mathematical proof that improving the accuracy of the FMV model's probability estimates, used as a proxy for the CTF, leads to higher expected profitability for a dealer, assuming the dealer's objective function remains unchanged. We divide this into the setup, intuition, formal proof for the continuous case, and a concrete numerical example.

11.5.1 Setup and Key Concepts

The CTF, denoted as $c(a)$, is the probability that a dealer's quote at a quote level a (the offered spread, in % of par) will result in a filled trade. This is influenced by client reserve prices and competing dealer quotes. It is typically a decreasing function: wider quotes make fills less likely.

The FMV model predicts the probability distribution of future TRACE prints at different levels. Dealers use these FMV probabilities as a proxy for the true CTF $c(a)$. We call the estimated CTF $\hat{c}(a)$. Accuracy means how close $\hat{c}(a)$ is to the true $c(a)$ across possible quotes a, measured for example by the maximum difference $\sup_a |\hat{c}(a) - c(a)|$.

The dealer's objective $\delta V(a)$ represents the value of filling a trade at quote a. A simple example is $\delta V(a) = a$ for spread capture. This is increasing: wider quotes are more profitable if filled. The full expected profit for a quote at a is $c(a) \cdot \delta V(a)$.

The dealer selects the optimal quote a^* that maximizes expected profit:

$$a^* = \arg\max_a \, c(a) \cdot \delta V(a)$$

The true maximum profit is $\pi^* = c(a^*) \cdot \delta V(a^*)$. With an estimated CTF, they choose $\hat{a} = \arg\max_a \, \hat{c}(a) \cdot \delta V(a)$, and the actual profit is $c(\hat{a}) \cdot \delta V(\hat{a})$.

11.5.2 Intuition

The true best quote a^* balances the trade-off: too tight (a small), and fills are frequent but with low profit per fill; too wide (a large), and fills are rare but with high profit if they occur. If the FMV estimate $\hat{c}(a)$ is inaccurate (e.g., overestimates fill probability at wide quotes), the dealer picks a suboptimal $\hat{a}$, leading to lower true profit. Improving accuracy makes $\hat{c}(a)$ match $c(a)$ better, so $\hat{a}$ gets closer to a^*, and profit approaches the maximum possible.

11.5.3 Formal Proof (Continuous Case)

Assume quotes a are continuous, and functions are smooth.

Define $f(a) = c(a) \cdot \delta V(a)$ as the true expected profit function, and $\hat{f}(a) = \hat{c}(a) \cdot \delta V(a)$ as the estimated one. Assumptions: - $c(a)$ is decreasing and continuously differentiable. - $\delta V(a)$ is increasing and continuously differentiable. - $f(a)$ is concave (has a unique maximum). - As accuracy improves, $\hat{c}(a) \to c(a)$ uniformly.

Since $f(a)$ is concave, there is a unique a^* where $f'(a^*) = 0$ and $f(a) \leq f(a^*)$ for all a.

With $\hat{c}(a)$, the dealer maximizes $\hat{f}(a)$, getting $\hat{a}$ where $\hat{f}'(\hat{a}) = 0$. The true profit is $f(\hat{a}) = c(\hat{a}) \cdot \delta V(\hat{a})$. By concavity of f, if $\hat{a} \neq a^*$, then $f(\hat{a}) < f(a^*)$.

As accuracy improves, $\hat{c}(a) \to c(a)$ uniformly, so $\hat{f}(a) \to f(a)$ uniformly. The maximum of $\hat{f}$ converges to the maximum of f:

$$\max_a \hat{f}(a) \to \max_a f(a) = \pi^*$$

The argmax $\hat{a} \to a^*$. Thus, the achieved true profit $f(\hat{a}) \to \pi^*$, and for finite improvements, profit is non-decreasing.

For small errors, profit loss is approximately quadratic in error size (from Taylor expansion around a^*):

$$f(\hat{a}) \approx f(a^*) + \frac{1}{2} f''(a^*)(\hat{a} - a^*)^2$$

Since $f''(a^*) < 0$, smaller errors mean smaller loss, higher profit.

11.5.4 Concrete Numerical Example

Use a logistic CTF:

$$c(a) = \frac{1}{1 + e^{k(a-m)}}$$

with $k = 100$, true midpoint $m = 0.01$. Assume $\Delta V(a) = a$.

The true optimum is $a^* \approx 0.01567$, profit ≈ 0.00567 (% of par).

Suppose FMV error shifts the estimated midpoint to $\hat{m} = m + e$ (e.g., $e = \pm 0.001$ to ± 0.005).

For each e, find $\hat{a}$ maximizing $\hat{c}(a) \cdot a$, then true profit $c(\hat{a}) \cdot \hat{a}$, and % loss $= \frac{\pi^* - \text{true profit}}{\pi^*} \times 100$.

As $|e|$ decreases, % loss drops toward 0, confirming profitability improves.

11.6 Accuracy in Mid-Level vs. CTF Function Function

We now distinguish between the impact of improving the accuracy of the mid-level (the average of bid and offer, effectively shifting the reference point for quotes) within $\delta V(a)$, and improving the accuracy of the CTF function $c(a)$ across all quote levels a. We provide a formal proof that while mid-level improvements enhance results on average, they are not guaranteed to do so in every case, whereas uniform improvements in CTF accuracy are guaranteed to improve or at least not worsen expected profitability.

11.6.1 Setup and Distinction

The quote level a is typically measured relative to an estimated fair market value or mid-level (denoted here as the model's perceived "mid"). Improving mid-level accuracy means better estimating the true center of the price distribution, which shifts the entire quoting grid. This primarily affects $\delta V(a)$, as the value of a fill at a given relative spread

Error e	Optimal Quote $\hat{a}$	True Profit	% Loss vs. Optimal
-0.005	0.01405	0.00562	0.88%
-0.004	0.01434	0.00564	0.59%
-0.003	0.01465	0.00565	0.34%
-0.002	0.01498	0.00566	0.16%
-0.001	0.01532	0.00567	0.04%
0.000	0.01567	0.00567	0.00%
0.001	0.01604	0.00567	0.04%
0.002	0.01642	0.00566	0.18%
0.003	0.01682	0.00565	0.41%
0.004	0.01723	0.00563	0.74%
0.005	0.01766	0.00560	1.18%

Table 1: Illustrative example of the profit loss as a function of estimation error in the logistic CTF midpoint.

depends on how well-aligned the quote is with the true fair value (e.g., via post-trade carry or hedging efficiency).

In contrast, improving CTF accuracy means reducing estimation error in $c(a)$ uniformly or across the board for all a, regardless of mid-level shifts.

11.6.2 Intuition

Improving the mid-level can enhance $\delta V(a)$ by better centering quotes around the true fair value, increasing expected carry or reducing risk per fill. However, if the functional shape of $c(a)$ is poorly estimated (e.g., wrong steepness or asymmetry), a better-centered but misshaped CTF proxy may lead to suboptimal trade-offs between fill probability and value.

Improving $c(a)$ accuracy directly enhances the dealer's ability to evaluate the probability-value trade-off at every possible quote level, leading to a better choice of $\hat{a}$.

11.6.3 Formal Proof for Mid-Level Improvement (Not Guaranteed)

To rigorously demonstrate that improving mid-level accuracy does not guarantee an increase in expected profitability, we consider the interaction between mid estimation error and structural errors in the estimated CTF $\hat{c}(a)$.

Let the true mid be m, and suppose the model estimates $\hat{m} = m + e$, where e is the mid error. Quotes a are defined relative to the estimated mid $\hat{m}$, so the absolute quote level submitted to the market is effectively shifted by e. This shift introduces a systematic bias into $\delta V(a)$, as the value conditional on fill (e.g., post-trade carry, hedging efficiency, or inventory risk) depends on alignment with the true fair value centered at m. For a fixed chosen relative quote a, reducing $|e|$ improves $\delta V(a)$, increasing profit conditional on fill.

However, the dealer optimizes over the estimated objective $\hat{f}(a) = \hat{c}(a) \cdot \delta\hat{V}(a)$, where $\delta\hat{V}(a)$ reflects the mid-biased value function. The selected quote is $\hat{a} = \arg\max_a \hat{f}(a)$, and true expected profit is $\pi = c(\hat{a}) \cdot \Delta V(\hat{a})$ (note that $\delta V(\hat{a})$ is evaluated at the absolute level corresponding to relative $\hat{a}$ plus shift e).

Now suppose $\hat{c}(a)$ contains a persistent structural error independent of the mid (e.g., incorrect steepness parameter k in the logistic form $c(a) = \frac{1}{1+e^{k(a-m)}}$). In certain regimes, the mid error e and the CTF structural error can partially offset each other, yielding a "fortuitously" reasonable $\hat{a}$ despite both errors.

Formally, let $\pi(e)$ be the true expected profit as a function of mid error e (holding the CTF error fixed). By the envelope theorem applied to the dealer's optimization problem, the marginal effect of a small change in e on achieved profit is

$$\frac{d\pi}{de} = \frac{\partial f(\hat{a}(e))}{\partial e},$$

where the total derivative accounts for both the direct effect on δV and the indirect effect through the adjusted optimal $\hat{a}(e)$. Crucially, $\frac{\partial f}{\partial e}$ can be positive or negative depending on the direction of the compensating errors. If the current e is offsetting a CTF shape bias in a way that places $\hat{a}$ closer to the true optimum, then $\frac{d\pi}{de}$ may have the opposite sign of e, meaning reducing $|e|$ (improving mid accuracy) decreases π.

A concrete counterexample arises when an underestimated CTF steepness (too flat $\hat{k} < k$) causes overly aggressive quoting, but a positive mid error $e > 0$ (systematic cheapening) counteracts this by shifting absolute quotes wider, restoring balance. Correcting only the mid error removes this accidental compensation, potentially leading to tighter quotes, lower spread capture, and reduced profitability until the CTF shape is also corrected.

Thus, while mid-level improvements are beneficial on average—especially over many heterogeneous RFQs where compensating errors average out—they are not guaranteed to increase expected profit in every individual case or model state. The improvement is conditional on the nature and interaction of concurrent errors in the CTF estimate.

11.6.4 Formal Proof for CTF Accuracy Improvement (Guaranteed)

We now rigorously establish that uniform improvement in the accuracy of the estimated CTF $\hat{c}(a)$ toward the true CTF $c(a)$ guarantees non-decreasing expected profitability, with strict improvement in generic cases.

Define the estimation error as $\epsilon(a) = \hat{c}(a) - c(a)$, and assume that accuracy improvements reduce the uniform norm $\|\epsilon\|_\infty = \sup_a |\epsilon(a)| \to 0$.

The true objective function is $f(a) = c(a) \cdot \delta V(a)$, and the estimated objective is

$$\hat{f}(a) = \hat{c}(a) \cdot \delta V(a) = f(a) + \epsilon(a) \cdot \delta V(a).$$

Since $\delta V(a)$ is bounded and continuous over the compact domain of feasible quotes a, the perturbation term $\epsilon(a) \cdot \delta V(a)$ converges uniformly to zero as $\|\epsilon\|_\infty \to 0$. Thus, $\hat{f}(a) \to f(a)$ uniformly.

Under the standard assumption that $f(a)$ is strictly concave (ensuring a unique global maximum a^*), uniform convergence of $\hat{f}$ to f implies: 1. $\max_a \hat{f}(a) \to \max_a f(a) = \pi^*$, 2. The argmax $\hat{a} = \arg\max_a \hat{f}(a)$ converges to the true optimum a^*.

The achieved true expected profit is $\pi = f(\hat{a})$. As $\hat{a} \to a^*$, continuity of f yields $f(\hat{a}) \to f(a^*) = \pi^*$.

For finite (non-infinitesimal) improvements that reduce $\|\epsilon\|_\infty$, the profit $f(\hat{a})$ is non-decreasing. This follows from the fact that the perturbation magnitude decreases monotonically, and strict concavity of f ensures that any deviation from a^* incurs a positive second-order loss (Taylor expansion around a^*): smaller perturbations yield argmax points closer to a^*, hence higher or equal true value.

This guarantee holds even if $\delta V(a)$ contains persistent mid-related or other biases. To see this mathematically, suppose $\delta V(a) = \delta V_0(a) + b(a)$, where $b(a)$ is a bias term (e.g., due to mid misalignment). The true objective becomes $f(a) = c(a) \cdot [\delta V_0(a) + b(a)]$. Improving $\hat{c}(a) \to c(a)$ still refines the probability weighting in the optimization, yielding $\hat{a}$ that maximizes a closer approximation to the biased $f(a)$. By the same convergence argument, $f(\hat{a})$ approaches the maximum achievable under the biased δV, which is at least as good as (and typically better than) the value from the less accurate $\hat{c}$. Thus, the optimizer exploits the available value more effectively, regardless of distortions in δV.

Furthermore, in practice, the mid-level (reference fair value) is often implicitly encoded within the CTF function itself. For example, in symmetric bid/offer settings, the mid can be identified as the point m where $c(a) = c(-a)$ for all a (symmetry), or where the derivative $c'(0)$ indicates the sensitivity around the center. More formally, if paired bid and offer CTF curves are available, the mid m can be recovered by solving for the shift that aligns $c_{\text{bid}}(a + m) = 1 - c_{\text{offer}}(-a + m)$ or similar equilibrium conditions. Consequently, broad improvements in CTF accuracy—by better capturing the shape and location of $c(a)$—simultaneously enhance mid estimation without requiring separate modeling. This provides a compounded robustness that isolated mid refinements cannot match, as errors in CTF shape could otherwise undermine even a perfectly estimated mid.

Thus, enhancing CTF accuracy offers a theoretically guaranteed path to superior quoting performance.1.2s

11.6.5 Concrete Numerical Example

Consider the logistic $c(a) = \frac{1}{1+e^{100(a-0.01)}}$, $\delta V(a) = a + \alpha\delta_p$ (with mid bias in δ_p, pre-trade carry signal).

If mid error $e = 0.002$ offsets a CTF slope error (e.g., $\hat{k} = 90$), correcting mid to $e = 0$ might reduce profit by 0.1-0.5% (simulation-dependent). Correcting slope to $\hat{k} = 100$ always increases profit toward optimum.

11.6.6 Conclusion

While improving mid-level accuracy enhances the quality of $\delta V(a)$ and is valuable—particularly for post-trade carry and risk management—it does not guarantee improved quoting de-

cisions due to potential interactions with CTF shape errors. In contrast, broad improvement in CTF accuracy $c(a)$ is guaranteed to improve (or at least not worsen) expected profitability, as it directly refines the core probability estimation driving the argmax optimization.

11.7 Reference Dealer's Objective Function

δV is typically called the objective function, some of the components of which could be, according to the reference dealer's business requirements:

- Expected costs
  - Hedge costs
  - Fees
  - Repo/coupon rates
  - Expected toxicity/slippage
  - Opportunity cost of capital
- Risk
  - Carry risk
  - Risk delta (risk increasing or risk reducing trade)
  - Liquidity. If the bond is less liquid, it is generally more risky to carry the bond long or short.
- Initial P&L
  - Initial Margin
  - Spread capture
  - Post-Trade Carry via Alpha Signal – expected inventory P&L days after execution. (See section 11.10)
- Long term effects
  - Bonus for completed trades to incentivize market share
- Macro assumptions
  - Long short bias
  - Future credit curve assumptions

Ideally, as far as possible we should create terms in the δV that have a theoretical basis for being comparable on the same scale and therefore won't need to be tuned. By doing this we reduce the attack surfaces for over-fitting, which is crucial as we have very little data with few market regimes represented, and if we over-fit our objective function

parameters we could, for example, increase our real carry risk dramatically. There are some values such as the subsidies for client hit rates that will need to be tuned in a multi-objective fashion as dealers are typically unable to define the true P&L to hit rate trade-off they want to hit forever. This is typically a moving target that needs to be responsive to client requests. Having defined this expected form of our objective function δV, which ideally only contains expectations of sources of P&L known at the time of the trade we can use equation 47 to price in a way that approximates the *expected* optimal profit under the assumptions we have made thus far.

11.8 Simple Formal Objective Function

As an example, we can explore a simple formulation of the objective function.

The objective function in this application quantifies the value of a trade to the desk *if it is filled*, and is represented as:

$$\delta V(a, b, p_{FMV}, l) = \gamma(\chi - 3) + \mu + b(p_{FMV} - a) - \frac{nv}{\eta l} + \zeta b + o \tag{48}$$

This function incorporates several adjustable parameters and other variables, such as client rating, market share bonus, spread capture, risk, long/short bias, and further adjustments. By changing these parameters, the trader can influence the quote recommendations and overall strategy for the trade. The units of the overall objective function is in terms of percent of the bond's par value, for which we will use the $ as a shorthand for this unit in the analysis below; in other words the same units in which the price of the bond is normally expressed. This has the advantage that the spread capture term, for example, doesn't need to be converted, assuming that the spread is expressed in dollar terms.

- χ is a five star rating given to a client of the reference dealer. 3 is considered neutral, whereas 1 is the minimum score and 5 is the maximum rating. This value is saved between RFQs and reused, but the trader / desk can always change it.
- γ is a non-negative trader or trading desk parameter that converts the star rating (after subtracting 3, which is the neutral rating) into the percent of bond par value units for all clients. So the unit of γ is $\frac{\$}{star}$.
- μ is the market share bonus for filling the trade, in terms of $.
- b is the buy / sell indicator. It is 1 if trade is one in which the reference dealer is buying, and -1 if the dealers is selling.
- $b(p_{FMV} - a)$ is the spread capture, in $.
  - p_FMV is the midpoint price as reported by the FMV model. (The mean of the expected buy and sell prices given the quantity and other properties of the RFQ).
  - As mentioned above, a is the reference dealer's quote. (As in equation 47)

- n is the change in net duration / DV01 risk incurred against the central risk book of the reference dealer for IG traders, or OAS spread for HY traders, or according to the user's preference. This net change can be positive or negative. Since DV01 is in terms of a percentage of the bond par value it is effectively in $ units. If n is OAS Spread, then the unit is basis points.

- v is the volatility metric, often some form of price standard deviation in the recent price history, and thus basically in $ units. See section 11.8.2 for more information about the possible composition of this metric.

- l is an estimate of the liquidity of the bond being traded and is an output by our liquidity model. It is inverse of the expected amount of time ($\frac{1}{time}$) before the same volume will post for the bond of the trade (on the opposite side as the trade in question). So in other words, l is more liquid the larger the value is (because if the time expected before the volume of the trade is posted on the opposite side is smaller it means there is a lot of expected volume coming).

- η is a non-negative trader or trading desk parameter which the risk metrics term $\frac{nv}{l}$ is multiplied by to get the $ total contribution of risk, should the trade fill. It is in terms of $\$ \times time$ so that the units cancel with that of $\frac{v}{l}$ and we're left with a risk term in DV01 or effectively $. Alternatively, if n is OAS spread, then the units of η are $bps \times time$, cancelling out the units of the other terms in the risk component except the $ unit from v. When η is smaller, this causes quote recommendations which are lower risk because the overall risk term is larger in magnitude (but negative).

- ζ is the long / short bias in terms of $. If ζ is positive then there is a long bias; if it is negative there is a short bias.

- o is a positive or negative user parameter which can be used to make further adjustments in the positivity or negativity of the trade in terms of $ if the trade fills.

11.8.1 User Parameters of the Objective Function

The Greek letters χ, γ, μ, η, ζ, o are all constants in the objective function which are controlled either by the individual trader or the overall trading desk. Every time one of these constants has changed, δV becomes a new objective function which prioritizes the terms differently. For example, the trading desk over time can decide the favorability of individual clients as χ stored in the system over time, while γ, μ, η, ζ might be selected by the individual trader or overall trading desk at the beginning of each trading day, and the system will remember the values until they are changed. o is selected for every trade where the trader would like to make a further adjustment to their quote. The English letters are all properties of the RFQ or output of models.

11.8.2 Volatility Measures

In the objective function example in equation 48, the v term is a volatility metric for the bond in question. Since it is often measured in terms of the standard deviation of the price movement, we can assume it is in dollar terms.

Market makers consider recent price volatility because it can signal changing market conditions, which may affect the liquidity and risk of the bond. High price volatility generally leads to wider bid-offer spreads as market makers attempt to mitigate their risk exposure.

To measure recent price volatility, market makers often use the following methods:

1. Historical Volatility: This measure is based on the standard deviation of the bond's price changes over a specified period (e.g., 30 days, 60 days, etc.). Higher historical volatility indicates greater price fluctuations and higher risk. Market makers monitor historical volatility trends to assess how the bond's risk and liquidity have evolved over time.

2. Implied Volatility: This measure is derived from the prices of options on bonds or bond indices. Implied volatility represents the market's expectation of future price volatility. Market makers use implied volatility to gauge the market's sentiment and adjust their bid-offer spreads accordingly.

3. Realized Volatility: This measure captures the actual price movements of a bond during a specific time frame. Realized volatility can be useful for market makers to understand how a bond's price has reacted to recent market events, such as economic data releases or geopolitical news.

4. Market Indicators: Market makers may also consider broader market indicators, such as the CBOE Volatility Index (VIX), which measures the implied volatility of S&P 500 index options. While the VIX primarily focuses on equity markets, it can still provide useful insights into overall market sentiment and risk aversion, which can impact bond market volatility as well.

By incorporating recent price volatility into their bid-offer spreads, market makers can better manage their risk exposure and provide more accurate pricing for bonds. This, in turn, helps ensure a more efficient and liquid bond market.

11.9 Objective Function User Interface (UI)

The user will interact with our CTF UI which displays the current chance to fill curve as produced by the CTF model, along with our current recommended quote derived from equation 47. The trader will have sliders which will allow them to adjust the parameters to the objective function defined above in section 11.8. Below find descriptions which explain how the UI will work for each slider.

11.9.1 χ Client Rating Slider

The client rating slider, χ, represents the favorability of a client, with a range of 1-5 stars (1 being the minimum score and 5 being the maximum rating). The default value is 3, which is considered neutral. The unit for this slider is "stars", and it affects the objective function by influencing the $\gamma(\chi - 3)$ term. Note – this slider is only meant to express the trader's or desk's client preferences, and can be adjusted everytime you have an interaction with a client that might affect your rating of this client. This value is saved between trades and is associated with this client's identifier.

11.9.2 γ Client Conversion Slider

The γ slider allows the trader to adjust the conversion factor from star rating to percentage of bond par value for all clients. This parameter is non-negative and has units of $\frac{\$}{star}$. The slider will be named "Client Conversion" and the unit of the slider will be "$\frac{\$}{star}$". The slider will move horizontally and the allowed range will be from 0 to a maximum value determined by the user or trading desk (starting value of \$1). Note – this is a slider that is meant to express the overall client effect on pricing, not for this particular bond / trade. A bonus or penalty for this particular trade should be put into the "Further Adjustment Slider." This slider should only be adjusted occasionally, not for every trade.

11.9.3 μ Market Share Bonus Slider

The μ slider allows the trader to adjust the market share bonus for filling the trade. The unit of the slider will be "\$". The slider will move horizontally and the allowed range will be from 0 to a maximum value determined by the trading desk (starting value of \$1). It expresses how much spread capture you are willing to sacrifice to get a trade like this to fill (which means that its effect is modulated by the CTF curve). Note – this is a slider that is meant to express the overall market share bonus, not for this particular bond / trade. A bonus or penalty for this particular trade should be put into the "Further Adjustment Slider." This slider should only be adjusted occasionally, not for every trade.

11.9.4 η Risk Slider

The η slider allows the trader to adjust the risk contribution parameter. The unit of the slider will is $\$ \times time$ if n is DV01, and $bps \times time$ if n is OAS spread. This parameter is non-negative, with a minimum value of 1 (we don't want to let it go to zero and divide by zero in our objective function), and a maximum of 90 for now. The left side is marked as low risk (meaning we go to extremes to avoid risk), and the right side is marked as high risk, meaning that risk factors in less to the decision as to whether we do the trade. Note – this is a slider that is meant to express the overall risk sensitivity of you as a trader, not for this particular bond / trade. A bonus or penalty for this

particular trade should be put into the "Further Adjustment Slider." This should only be adjusted occasionally, not for every trade.

11.9.5 ζ Long/Short Bias Slider

The ζ slider allows the trader to adjust the long/short bias of the trade. The unit of the slider is "$". The slider will move horizontally and the allowed range will be from a negative maximum value determined by the trading desk to a positive maximum value determined by the trading desk. For now we can set the minimum to -$1, and the maximum to $1. Note – this is a slider that is meant to express the overall long or short bonus, not for this particular bond / trade. A bonus or penalty for this particular trade should be put into the "Further Adjustment Slider." This slider should only be adjusted occasionally, not for every trade.

11.9.6 o Further Adjustment Slider

The o slider allows the trader to make further adjustments to the positivity or negativity of the trade in terms of $. It expresses how much spread you would want to sacrifice (if it is positive) or gain to fill this particular trade. This parameter is positive or negative and is determined by the trader for each trade. The unit of the slider is "$". The slider will move horizontally and the allowed range will be from a negative maximum value determined by the trading desk to a positive maximum value determined by the trading desk. This slider can be adjusted for every trade.

11.10 Incorporating Predictive Alpha Signals for Post-Trade Carry

Many trading organizations consolidate algorithmic and human-trader books for internal performance measurement. In such environments a sale that is immediately profitable on spread can still hurts the desk's score if the bond subsequently rallies. To give traders explicit control over this risk we extend the objective function with an alpha parameter α that incorporates a predictive signal of post-trade price change (carry).

Let Δp be the expected post-trade price movement (

$$p'_{\mathrm{FMV}} = p_{\mathrm{FMV}} + \alpha \cdot \Delta p, \tag{49}$$

where $\alpha \in [0,\ 1]$ is a new user-controlled parameter. The objective function then becomes

$$\delta V'(a, b, p'_{\mathrm{FMV}}, l) = \gamma(\chi - 3) + \mu + b(p'_{\mathrm{FMV}} - a) - \frac{n\,v}{\eta\,l} + \zeta b + o. \tag{50}$$

The optimal quote remains

$$a^* = \arg\max_a \; c(a)\,\delta V'(a, b, p'_{\mathrm{FMV}}, l), \tag{51}$$

with Chance-to-Fill modelled as the realistic sigmoid

$$c(s) = |b(p'_{\text{FMV}} - a)|) = \frac{1}{1 + \exp\big(k(s-m)\big)}, \tag{52}$$

k = 100 and m = 0.01 in the experiments below.

When $a > 0$ the system deliberately skews quoting: positive predicted Δp encourages tighter offers to capture anticipated appreciation; negative predicted Δp widens spreads to avoid unwanted inventory. In aggregate this creates a controllable trade-off between immediate spread capture and post-trade carry P&L.

11.10.1 Simulation Design

We simulate 10 000 independent RFQs with the following assumptions (all values in

- True post-trade move: $\Delta p_{\text{true}} \sim \mathcal{N}(0, 0.01)$
- Predicted move (noisy signal): $\Delta p_{\text{pred}} = \Delta p_{\text{true}} + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 0.015)$
- Sell-side direction (b = -1) to highlight adverse rally penalty
- Baseline $p_{\text{FMV}} = 100$ (w.l.o.g.)
- All other objective-function terms except the spread term set to zero for clarity

For each $\alpha \in 0.00, 0.05, \ldots, 1.00$ we numerically maximise $c(s) \cdot \delta V'$ over a fine spread grid, simulate fills according to the sigmoid CTF, and record average spread capture, post-trade carry, fill count and total P&L.

11.10.2 Results

α	Avg Spread Capture	Avg Post-Trade Carry	Filled Volume (out of 10k)	Total P&L
0.00	0.01567	-0.00012	3687	0.01555
0.05	0.01565	0.00039	3587	0.01604
0.10	0.01563	0.00034	3598	0.01597
0.15	0.01551	0.00045	3554	0.01595
0.20	0.01543	0.00066	3604	0.01609
0.25	0.01529	0.00103	3663	0.01632
0.30	0.01511	0.00130	3643	0.01641
0.35	0.01486	0.00164	3617	0.01651
0.40	0.01486	0.00135	3636	0.01620
0.45	0.01451	0.00179	3641	0.01630
0.50	0.01424	0.00170	3663	0.01594
0.55	0.01407	0.00229	3581	0.01636
0.60	0.01367	0.00241	3538	0.01608
0.65	0.01370	0.00200	3554	0.01570

0.70	0.01315	0.00246	3638	0.01560
0.75	0.01300	0.00250	3598	0.01550
0.80	0.01276	0.00254	3653	0.01530
0.85	0.01231	0.00284	3683	0.01516
0.90	0.01200	0.00284	3622	0.01485
0.95	0.01176	0.00290	3553	0.01467
1.00	0.01150	0.00305	3662	0.01455

Table 2: Simulation results (10 000 RFQs) for the alpha parameter trade-off.

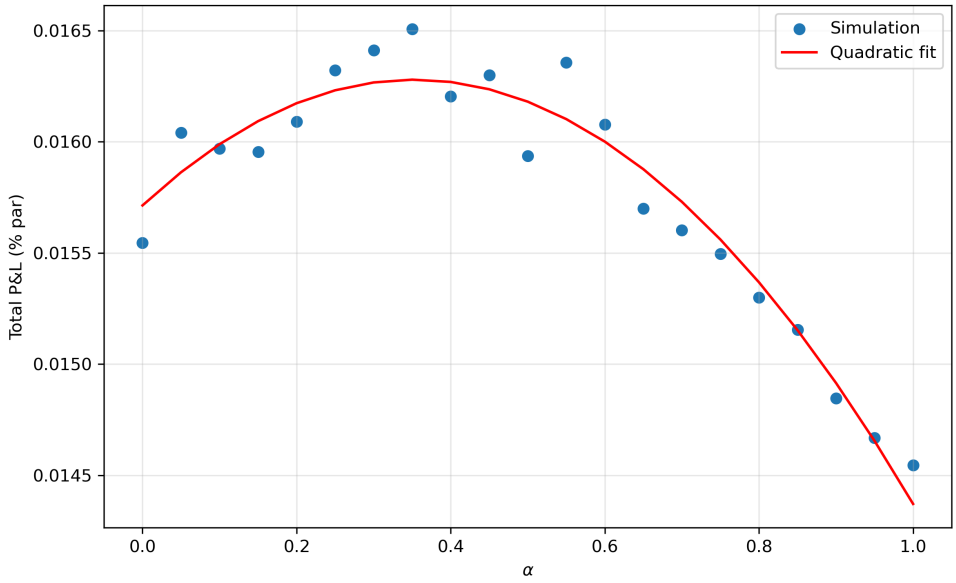


Figure 5: Total risk-adjusted P&L versus alpha parameter α. Peak performance occurs around $a \approx 0.4$, demonstrating that moderate reliance on a noisy alpha signal is optimal.

The results confirm that $a = 1.0$ is *not* optimal when the alpha signal contains realistic noise: total P&L peaks at $a \approx 0.4$ and then declines as over-aggressive reliance on the noisy forecast erodes spread capture faster than it improves carry. The new parameter therefore belongs on the objective-function UI as an additional slider (suggested name: "Carry Signal Weight").

11.11 Optimizing Objective Function Parameters with Multiobjective Optimization

The objective function described in Section 11.7 incorporates several user-controllable parameters: γ (client conversion), μ (market share bonus), η (risk aversion), ζ (long/short bias), o (further adjustment), and the carry signal weight α introduced in Section 11.10. These parameters allow traders to steer quoting behavior toward specific goals. How-

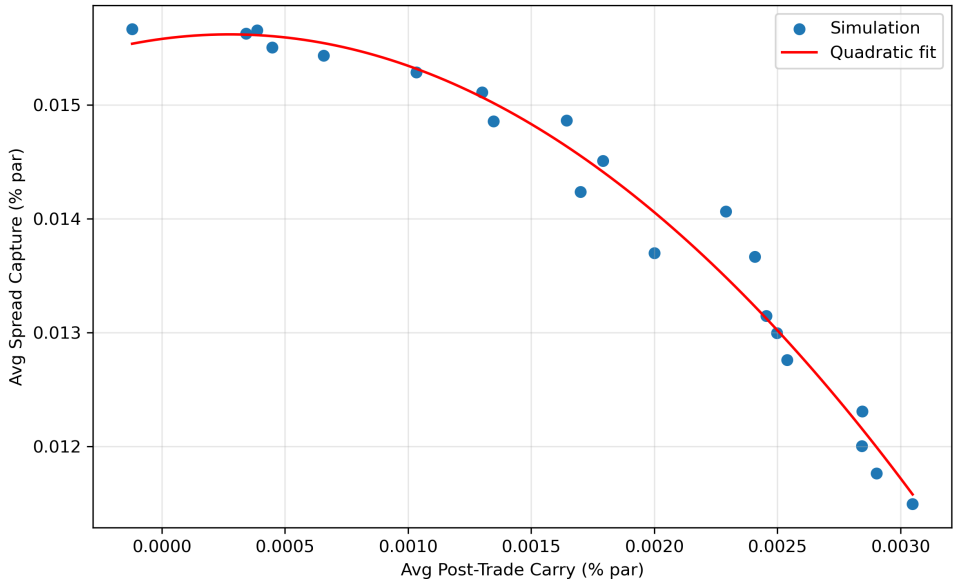


Figure 6: Spread-capture versus post-trade carry trade-off frontier (α from right to left as α increases). Data points with quadratic trend line.

ever, trading desks typically pursue multiple, often conflicting objectives simultaneously — for example:

- maximizing immediate spread capture,
- maximizing market share / fill rate,
- minimizing inventory risk (various forms),
- optimizing post-trade carry / risk-adjusted P&L.

Multiobjective optimization (MOO) provides a systematic framework for exploring these trade-offs. Instead of seeking a single "best" parameter set, MOO identifies the *Pareto front* — the set of non-dominated solutions in which no objective can be improved without degrading at least one other objective.

It is essential to distinguish between *pre-trade* signals and *long-term* objectives:

Pre-trade (known at quote time)	**Long-term (observable ex-post)**
Fair Market Value (FMV)	Risk-adjusted P&L
Chance to Fill (CTF)	Realized spread capture
Immediate spread capture	Inventory volatility / VaR
Client rating, liquidity, volatility	Market share over days/weeks
Alpha signal Δp	Post-trade carry

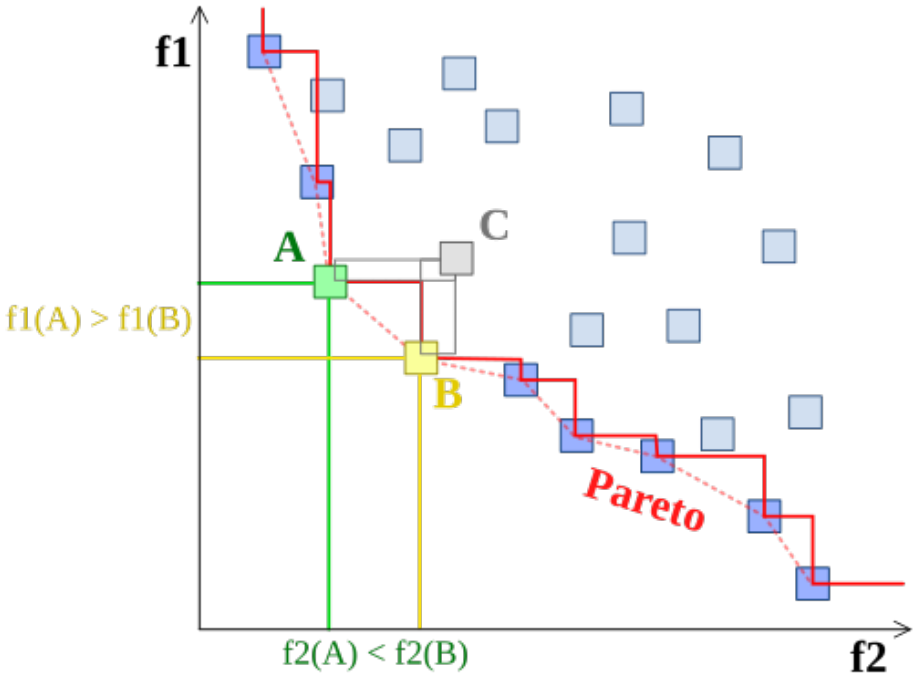


Figure 7: Illustration of a Pareto front in a two-objective minimization problem. Points on the frontier (e.g., A and B) are non-dominated; point C is dominated because both objectives can be improved simultaneously. Source: Wikipedia.

The objective function uses only pre-trade information to select the optimal quote in real time, but its parameters directly control the long-term outcomes. By varying the parameter vector $(\gamma, \mu, \eta, \zeta, o, \alpha)$ and evaluating historical or simulated performance, we can trace out the Pareto front in the space of long-term objectives.

A practical workflow for finding a strong baseline and for ongoing adaptation is:

1. Define the desired long-term objectives (e.g., maximize average daily spread capture, maximize monthly fill rate, minimize 95 % VaR of inventory P&L).
2. Use an evolutionary multiobjective algorithm such as NSGA-II [6] or MOEA/D [39] on historical RFQ data.
3. Generate a population of parameter sets, simulate or back-test each set (Section 17.3), and compute the objective vector.
4. Evolve the population over generations to approximate the Pareto front.
5. Select a baseline configuration from the front — often the "knee point" that offers the best compromise — as the daily baseline.
6. Re-optimize periodically (e.g., weekly) or trigger re-optimization when long-term metrics deviate significantly from targets.

Because the Pareto front is typically low-dimensional (2–4 objectives), traders can visualize it interactively and move along the frontier in real time by adjusting a small number of high-level sliders (e.g., "Aggressiveness vs. Risk" or "Profit vs. Market Share"). This turns the complex multi-parameter tuning problem into an intuitive decision-making tool aligned with the desk's strategic priorities at any given moment.

In summary, multiobjective optimization transforms the objective function parameters from static knobs into a dynamic control surface that continuously adapts to changing market conditions and shifting desk mandates, while always respecting the fundamental trade-offs revealed by the Pareto front.

12 Markets

There are many OTC fixed income markets. For this paper we focus on the following:

1. US Treasuries (UST)
2. European Government Bonds (EGB)
3. US Corporate Bonds

For the CTF model we discuss relative to USTs and EGBs. For the FMV we are focused on U.S. corporate bonds, for illustrative purposes. We have implemented an FMV model for U.S. corporate bonds, and the FMV could also be implemented in terms

of EGBs and USTs given the necessary data (which is unavailable in a FINRA TRACE-like system but could be obtained through a partnership). Similarly, the CTF model can be implemented in terms of U.S. corporate bonds, and an analysis can be performed for a CTF model as was done for USTs and EGB.

12.1 US Treasuries

1. If the reference dealer won the trade, we know both the winning price and the cover price.
2. If tied with the winning price, we know we tied and thus we know the winning price, and the cover price is also the traded price.
3. If we covered we know the cover price (because it was our quote), but we don't know the winning quote.
4. If the RFQ traded away, we know neither the winning nor the cover price.
5. If the client cancelled they didn't trade with anyone.
6. We always know the number of dealers during the trade and also in the historical record.
7. If the reference dealer didn't respond in time, we don't know the final state of the trade.

12.2 European Government Bonds

In the European Government Bond (EGB) market, market making dealers operating in the TradeWeb marketplace stream non-binding quotes to potential clients. Clients who wish to buy or sell a particular bond will send a request for quote (RFQ) to some number of dealers, and some number of those dealers will respond with quotes to the client. The client will select zero or one of the returned quotes to trade. All trades are recorded in a system called MIFID II which allows all dealers to see the winning quote. If a dealer submitted the winning quote for an RFQ, the dealer is also able to see the "covered quote," which is the next best quote to the winning quote. All of this information is recorded in a historical table, and each RFQ has the fields of information in table 3.

12.3 US Corporate Bonds

The advantage of OTC trading with U.S. Corporate bonds over USTs is that more is known about the outcome of the trade than for USTs. For example, for U.S. Corporate Bonds, the market making dealer can match their own records of the RFQs with the data obtained from the FINRA TRACE system. FINRA requires dealers to report all of the corporate bond trades to the TRACE system within 15 minutes. This not only gives a better view of the current market in real time, it also helps create a set of historical

data for the dealer where the final winning price and the cover price are known. On the other hand, the corporate bond market can be a much more challenging market to accurately price and predict the chance to fill probabilities, because of the vast number of issuers and issues in the secondary market. Later in this paper we will try to address machine learning architectures that can scale to the complexity of the credit market.

12.4 European Corporate Bonds

The European corporate bond market has similar advantages and disadvantages to the U.S. corporate bond market, now that the MIFID II system has been put in place. Similar to the U.S. FINRA TRACE system, it requires dealers to report trades promptly, and the trades are disseminated to those listening to the feed, increasing transparency.

13 Current FMV Model Data

13.1 Features

We have several groups of features derived from TRACE:

- RFQ-level features
- Recent trades from the same:
  - Bond
  - Issuer
  - Sector
  - Credit Rating

Each recent trade includes the features in the RFQ field list below, plus the price and yield of the trade.

13.2 RFQ Fields

The RFQ-level features fed into the model include:

- Execution date
- Report date
- Coupon
- Time to maturity
- Quantity
- Buy/sell indicator

- Buyer commission
- Buyer capacity
- Seller commission
- Seller capacity
- Side
- Locked in
- Trading market indicator
- Ats indicator
- Credit Rating
- Sector
- Issue date
- Issuer

See section 18.1 for a list of features which include features which are coming soon to the FMV model, but are not yet implemented.

13.2.1 Recent Trade Features

Each recent trade from the same and related bonds includes the above RFQ features, as well as the price. These features (such as dates and prices) were normalized in an effort to transform the raw non-stationary time series into an approximation of stationary, from the perspective of the model. Standard techniques for doing this include translation and transforming to return space.

13.3 Labels

The training labels are the actual prices of the target trades.

14 CTF Model Data

We define the *reference dealer* as the dealer that has collected the historical data we are attempting to model.

Table 3: RFQ fields

Field	*Description*	*Type of Distribution*	*Available at Arrival*
Arrival Time	Time of arrival of the RFQ	Continuous	Yes
ISIN	The identifier of the bond	Categorical	Yes
Number of Dealers	The number of dealers who quoted the RFQ	Discrete	No
Ref. Dealer Quote Time	Time dealer responded with a quote	Continuous	No
Client Trade Time	Time client selected a quote	Continuous	No
Final State	Won, covered, traded away, not traded	Categorical	No
Notional	Size of the proposed trade	Discrete	Yes
BSI	Buy/sell indicator	Boolean	Yes
Price	Price quoted by reference dealer	Discrete	No
Winning Price	The best quote accepted by the client	Discrete	No
Cover Price	The second best quote when traded	Discrete	No

14.1 RFQ Fields

Table 3 lists the properties of the RFQs. As explained previously, not all of the fields below will be known and recorded in the historical log, depending on the market.

In table 4 is a list of the possible end states of the RFQs. The descriptions are written by the authors under their own understanding and should be double-checked with data experts at the dealer.

For the purposes of our model we will call both ClientCancelled and ClientTimedOut "NotTraded", and call both DealerRejected and DealerTimedOut "NotQuoted". With that adjustment we have table 5.

14.2 Fields Useful for Predicting the CTF

We have all of the attributes of the RFQ that are known at the time of the arrival of the RFQ as indicated in table 3 that we can use as signals for our CTF model. We also have the time series of RFQs that have arrived and traded in the past for the same and related ISINs and clients. We also have the current state of the futures markets and also the streaming price quotes available from TradeWeb. Further macro and related signals may be helpful, such as broad-index real-time movements.

We assume that all features that are experimentally found to be useful will be included in the s_i vector.

15 Modeling U.S. Credit FMV

For the purposes of this section focused on FMV, we focus on U.S. credit, as that is the scenario implemented initially.

Table 4: RFQ Final States

Final RFQ State	*Description*
Done	The RFQ traded with the reference dealer
DoneTied	The RFQ traded with the reference dealer, but the quote was tied with that of at least one other dealer
Covered	The reference dealer had the second best quote to the one that the client traded
CoveredTied	The reference dealer tied with the winning quote, but did not win the trade
TradedAway	The reference dealer did not win the trade or cover the trade (there are at least two quotes better than the reference dealer's quote)
DealerRejected	The reference dealer chose not to quote the RFQ
DealerTimedOut	The reference dealer did not quote by the RFQ's deadline
ClientCancelled	Sometime after submitting the RFQ, the client cancelled the RFQ
ClientTimedOut	The client did not accept any of the quotes returned by the dealers before the deadline

The objective of the Fair Market Value model is to estimate the median value of the price of bond, *given*:

1. That it is reported immediately, notated as the random variable Z
2. Input x which includes all of the features designated in section 13.1
3. Parameters θ of the neural network, (more on why we chose neural networks in 19.1), which estimates the median value

We designate this estimate as $\tilde{y}$, where Y is the random variable representing the price of a trade reported by FINRA TRACE. Below M represents a function which computes the estimated *median* given Z, θ, x.

$$\tilde{y} = M[Y|Z, \theta, x] \tag{53}$$

The parameters are trained using a variant of stochastic gradient descent as mentioned in section 19.1 called adadelta. [38]

15.1 FMV Model's Loss Function: MAE

Optimizing for the median solution rather than the expected value reduces the influence of outliers and improves the final out-of-sample mean absolute error. When training a machine learning model, if the objective is to estimate the expected value of the label given the features, mean squared error is normally used as the loss function for the model, as mean squared error is the solution for maximum likelihood estimation assuming that the error term of our model's specification is normally distributed. [12, p. 130-131] However, if instead we wish to estimate the median using a machine learning model, it is

Table 5: RFQ Final States Simplified

Final RFQ State	*Description*
Done	The RFQ traded with the reference dealer
DoneTied	The RFQ traded with the reference dealer, but the quote was tied with that of at least one other dealer
Covered	The reference dealer had the second best quote to the one that the client traded
CoveredTied	The reference dealer tied with the winning quote, but did not win the trade
TradedAway	The reference dealer did not win the trade or cover the trade (there are at least two quotes better than the reference dealer's quote)
NotQuoted	The reference dealer did not return a quote in time to the client
NotTraded	The client did not accept any of the quotes returned by the dealers before the deadline

well understood that we should use mean absolute error as the loss function for training our model:

Quoting the math stack exchange:

> ... you can indeed use derivatives to justify the median as the minimizer of the absolute value loss. Considering $X \sim F$, where F has density f for simplicity, we are interested in:
>
> $$min_{c\in\mathbb{R}}\mathbb{E}[|X-c|] = min_{c\in\mathbb{R}}(-0.5\int_{-\infty}^{c}(x-c)f(x)dx + 0.5\int_{c}^{\infty}(x-c)f(x)dx) \quad (54)$$
>
> Now we can try to get to our first order condition. When the variable (here c) is in the integral boundary this can be done via the Leibniz rule. Taking derivatives with respect to c then leads to the first order condition:
>
> $$0 = 0.5\int_{-\infty}^{c} f(x)dx - 0.5\int_{c}^{\infty} f(x)dx = 0.5F(c) - 0.5(1-F(c)) = F(x) - 0.5 \quad (55)$$
>
> where I assume that we can interchange limits and integrals. The last equation now is now equivalent to $c = F^{-1}(0.5)$ showing that the median is a stationary point. To show that the median is actually the minimum you can consider the function $g(c) = \mathbb{E}[|X-c|]$ and show that it is convex, which follows from the convexity of $|x|$.

Although the problem we are solving in mapping our trade input features to our outputs is not convex, this result implies is that in a locally convex region, minimizing MAE results in a predictor that approximates the median.

15.2 Predicting the Probability Density of the FMV Median Estimates

When we have an FMV model prediction for a given RFQ/hypothetical trade, a good question to ask is what is the confidence interval in our estimate of the median of such a trade? We have computed the overall RMSE across our test set, but we can take this further and compute a confidence interval which is customized to the current market conditions. [5] details how applying dropout resampling of the neural network output and then Conformal Prediction outperforms applying Conformal Prediction to random forest models. We propose doing so and adding this to our implementation road-map. It is mostly a software engineering task as using this method does not require retraining the model. It does require inferring from the model N times for each RFQ we wish to price, so there is an order of magnitude increase in raw computation required for this method, without consuming additional memory.

16 Modeling the CTF

As noted in section 12, for USTs we do not always know what the best competing price was historically, depending on the final status. For both USTs and EGBs, the client may have chosen not to trade with any of the dealers who posted quotes for the RFQ because the dealers did not meet the client's reserve price, which is a theoretical minimum price that would have induced the client to trade with one of the dealers. For example, even though a client may not have been intending to trade the RFQ, and were merely fishing for pricing information when they sent an RFQ, it is reasonable to assume that if the price offered was highly favorable to the client the client would have chosen to trade the quote anyway. In other cases there may be few dealers who quoted the RFQ and so none of the quotes met the client's reserve price.

Also, there may be confounding factors that induce the dealer to not quote historically, which causes us to have 0 probability instead of a non-zero probability of finding out information such as the best competing quote.

There are many methods that have been devised by statisticians to work around missing data issues. Some techniques relate to causal-inference methods such as censorship classification, inverse-propensity score weighting, and outcome regression. The state of the art statistical methods for applied missing data analysis are multiple imputation and maximum-likelihood estimation. [8]

We focus on censorship classification and maximum likelihood estimation (MLE) methods similar to [9]. However, because we are an AI/ML fintech (and because the authors believe it will help the performance of the model and in some ways simplify the problem by utilizing TensorFlow's automatic differentiation capability), we are going to combine MLE with deep learning/stochastic gradient descent related methods.

16.1 Unknowns

We posit that there are two random variables, that, if known, reveal $c(a, s_i)$:

1. $P_{c,i}$: Best competing price for r_i.

2. $P_{r,i}$: The client reserve price for r_i

$P_{r,i}$, the random variable representing the reserve price, is defined as the minimum price (multiplied by the bsi), at which the client is willing to trade the RFQ. "Reserve price" is a term from auction literature [24] and practice. For example, on e-bay a seller can set a minimum reserve price. If none of the buyers bids at least the reserve price, then the good is not sold, i.e. "reserved."

$p_{c,i}$ and $p_{r,i}$ are samples from $P_{c,i}$ and $P_{r,i}$, respectively.

We make the assumptions discussed below about the value of $c(a, s_i)$ depending on the values of $p_{c,i}$ and $p_{r,i}$ in relation to a, reminding the reader that a is the action, the reference dealer's quote.

We define $p_{0,i}$ as the historic winning price and $p_{1,i}$ as the historic cover (second most client-favorable) price (or tied with the winning price but not traded). We are also defining a price ordering here, where $p_{0,i}$ is the most client-favored price for r_i, $p_{1,i}$ is the next-most client favored price, and so on. Included in this list of prices are $p_{c,i}$ and $p_{r,i}$, and the dealer's quote a.

If we know $p_{c,i}$ and $p_{r,i}$, we can assume that if:

$$bsi * a > bsi * p_{c,i} \tag{56}$$
$$bsi * a \geq bsi * p_{r,i} \tag{57}$$

Then $c(a, s_i) = 1$. Likewise if:

$$a = p_{c,i} \tag{58}$$
$$bsi * a \geq bsi * p_{r,i} \tag{59}$$

Then we assume that $c(a, s_i) = 0.5$

If, on the other hand:

$$bsi * a < bsi * p_{c,i} \tag{60}$$

Or:

$$bsi * a < bsi * p_{r,i} \tag{61}$$

Then we assume $c(a, s_i) = 0.0$.

For the quote optimization use case of CTF, $p_{c,i}$ and $p_{r,i}$ are never known at the time of quoting. For back-testing with EGBs we usually know the historical $p_{c,i}$, except in cases where the client did not trade with anyone or the dealer did not submit a quote. For USTs, we usually don't know $p_{c,i}$ unless the reference dealer won the trade. We don't have a historical explicitly recorded $p_{r,i}$ regardless, so it must be modeled. We propose modeling the distribution of $p_{c,i}$ and $p_{r,i}$ for both use cases using MLE.

16.2 Historical $p_{c,i}$ and $p_{r,i}$ Values for EGBs

Here in tables 6 and 7 we document what is known about $p_{c,i}$ and $p_{r,i}$ historically depending on the historical end state of r_i for the EGB market and the UST market, respectively.

Table 6: Values of $p_{c,i}$ and $p_{r,i}$ for EGBs

Final RFQ State	$p_{c,i}$	$p_{r,i}$
Done	$p_{1,i}$	$bsi * p_{r,i} \leq bsi * p_{0,i}$
DoneTied	$p_{0,i}$	”
Covered	”	”
CoveredTied	”	”
TradedAway	”	”
NotQuoted	”	”
NotTraded	Unknown	$bsi * p_{r,i} > bsi * a$

Table 7: Values of $p_{c,i}$ and $p_{r,i}$ for USTs

Final RFQ State	$p_{c,i}$	$p_{r,i}$
Done	$p_{1,i}$	$bsi * p_{r,i} \leq bsi * a$
DoneTied	a	”
Covered	$bsi * p_{c,i} > bsi * a$	Unknown
CoveredTied	a	$bsi * p_{r,i} \leq bsi * a$
TradedAway	$bsi * p_{c,i} > bsi * a$	Unknown
NotQuoted	Unknown	Unknown
NotTraded	Unknown	$bsi * p_{r,i} > bsi * a$

For U.S. credit, the formulas are identical to the EGB case shown here in table 6, because similar to EGB, in the U.S. credit market the winning and cover price are known.

16.3 Calculating $c(a, s)$ from the Distributions of $p_{c,i}$ and $p_{r,i}$

Since at the time of quoting we never know $p_{c,i}$ and $p_{r,i}$, and at the time of back-testing we never know the exact value of $p_{r,i}$ and often don't know the value of $p_{c,i}$ depending on the end state, it is useful to model the respective distributions of the random variables C and R, from which $p_{c,i}$ and $p_{r,i}$ are sampled. We assume that C and R are independent because R is unknown by the dealers and so should not affect their bids. However, we do assume a causal relationship between the state s and both C and R. Suppose that $f(bsi * a, s)$ is the probability density function (PDF) across all values of a for C given state s, and $F(bsi * a, s)$ is the equivalent cumulative distribution function (CDF) of C. Assume that likewise $g(bsi * a, s)$ and $G(bsi * a, s)$ is the equivalent PDF and CDF of R.

Assuming these distributions are known and the prices are continuous, then:

$$c(a,s) = F(bsi * a, s) * G(bsi * a, s) \tag{62}$$

In other words, $c(a,s)$ is the probability that the best competing bid AND the client's reserve price are less than a (when accounting for the bsi). So in other words, equation 62 is simple Boolean logic.

However, the prices are not in fact continuous and the dealers' prices are sometimes tied as discussed above. Therefore, as a convenience we define a^- as the quote one tick less favorable to the client than a:

$$bsi * a^- = bsi * a - k \tag{63}$$

Where k is the size of the tick.

Using this and assuming that the prices come in discrete ticks, we modify equation 62 to be the following:

$$c(a,s) = [F(bsi * a^-, s) + \frac{1}{2} f(bsi * a, s)] * G(bsi * a, s) \tag{64}$$

The $\frac{1}{2}$ multiplier in the second term comes from the assumption that if the reference dealer's price exactly matches the best competing price, then with a coin toss the client will pick one of the dealer's quotes. This is a simplifying assumption that ignores that there may be timing differences or client-dealer relationship differences that affect this choice. This assumption also ignores outcomes where more than two dealers tie at the best price. We leave removing this simplification to further research.

Given that if we know the distributions for C and R, we can compute $c(a,s)$, the question becomes, how do we find these distributions? In the next section we will discuss using maximum likelihood estimation (MLE) to estimate these distributions.

For notational completeness, we introduce estimates $\hat{f}(\theta, bsi * a, s)$ and $\hat{g}(\theta, bsi * a, s)$ which are distribution PDFs estimated using the θ machine learning model parameters.

16.4 The Maximum Likelihood Estimation Model

See [8] for a general introduction to maximum likelihood estimation, and [4] for an overview of how to apply machine learning to MLE.

In summary, calibrating MLE distributions is about finding the parameters of the distributions that maximize the joint likelihood of a set of observations in the historical record. Suppose that we have a set of RFQs $\{r_0, r_1, ..., r_n\}$, then to calibrate the distributions which we want to model, we need to maximize the joint likelihood of this set of RFQs. The set of parameters which minimize (either globally if using a convex solutions, or locally otherwise) the negative likelihood L for a theoretical total population of data (from which we sample observations), we will be maximizing the fit of the distributions to the observed data under whatever distributional assumptions we make:

$$min[J(\{r_0, r_1, ..., r_n\})] = -\prod_{i=0}^{n} L(r_i, s_i) \tag{65}$$

Normally, when dealing with likelihood equations we use the logarithm because it is much more stable numerically than taking a product of a large number of values between 0 and 1:

$$min[logJ(\{r_0, r_1, ..., r_n\})] = -\sum_{i=0}^{n} logL(r_i, s_i) \tag{66}$$

We will then be able to estimate the parameters via a sample of the total population, which we designate later as $\hat{L}$ in section 16.6. So we are going to minimize the negative log likelihood (NLL) $-log(J)$ of the historical RFQs. The equations for the likelihood functions we can use to compute L and thus $log(J)$ are displayed in tables 8 and 9.

16.5 Market Likelihood Functions

Table 8: Likelihood Functions for EGB End States

Final RFQ State	$p_{c,i}$	$p_{r,i}$	$L(a, s_i)$
Done	$p_{1,i}$	$bsi * p_{r,i} \leq bsi * p_{0,i}$	$f(bsi * p_{1,i}, s) * G(bsi * p_{0,i}, s)$
DoneTied	$p_{0,i}$	”	”
Covered	”	”	$f(bsi * p_{0,i}, s) * G(bsi * p_{0,i}, s)$
CoveredTied	”	”	”
TradedAway	”	”	”
NotQuoted	”	”	”
NotTraded	Unknown	$bsi * p_{r,i} > bsi * a_i$	$1 - G(bsi * a_i, s)$

Table 9: Likelihood Functions for UST End States

Final RFQ State	$p_{c,i}$	$p_{r,i}$	$L(a, s_i)$
Done	$p_{1,i}$	$bsi * p_{r,i} \leq bsi * a$	$f(bsi * p_{1,i}, s) * G(bsi * p_{0,i}, s)$
DoneTied	a	”	$f(bsi * a_i, s) * G(bsi * a_i, s)$
Covered	$bsi * p_{c,i} > bsi * a$	Unknown	$1 - F(bsi * a_i, s)$
CoveredTied	a	$bsi * p_{r,i} \leq bsi * a$	$f(bsi * a_i, s) * G(bsi * a_i, s)$
TradedAway	$bsi * p_{c,i} > bsi * a$	Unknown	$1 - F(bsi * a_i, s)$
NotQuoted	Unknown	Unknown	N/A
NotTraded	Unknown	$bsi * p_{r,i} > bsi * a_i$	$1 - G(bsi * a_i, s)$

Again the U.S. credit market likelihood functions are identical to the cases listed in in table 8 for EGBs.

16.6 Estimating the Distributions

The equations for L in tables 8 and 9 is for when the distributions for $p_{r,i}$ and $p_{c,i}$ are known given a and s_i. However, in reality these distributions are not known a priori,

but must be estimated using the data collected by the reference dealer. We do this by first selecting our distributional assumptions for $p_{c,i}$ and p_{r_i}. One option is to select a parameterized distribution such as the normal distribution, Gaussian mixture models, etc. Another option is a non-parameterized or empirical distribution technique such as normalizing flow, quantile regression, etc. Assuming for simplicity in our discussion that we choose a parameterized distribution such as Gaussian, [9] finds constant values of the parameters by solving the equations somewhat similar to the ones in tables 8 and 9, except in their case they are analyzing corporate rather than government bonds, so what is known by the dealer is different and thus the likelihood equations are somewhat different. Also, in their likelihood model, they chose to model not just the distribution of the *best* competing dealer price, but the distributions of *all* of the competing dealer prices, which results in more complex equations. By solving the likelihood equations, they manage to find optimized constant parameters for their assumed distributions of these underlying causal factors for the CTF.

This work takes a machine learning approach to solving the maximum likelihood estimation problem, as stated in the introduction. Rather than solving the likelihood equations analytically to derive static parameter values, we create a set of machine learning models to estimate these parameters numerically, while also allowing the parameters to vary as a function approximation mapping s_i to the distributional parameters.

To simplify this discussion, we assume that the distributions for $p_{c,i}$ and $p_{r,i}$ are normally distributed conditioned on s_i. A similar analysis can be done for a variety of distributional assumption choices and is left for further research.

In this context we define $\hat{f}$, $\hat{g}$, $\hat{F}$, $\hat{G}$ as approximations of f, g, F, and G respectively, in terms of the PDF for the normal distribution using parameters estimated by machine learning models $\mu(\theta, s)$ and $\sigma(\theta, s)$:

$$h(\theta, b, s) = \int_{b^-}^{b} \frac{1}{\sigma(\theta, s)\sqrt{2\pi}} e^{-\frac{x-\mu(\theta,s)}{2\sigma(\theta,s)}} dx \tag{67}$$

$$H(\theta, x, s) = \sum_{b=min(A)}^{x} h(\theta, b, s) \tag{68}$$

$$\hat{f}(\theta, p_{c,i}, s_i) = h(\theta_f, p_{c,i}, s_i) \tag{69}$$

$$\hat{g}(\theta, p_{r,i}, s_i) = h(\theta_g, p_{r,i}, s_i) \tag{70}$$

$$\hat{F}(\theta, p_{c,i}, s_i) = H(\theta_f, p_{c,i}, s_i) \tag{71}$$

$$\hat{G}(\theta, p_{r,i}, s_i) = H(\theta_g, p_{r,i}, s_i) \tag{72}$$

Reminding the reader that A represents the set of all prices that could be quoted for the RFQ; also, b^- represents one price tick below b. $\mu(\theta_f)$ and $\sigma(\theta_f)$ are function approximators for the parameters of the distribution of $p_{c,i}$.

Equations 69-72 are a complete set of distribution approximations to approximate L which we will call $\hat{L}$ as defined in tables 10 and 11.

Table 10: Approximate Likelihood Function for EGB End States

Final RFQ State	$p_{c,i}$	$p_{r,i}$	$\hat{L}(\theta, a, s_i)$
Done	$p_{1,i}$	$bsi * p_{r,i} \leq bsi * p_{0,i}$	$\hat{f}(\theta, bsi * p_{1,i}, s) * \hat{G}(\theta, bsi * p_{0,i}, s)$
DoneTied	$p_{0,i}$	"	"
Covered	"	"	$\hat{f}(\theta, bsi * p_{0,i}, s) * \hat{G}(\theta, bsi * p_{0,i}, s)$
CoveredTied	"	"	"
TradedAway	"	"	"
NotQuoted	"	"	"
NotTraded	Unknown	$bsi * p_{r,i} > bsi * a_i$	$1 - \hat{G}(\theta, bsi * a_i, s)$

Table 11: Approximate Likelihood Function for UST End States

Final RFQ State	$p_{c,i}$	$p_{r,i}$	$\hat{L}(\theta, a, s_i)$
Done	$p_{1,i}$	$bsi * p_{r,i} \leq bsi * a$	$\hat{f}(\theta, bsi * p_{1,i}, s) * \hat{G}(\theta, bsi * p_{0,i}, s)$
DoneTied	a	"	$\hat{f}(\theta, bsi * a_i, s) * \hat{G}(\theta, bsi * a_i, s)$
Covered	$bsi * p_{c,i} > bsi * a$	Unknown	$1 - \hat{F}(\theta, bsi * a_i, s)$
CoveredTied	a	$bsi * p_{r,i} \leq bsi * a$	$\hat{f}(\theta, bsi * a_i, s) * \hat{G}(\theta, bsi * a_i, s)$
TradedAway	$bsi * p_{c,i} > bsi * a$	Unknown	$1 - \hat{F}(\theta, bsi * a_i, s)$
NotQuoted	Unknown	Unknown	N/A
NotTraded	Unknown	$bsi * p_{r,i} > bsi * a_i$	$1 - \hat{G}(\theta, bsi * a_i, s)$

Once again, the U.S. credit approximate likelihood functions are identical to those shown in table 10.

17 Modeling the c.d.f. Functions Directly to Model the CTF

17.1 Introduction

Modeling the CTF through maximum likelihood on the distributions for the underlying causal factors has the benefit that there is a monotonicity guarantee conditioned on the quote. For example, as the quote becomes more favorable for the client, the CTF should always go up. In other words:

$$c(a - \epsilon, s) \leq c(a, s) \tag{73}$$

For any non-negative ϵ and quote a. This is because the integral of any non-negative function is monotonic, so F and G are monotonic with respect to a, and therefore per equation 62, c(a,s) will also be monotonic. The p.d.f.s of f and g are non-negative because they express probability density, which is always non-negative.

Although modeling the p.d.f.s of the underlying causal factors has some great properties, it suffers from a high amount of complexity and only partially solved challenges when approaching them from a machine learning perspective. In theory, a neural net-

work can be trained using maximum likelihood to output the parameters of any assumed distribution or an empirical distribution in the case of normalizing flow, etc., but using non-standard loss functions for neural networks often causes problems for numerical stability during training time. Workarounds and solutions are possible, but further research is needed to ensure that the training using the derived equations in this paper are consistently numerically stable and yield good training results and at this time are aspirational.

It turns out that there is a simpler approach, which is simply to model the cumulative probability density functions F and G directly through logistic regression. Logistic regression is a standard machine learning technique used for binary classification problems wherein it is desirable for the model to output a probability for the outcome. For modeling F the model would be classifying whether the best competing dealer's quote is below a given quote which is included as one of the model's features, along with the same state s that we have been discussing. The G model would classify whether the client's reserve price is below or equal to the best quote, which is given as one of the features, which is labeled according to whether the RFQ filled at all. s should be restricted to only include factors visible to the reference dealer, so that the model's outputs are safe to share with the dealer.

This approach does not have the monotonicity guarantees, but as long as the domain of the best competing dealer quote and client reserve price are within the range which is well represented in the training data (relative to the current market price), the model will nonetheless learn to be monotonic within this range, as it will learn the relationship that all else being equal, the probability of being above these underlying causal factors will go up as the quote increases. This is because it is proven that neural networks of sufficient depth or width are universal function approximators, [15] [13] so given sufficient training data and a good optimization function, the calibrated model will be a locally-optimum approximation of the real data-generation process. Of course, the function being approximated must be stationary for the model to generalize beyond the training data; we suggest that various data normalization techniques such as those discussed in section 18.3 help to make the training and out of sample data stationary, thus improving the model's ability to generalize.

In order for this approach to be correct in terms of causal inference, we need to assume that there are no causal factors which affect both the dealer's propensity to quote a particular level and also affect the propensity of the client to accept the quote, which is not already captured as a feature in our model. If there were such a confounding factor, then it would invalidate the predictions of our model, and would necessitate correcting for that factor by including it in our model's inputs in order to remedy the situation.

17.2 Model Definition

We define c again as shown in equations 62 - 64 for the continuous and discrete cases, respectively.

So for the continuous case (where we simplify the models to assume that a $($ has not) been multiplied by the bsi before being included as an input into the model):

$$c(a,s) = F(a,s) * G(a,s) \tag{74}$$

And for the discrete case (as a reminder we assume there is a 50% probability of the client picking the reference dealer's quote if their quote is tied with the best competing quote):

$$c(a,s) = [F(a^-,s) + \frac{1}{2}(F(a,s) - F(a^-,s)] * G(bsi * a, s) \tag{75}$$

$$= \frac{1}{2}[F(a^-,s) + F(a,s)] * G(bsi * a, s) \tag{76}$$

$$l \leq a \leq r \tag{77}$$

To approximate c, we simply train a logistic regression neural network to create an approximation for F and G we will call $\hat{F}$ and $\hat{G}$ using stochastic gradient descent. For $\hat{F}$ we can create true / false labels by using tuples of (a,s) where a is set to the best competing dealer price for each dealer that participated in the auction. We can further enhance the labeled data set to improve the generalization capability of the model by adding training instances where a is set to values above the best competing dealer's quote.

For G, the true labels will be created paired with input tuples (a,s) where a is set to the the winning dealer's quote, and false labels where a is set to quotes equal to and below the best dealer quote for the auction.

17.3 Back-testing

Suppose that a reference dealer followed some policy π historically and wants to evaluate the effect of a counterfactual π' on their metrics such as P&L, spread capture, risk, market share, etc., then what?

For example, in the case where the RFQ filled by some dealer, even if not by the reference dealer, then the minimum quote needed to win with 100% probability is known, for example in US Credit, because the dealer knows through TRACE matching what the best competing price was in that case. Also, for a quote equal to the best competing price, we can assume that the CTF is 50%.

For the cases where the RFQ did not fill, we can assume that the minimum quote x needed to fill is above the best quote during the RFQ auction, because none of the quotes were equal to or greater than the client's reserve quote. So if we back-test a quote that is less than or equal to the maximum price quoted in the auction, we can assume that the CTF is 0. In the case where a quote a is greater than the best price quoted in the historical auction, we need to adjust for the fact that we know that the client's reserve price is greater than this best price quoted. p_m is the maximum price quoted for the RFQ. So therefore for all quotes $a > p_m$ we compute the CTF which takes the maximum dealer's quote p_m into account as $c(a, p_m, s)$, with $F(a,s)$ set to 1 for the case where $a > p_m$ and $G(a, p_m, s)$ representing G adjusted for the fact that the best dealer quote p_m did not fill and $a > p_m$ for back-testing purposes as follows:

$$c(a, p_m, s) = F(a, p_m, s) * G(a, p_m, s) \tag{78}$$
$$c(a, p_m, s) = 1 * G(a, p_m, s) \tag{79}$$
$$G(a, p_m, s) = \frac{\int_{p_m+}^{a} g(x, s)dx}{\int_{p_m+}^{\infty} g(x, s)dx} \tag{80}$$
$$= \frac{G(a, s) - G(p_m, s)}{1 - G(p_m, s)} \tag{81}$$
$$\hat{c}(a, p_m, s) = \frac{\hat{G}(a, s) - \hat{G}(p_m, s)}{1 - \hat{G}(p_m, s)} \tag{82}$$

Where we define $c(a, p_m, s)$ as the CTF given that the RFQ did not fill with p_m and $a > p_m$, and $\hat{G}(a, s)$ as our logistic regression model for G. The $+$ in p_m+ designates we are starting the integration one discrete price tick above p_m.

So in summary, if we want to estimate a metric such as spread capture which is 0 when the RFQ didn't fill but some recorded value when the RFQ did fill for a set of historical RFQ auctions for a particular reference dealer, we would do the following, with the procedure ctf_simulate with the following inputs:

- A: Array of quotes for the dealer for each RFQ for which we want to simulate the metric
- V: Array of metric values given the respective quotes
- S: Array of market states for each RFQ
- P: Array of best competing dealer quotes for reference dealer
- F: Array of Boolean values indicating whether the RFQ filled

The procedure returns the mean of the value estimated.

18 Data Relevant for FMV & CTF

18.1 Base Features for the CTF Model

The authors propose including all features that are part of s_i that are believed to have a causal relationship with the client's reserve price $p_{r,i}$ and the best competing dealer's quote $p_{c,i}$. We start by including all of the components of the objective function as listed in section 11.7 that are estimable at the time of quoting. Features regarding the RFQ should be included:

1. Size of the RFQ
2. Side of the RFQ

Algorithm 1 Estimates counterfactual (simulation) metric value average

```
function ĉ(a, p_m, s)
    gpm ← Ĝ(p_m, s)
    return [Ĝ(a, s) − gpm]/[1 − gpm]
end function
function CTF_SIMULATE(A, V, S, P, F)
    i ← 0
    v ← 0.0
    n ← A.size()
    while i < n do
        if F[i] then
            v ← v + [(A[i] == P[i]) ∗ 0.5 + (A[i] > P[i])] ∗ V[i]
        else if A[i] > P[i] then
            v ← v + ĉ(A[i], P[i], S[i]) ∗ V[i]
        end if
        i ← i + 1
    end while
    return v/n
end function
```

3. The number of dealers
4. RFQ's bond identifier
5. Recent FINRA TRACE (UST market – FINRA is currently investigating expanding TRACE to treasuries [1]) or MIFID II (EGB) reported OTC transactions for the the bond in question and also for related bonds and benchmark bonds
6. Dealer Runs / Streaming prices for the bond in question and related bonds
7. Bond market indices which have a correlation with the bond in question
8. RFQs including their outcome recorded by the reference dealer from the various OTC electronic trading platforms or voice trading avenues (may need to be reconciled with FINRA TRACE or MIFID II records).
9. Recent transactions previously with the same client
10. News sentiment signals relevant to the issuer of the bond in question
11. Recent equity prices

The above RFQ features would likely need to be obtained in partnership with a primary dealer or OTC trading platform such as TradeWeb, MarketAxess, TrueMid, etc.

There are many more features that could be included and have a causal relationship with the client's reserve price for the RFQ $p_{r,i}$ and the best competing dealer quote $p_{c,i}$. For this reason it is important that whatever machine learning algorithm is chosen allows the addition of new features, as will be discussed in the next section. We will currently use as an implementation example U.S. Corporate Bonds, and break down the FINRA TRACE data and document how the columns will be used as features or labels in the model, and leave such work for USTs and EGBs for further work.

18.2 US Credit FINRA TRACE Features for FMV & CTF

18.2.1 FINRA TRACE Fields

The following table shows the full set of FINRA TRACE fields for U.S. Corporate Bonds and what we do with them. We try to focus on the most relevant features for the problem at hand. You can see that there are several fields we use to trade to the most relevant trade records, but we don't retain them as features, because by filtering the value, such columns would only take a single value, so there would be no information present for the model to pick up on.

Table 12: Our use of FINRA TRACE data columns

Field Name	**Description**	**Filter**
Price	This field represents the reported bond price. Is inclusive of any mark-ups, and/or mark-downs reported by the firm in the trade transaction.	Nome
Record Count Number	Output values are 1,2,3 etc. in sequence	None
Reference Number	Seven digit identifier. Abbreviated Control Number where last 7 digits are used.	None
Trade Status	The type of report. Applicable values are: T = Trade Report X = Trade Cancel C = Cancelled Correction R = New Correction Y = Reversal (a transaction that has been reverse more than 20 days after it was input)	T
TRACE Symbol	Unique FINRA identifier assigned to each issue.	None
Bloomberg Identifier	Unique Bloomberg identifier assigned to each issue.	None

Continued on next page

Table 12 – continued from previous page

Field Name	Description	Filter
CUSIP	Unique identifier assigned to each bond by Standard & Poor's CUSIP Service Bureau. Please note: this field will be blank on the non-CUSIP version.	None
Company Name/Issuer	Identifies the Company/Issuer Name	None
Coupon Rate	Identifies Coupon Rate in a decimal format	None
Maturity Date	Identifies the bond's maturity date YYYYMMDD format	None
Sub-Product	Identifies the type of security reported in the trade. Applicable values are: AGCY = Agency (in agency file only) CORP = Corporates (in corporate file only) CHRC = Church Bonds (in corporate file only) ELN = Equity Linked Notes (in corporate file only)	'CORP'
When Issued Indicator	Indicates if the bond was traded on a 'When Issued' basis. Applicable values are: Y = When Issued N = Regular Way	'N'
Commission Indicator	Indicates if the reported price is inclusive of dealer commission. Applicable values are: Y = commission charged N = commission not charged	None
Quantity	The uncapped par value volume reported on the trade. May include a decimal, if entered (for mixed-lot and baby bond trades).	None
Yield Direction	This field indicates the yield direction for the subsequent Yield field. Applicable values are: - = minus sign indicates a negative yield. Blank field = indicates a positive or zero yield.	None
Yield	This field indicates the effective rate of return earned on a security, expressed as a percentage. The field will be blank if no yield is available. Yield as calculated by FINRA.	None

Continued on next page

Table 12 – continued from previous page

Field Name	Description	Filter
As Of Indicator	This field indicates if the transaction being reported is an As/Of trade or Reversal from a prior business day. Applicable values are: A = As-of R = Reversal3 Blank = regular trade	”
Execution Date	This field represents the date that the trade was executed. Date is in YYYYMMDD format.	None
Execution Time	This field represents the execution time of the trade. Time is in HHMMSS military time format.	None
Trade Report Date	This field represents the date that the trade was reported to TRACE. Date is in YYYYMMDD format.	None
Trade Report Time	This field represents the time that the trade was reported to TRACE. Time is in HHMMSS military time format.	None
Settlement Date	YYYYMMDD format	None
Trade Modifier 3	Z = reported late T = reported after market hours U = reported late after-market hours May be blank.	None
Trade Modifier 4	W = weighted avg price	None
Buy/Sell Indicator	This field is used to identify whether the reported trade is a buy or sell. Applicable values are: B = Buy S = Sell	None
Buyer Commission	Represents the commission rate charged by the buyer, if applicable. Reported as dollar amount.	None
Buyer Capacity	Represents the capacity reported by the buyer. Applicable values are: A = Agency P = Principal	None
Seller Commission	Represents the commission rate charged by the seller, if applicable. Reported as dollar amount.	None

Continued on next page

Table 12 – continued from previous page

Field Name	Description	Filter
Seller Capacity	Represents the capacity reported by the seller. Applicable values are: A = Agency P = Principal	None
Contra Party Indicator	Identifies the type of trade based on the contra party reported. Applicable values are: C = Customer trade D = Inter Dealer trade4 A = Non-Member Affiliate T = ATS	None
Locked In Indicator	Y = Locked In trade Will be blank on trades not reported as Locked-In.	None
ATS Indicator	This field will be populated if the transaction being reported is an ATS execution. An ATS execution can be either where an ATS reports the trade, is a contra party to the trade, or is neither but the trade is executed on an ATS's platform. Y = ATS Trade Blank = Not an ATS Trade	None
Special Price Indicator	Y = Special Price trade	'N'

Continued on next page

Table 12 – continued from previous page

Field Name	Description	Filter
Trading Market Indicator	This field indicates whether a trade was reported as a secondary market trade or a primary market trade. Applicable values are: S1 = Secondary market trade or a primary market trade executed at a market price P1 = Primary market trade that qualifies as a List or Fixed Offering Price transaction, or a Takedown transaction. Trades reported with the S1 trading market indicator are eligible for dissemination. Trades reported with the P1 trading market indicator are not eligible for dissemination and are reportable on a T+1 basis.	None
Dissemination Flag	Indicates whether the trade was disseminated (via BTDS, or ATDS for Agency Bonds) or not. Applicable values are: Y = Trade was disseminated N = Trade was not disseminated	'Y'
Prior Trade Report Date	YYYYMMDD format Populated on Cancels, Corrections and Reversals. Blank on regular Trade Reports.	None
Prior Reference Number	7 digits identifier Populated on Cancels, Corrections and Reversals. Blank on regular Trade Reports.	None
First Trade Control Date	YYYYMMDD format Populated on Cancels, Corrections and Reversals.	None
First Trade Control Number	7 digits identifier Populated on Cancels, Corrections and Reversals.	None

The yield is discarded because it is calculated using the price. We don't want to include as features in the model information that is calculated using a label if that is

Feature Name	Deep MM Name
Price	price
Yield Direction	yield_direction
Yield	yield
Bloomberg Identifier	figi
Company Name/Issuer	company
Coupon Rate	coupon
Maturity Date	maturity
Quantity	quantity
Execution Date	execution_date
Execution Time	execution_time
Trade Report Date	report_date
Trade Report Time	report_time
Buy/Sell Indicator	buy_sell
Buyer Commission	buyer_commission
Buyer Capacity	buyer_capacity
Seller Commission	seller_commission
Seller Capacity	seller_capacity
Contra Party Indicator	side
Locked In Indicator	locked_in
ATS Indicator	ats_indicator
Trading Market Indicator	trading_market_indicator

Table 13: Our retained FINRA TRACE features
s

information that it won't have when the product is in use. In production we don't have the estimate of the yield before we estimate the price itself, so we shouldn't include it as a feature.

In the model preprocessing steps, the report_date and report_time are used to find trades that were reported before the RFQ we are trying to price is executed. However, when we arrange the trades in a sequence for the model to process it will make sense to order the ones we have available in terms of execution time for LSTMs, and to use time since execution for the embedding of time for each trade for transformer/attention-style models.

Having gone through the FINRA TRACE list, here in table 14 are the remaining values we are retaining as features (with price shown above as the single label we will be training towards. Note: we will discuss later how to transform the price to spread space, along with other data transformations we will perform).

We do not use yield & yield_direction_indicator for pricing a particular RFQ during training time, because when we are pricing an RFQ, the yield is not yet known as the price is unknown. However, we can still use price & yield as features for RFQs received in the past.

Feature Name	Transformation
price	Log return
yield	Log return
bond_id	Embedding
company	Embedding
coupon	Log return
maturity	Time to maturity time embedding
quantity	Log
execution_date	Relative time embedding
execution_time	Relative time embedding
report_date	Relative time embedding
report_time	Relative time embedding

Table 14: Industry-Standard FINRA TRACE features transformations

18.3 US Corporate Bond Feature Transformations

Many other features will not be used in their raw form. Each of the following features could reasonably be transformed as indicated in the table below:

18.3.1 Log

We take the log of the quantity because that is how traders typically think of them and thus the quantities end up clustering around certain values, resulting in a logarithmic scale in practice. Taking the log makes training a neural net utilizing the quantity feature easier because it helps prevent numerical problems such as loss saturation.

18.3.2 Log Return

In many financial time series applications, the original time series numeric values for the events or time slices are transformed in such a way that the values have a more *stationary* distribution. For example, during one market regime corporate bonds may have an average price of close to par, and in another regime the average values may be quite a bit lower than par. If the model is trained during the first regime, then it will not generalize well to a feature & label space in the second regime. Therefore, if the prices can be expressed in a more relative term when used as features or when predicted by the model, the problem becomes more tractable to solve using machine learning.

One way to accomplish this is to express prices as log return prices. Suppose that the most recent trade for the same bond (i) before the RFQ we are pricing occurred at time t. Then we will transform any price p into p' for other RFQs for the same bond or other bonds we are using as features to price bond i relative to that:

$$p' = log(\frac{p}{p_{i,t}}) \tag{83}$$

We can use this same procedure for the yield and the coupon (although for the yield and coupon care must be taken by in some cases adding a constant to the numerator and denominator in cases where the values are close to 0 or negative). The reason we take the log is to reduce the effect of outliers.

18.3.3 Embedding

For the bond issuer and other high dimensional index features we can use embedding which is a transformation from a higher dimensional one-hot encoding of the index to a much more manageable lower dimension vector which represents the entity in question. There are multiple ways this embedding can be accomplished. Priciple Component Analysis (PCA) can be done on the covariance matrix between bonds and bond issuers. However, this mapping has to be re-computed every time a bond or issuer is added or subtracted to the universe used during training time, and such an embedding model cannot be used out of sample on bonds which were not trained during training time. For example, if there is a bond which was not issued until a time after the training period, this embedding is not available.

Another, more flexible, approach is to use attributes of the bond or issuer as features for a model who's sole job it is to map between those attributes and the embedding vector. We can still use the covariance matrix approach as we did before, but instead of using PCA, we can create a model which has two stages:

1. Model which maps the bond and issuer attributes to an embedding vector
2. A loss function which takes the cosine similarity between any two embedding vectors and compares it to the covariance of the pair of bonds

Item 2 is a common technique for a machine learning/data science technique called *collaborative filtering* [21], where the task is to create a model of the relationship between entities based on their common utility in some task. For example, collaborative filtering approaches have been used to recognize related songs on Spotify [16] and many other online services. [34]

A variety of machine learning/data science algorithms could be used for learning this mapping between the attributes of the bond and issuer and the embedding vector. We plan on using deep learning/neural network techniques for consistency for tooling purposes with the other modules in our models as discussed in this paper.

18.3.4 Relative Time Embedding

With the invention of transformers, which will be discussed in more detail in section 19.2.2, the ordering that was parsed by legacy algorithms such as LSTMs is no longer there, so it becomes important that the information encoded in the ordering of events is part of the model's learned function approximation. This is typically done by embedding the location information for NLP applications of transformers, and is done with time series information through various time embedding methods. Typically, these methods use

many sinusoidal functions to embed the various dimensions of time. We will experiment with using such sinusoidal functions as well as using time embedding methods which utilize the natural time elements such as seconds, minutes, hours, days of the week, month, and year to understand which types of embedding features become most important in the trained model. It is important to note that all of these time embeddings will be done relative to the RFQ being priced, so that what we are embedding is primarily a time delta. We may experiment with adding some of the time elements as absolute time adjustments used in the estimates of the bond price. For example, it may be helpful in optimizing the log loss of the model to take into account the time of day and day of the week in estimating the confidence interval – we can expect more variance in our error when we are outside of market hours, when the market is opening and closing, etc.

19 Machine Learning Architecture Choices for the FMV & CTF Models

Here we discuss how to specify the functional forms of $\mu(\theta, s)$ and $\sigma(\theta, s)$, which will then be used for training θ and for inference in production.

19.1 Why Neural Networks

We suggest using neural networks for the following reasons:

1. Neural networks are trained using variants of gradient descent, meaning as long as there is a differentiable loss function, then the automated differentiation capability of deep learning libraries such as PyTorch and TensorFlow can automatically perform the mathematical calculations to minimize the loss function. This is important to ensure the flexibility to try different distributional assumptions in maximum likelihood estimation without having to redesign the machine learning kernel. Having this automated differentiation capability also reduces the opportunity to introduce human error in the mathematical computations.

2. Feed forward neural networks are theoretically universal function approximators given either unbounded depth (number of layers) and bounded width [13] or unbounded width [15], and given sufficient IID training data which is also IID with respect to the target problem.

3. For neural networks, there are ready made libraries for training neural networks to output the parameters of a variety of distributions using maximum likelihood estimation. For example, the TensorFlow Probability library already does the heavy lifting of defining numerically sound PDF and CDF layers for the Gaussian distribution

4. Deep neural network (DNN) algorithms can more easily incorporate additional features, because much of the featurization work that was previously done by hand using older machine learning techniques can now be automated. This happens

through the lower levels of the network, learning abstract representations of the input features which can then be used as features for the next layer(s). [19]

5. Neural networks are amenable to multi-task learning

Simple Feed-Forward Neural networks have an input layer, which is a vector z, and each layer of the neural network is composed of a matrix multiplication followed by an activation function ϕ:

$$o_1 = \phi(A_1 x + b_1) \tag{84}$$

$$o_i = \phi(A_i o_{i-1} + b_i) \tag{85}$$

Where $i \in \{1, .., n\}$, where n is the number of layers in the feed forward neural network, and $o_0 = z$. o_n is the output vector of the neural network. If this neural network is outputting the two parameters of the Gaussian distribution, as we have implied previously when discussing the $\mu(\theta, s)$ and $\sigma(\theta, s)$ functions, then o_n will be of dimension 2. Essentially, in the final layer of the neural network we have two final models which each separately infer the μ and the σ parameters using the output o_{n-1} of the previous layer as abstract features. We have two models in one that share many parameters in the lower levels of the model, but have separate parameters that combine the abstract features into the approximation of the μ and σ.

The elements of the A matrices and b bias vectors are the trainable parameters of the neural network. Many different element-wise non-linear functions are used for ϕ, and they all have different pros and cons. Two of the most popular activation functions are Rectified Linear Unit (ReLU), and Sigmoid:

$$ReLU(z_k) = max(0, z_k) \tag{86}$$

$$Sigmoid(z_k) = \frac{1}{1 + e^{-z_k}} \tag{87}$$

where $k \in \{1, .., m\}$, where m is the size of z. If there were no non-linear activation function applied after each linear transformation, then the multiple neural network layers could simply be multiplied together to form a single linear transformation, and the model would be a simple linear transformation. The benefit of ReLU over Sigmoid is that as the number of layers increase, Sigmoid tends to cause vanishing gradient due to its exponential term. [11] On the other hand ReLU is linear when not equal to 0, and therefore does not suffer from the same vanishing gradient issue as Sigmoid. It can suffer from a problem in low dimensional neural networks wherein if an activation is less than 0, there will be no gradient to adjust the weights of the unit during back-propagation, which is the algorithm used to train neural networks with respect to their loss function.

Suppose that we have a FFNN that outputs abstract features o_n, which is a function of x as outlined above, and four final neural networks are added that are used to estimate the μ and σ of the Gaussian distributions f and g as follows:

$$\mu_f = \phi(A_{\mu_f} o_n + b_{\mu_f}) \tag{88}$$
$$\mu_g = \phi(A_{\mu_g} o_n + b_{\mu_g}) \tag{89}$$
$$\sigma_f = \phi(A_{\sigma_f} o_n + b_{\sigma_f}) \tag{90}$$
$$\sigma_g = \phi(A_{\sigma_g} o_n + b_{\sigma_g}) \tag{91}$$

where μ_f, σ_f are the dynamic parameters given the state s of the Gaussian distribution for $p_{c,i}$ and likewise μ_g and σ_g for $p_{r,i}$. Together $\{A_1, .., A_n, b_1, .., b_n, A_{\mu_f}, A_{\sigma_f}, b_{\mu_f}, b_{\sigma_f}\}$ form θ_f referenced in equations 69 and 69, and together $\{A_1, .., A_n, b_1, .., b_n, A_{\mu_g}, A_{\sigma_g}, b_{\mu_g}, b_{\sigma_g}\}$ form θ_g referenced in equations 70 and 72. We will designate $\theta = \theta_f \cup \theta_g$, as both θ_f and θ_g are jointly trained to minimize $-log(J)$.

We propose training θ using standard stochastic gradient descent (SGD) techniques commonly used for optimizing neural networks (as well as linear and other types of models). The benefit is that we can solve for an approximation of the optimal θ which minimizes $-log(J)$ without attempting to solve the likelihood equations or even manually compute the partial derivatives of $-log(J)$, by taking advantage of the automatic differentiation capabilities of modern deep learning tool-kits such as TensorFlow, PyTorch, etc.

In SGD, θ is initialized to some initial vector value θ_0 [11], and updated according to the following update rule for each training example i:

$$\theta_t = \theta_{t-1} + \alpha \nabla logL(\theta, r_i, s_i) \tag{92}$$

In practice the training examples are grouped into mini-batches, and the above gradient update of θ is computed as a summation over the minibatch where m is the size of the mini-batch:

$$\theta_t = \theta_{t-1} + \alpha \sum_{i}^{m} \nabla logL(\theta, r_i, s_i) \tag{93}$$

α is a learning rate constant parameter. A good overview of stochastic gradient descent and related techniques can be found in section 5.9 of [12].

19.2 Other Relevant Common Advanced Deep Learning Algorithms

We have previously discussed the simplest deep learning architecture, the feed forward neural network (FFNN). However, there are several commonly used deep learning neural network designs that can serve as lower layers of the network to help extract features from the list introduced in section 18.1.

19.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [27] are similar to FFNNs in that they both have a hidden intermediate vector representation between the input and the output. However

for RNNs, the hidden representation is kept between different inputs and the hidden "state" is updated as new inputs are received, and the output of the model is either computed after each update of the state (many-to-many), or at the end of the sequence (many-to-one). RNN models are a good hypothesis when the data are arranged as sequential data, for example, for natural language processing (NLP), and time series data (such as sequences of bond trade transactions).

For example, one common NLP task is language translation. Suppose we are translating the following sentence from Portuguese to English:

- Portugues: "Quantos aprendizado de máquina e *outros componentes* diferentes precisam *funcionar em harmonia* para automatizar a negociação de renda fixa?"
- English partial translation: "How many different machine learning and other components *need to work in* <next word>"

Many translation models use RNNs: first there is an encoder RNN model which reads each of the words in the source sentence, updating the hidden state after reading each word. The hidden state at the end of the encoder processing is then used to initialize the hidden state of the decoder RNN, which generates each word as an output, reading each previously generated word as an input. The encoder model is a many-to-one RNN, while the decoder is a many-to-many RNN because it is outputting a word after receiving each input (the previously generated word). RNNs have been shown to outperform older techniques such as linear statistical machine translation models. It is the author's experience that RNNs can outperform the predictive power of auto-regressive and kalman filters for time-series corporate bond pricing data, if the RNNs process the time series asynchronously instead of synchronously. Kalman filters are shown to be a type of RNN with a linear rather than non-linear hypothesis – and RNNs are a universal filter [37]. Kalman filters typically must have many assumptions built in and carefully crafted to take advantage of the input features as they don't have the same capabilities as deep neural networks to automatically learn abstract non-linear representations of the input features.

When most people think of time series analysis, they think of *synchronous* time series, where the data points are equally spaced in terms of time. In the case of corporate bonds, this means that all of the trades that happened during a certain slice of time would be aggregated into summary statistics, such as the mean price and the total volume traded.

The synchronous schema may work well for cases where there is plenty of trading volume occurring inside each time slice. However, for corporate bonds, this is often not the case; also for many derivative products for USTs or EGBs this is not the case, as well as many off the run bonds. For some products, there may be enough trading volume, but many of the less-liquid bonds may not have any transactions in the time slices. For example, many high yield corporate bonds are not traded on a daily basis, so even if the length of the time slice is chosen to be very long (daily), there will be many bonds which have no data present in many of the time slices. Having such a wide time slice may not be ideal for liquid bonds because there is information loss as too many trades are reduced to their summary statistics.

Traditional time series models such as Auto-Regressive (AR) or Auto-Regressive Integrated Moving Average (ARIMA) typically assume a synchronous time series. In the case of bonds, typically each time series' (bond's) parameters are trained separately. This typically means training separate weights for tens of thousands of bonds.

We can get better performance if we take advantage of deep RNNs ability to process more raw features, asynchronously meaning that each trade is a raw input to the RNN. The authors have found that RNNs generally work better when maintaining the state for a single bond rather than multiple bonds, or the entire universe of bonds (but the parameters can and should be shared across all bonds), whereas it is hypothesized that transformers described in the next section will better be able to utilize recent transactions from a variety of bonds because of the attention mechanism.

19.2.2 Transformers/Attention Networks

Transformers are a relatively recent innovation that have greatly improved the scale and accuracy of Natural Language Processing (NLP) networks. The broad idea is that instead of maintaining a single state as is the case for an RNN or LSTM, an attention network pays *attention* to the right set of features that are relevant to each part of the output. Consider again our Portuguese to English translation example from the previous section 19.2.1.

We emphasized the words in the Portuguese sentence and the already generated partial English translation that an attention-based model should focus on as it is translating the next word in English (harmony). Transformers are able to do this through a transformation of each of the input x corresponding to each of the word vectors in the source sentence and the already generated words along with an encoding of their position into three vectors:

1. Query q
2. Key k
3. Value v

There are correspondingly three feed forward neural networks FFN_q, FFN_k, and FFN_v which transform the set of input word embeddings with their corresponding position encodings X into query matrix Q, key matrix K and value matrix V. Each row of the Q, K, and V matrices corresponds to a different word or position.

$$Q = FFN_q(X) \tag{94}$$
$$K = FFN_k(X) \tag{95}$$
$$V = FFN_v(X) \tag{96}$$

Using these values we can compute the attention matrix, where each row is the vector of features for one of the words, with features extracted by "paying attention" via the dot

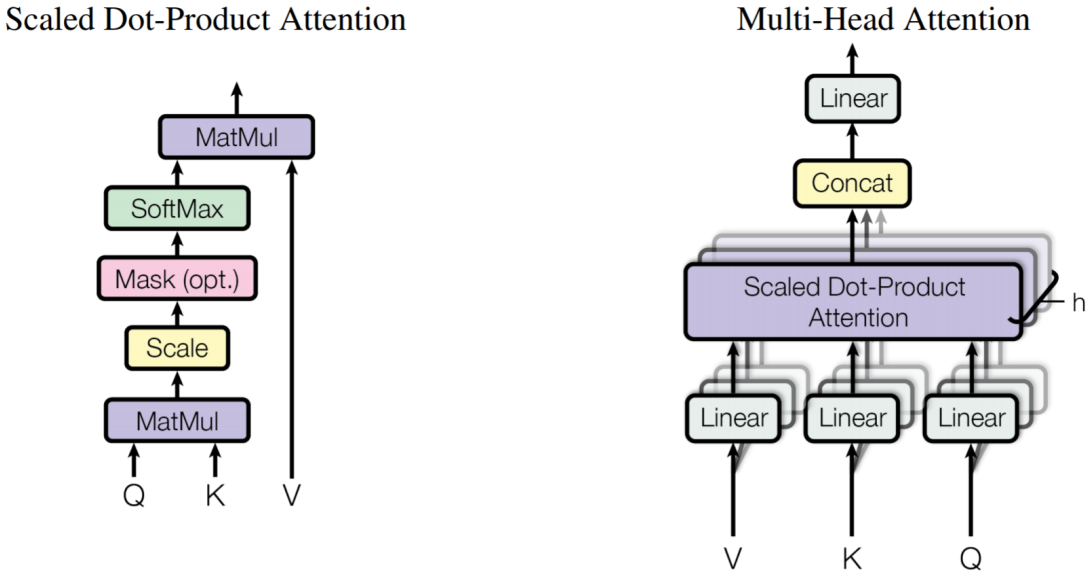


Figure 8: Scaled dot-product attention (left) and multi-head attention (right) [36]

products between the queries and keys inherent from the Q and K matrix multiplication in the attention formula below:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{97}$$

In figure 8 (left) you can see a visual version of the attention equation.

This includes a feature vector specific to generating the next word in the target sentence. The other attention vectors generated in the above formula for all of the other words are useful because the attention layers are often stacked in layers. One way to intuit the benefit of this is to imagine the two sentences as part of a meaning relationship graph. Each word can only be understood in the context of the related words in the source and target sentences. Each layer of attention propagates the relevant information for each word to some degree from the related words, until ideally all of the relevant information for understanding a word in its context has been propagated by the multi-layer attention network. Each layer can be composed of multiple heads of attention in what is called "multi-head attention", which allows each attention head to focus attention on different modalities of related semantic meaning.

To see the full visual overview of the encoder-decoder transformer for language translation architecture, see figure 9. This is a brief overview of transformers that glosses over many details of how they work. For a more thorough explanation see the original "Attention is All You Need" paper. The authors of that paper show that the method is able to outperform previous LSTM/RNN architectures, even when those architectures incorporated attention.

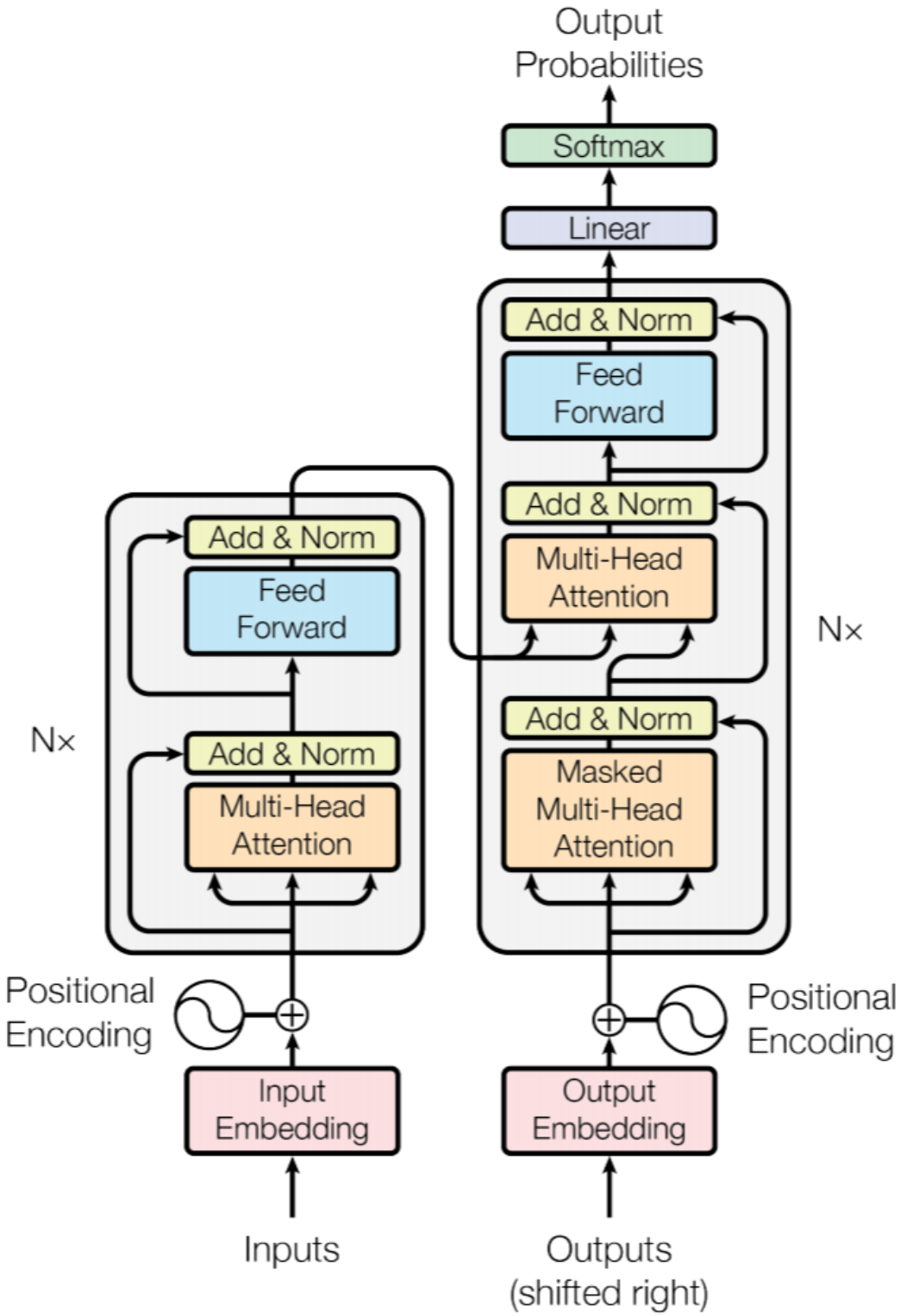


Figure 9: Language translation model architecture utilizing layered multi-head attention transformers. From [36]

Similar to feed-forward networks and RNNs, attention networks are also trained using various forms of stochastic gradient descent, again using the automated partial derivative computation ability of modern deep learning tool-kits, as all of the operations of the transformer layers are differentiable.

How is a neural network architecture originally designed for natural language translation useful for fixed income market making/trading? The authors of this paper hypothesize that transformers out-perform LSTMs by being better able to extract information from relevant related bonds when making a pricing decision. The attention queries and keys can be learned in terms of the bond's properties (which conceptually map to their covariance matrices in terms of how related and thus correlated the prices and other trading properties of any pair of bonds are). This may allow the time series of RFQ trades to be processed in our model as one unified time series of recorded trades, rather than separate time series for each of the bonds, because the attention mechanism of transformers allows the model to focus its attention on trades from the same or sufficiently related bonds, or highly informational benchmark bonds or indices. Rather than utilizing a position encoding as is the case for NLP, for fixed income time series data a time encoding can be made to encode the relative time of each trade. If this paper's proposal is accepted, the authors would enjoy the opportunity to test this hypothesis and compare it to other methods such as RNNs.

Like RNNs, transformers are useful whenever there is a time-series element of the features, but transformers are also useful for when there is a cross-bond dependency aspect regardless of whether the data is arranged as a time series. For example, suppose that each of the individual bond's time series data has already been transformed/filtered so that each bond's state is encoded by an LSTM or other machine learning algorithm. In this case, a transformer can propagate the relevant information between related bonds without the time encoding being included.

20 Feature Importance for the FMV and CTF Models

20.1 Introduction

SHAP (SHapley Additive exPlanations) [22] values are a powerful tool for understanding the contribution of individual features to the output of machine learning models, including financial regression models built using deep neural networks. In this context, SHAP values help to explain and interpret the predictions of the model by assigning an importance value to each input feature. These values can be used to reveal the most influential features that drive the model's predictions, offering valuable insights into the model's decision-making process.

20.1.1 Using Background Sample

To compute SHAP values, we can use a background sample that represents the features for the model at a specific point in time, such as the same time on the last trading day. This background sample serves as a reference to evaluate the importance of each feature

in the model relative to the current trading day. By comparing the SHAP values of the current day to those from the background sample, we can understand how the feature values have changed and the impact of these changes on the model's output.

20.1.2 Percent Change in Regression Output

SHAP values can also be used to ascribe a percent change to the regression output of the model up or down, depending on the value of each feature. This is done by comparing the SHAP value of a feature to the SHAP value of that feature in the background sample. If the SHAP value of a feature has increased, it means the feature is contributing more to the model's output, leading to an increase in the regression output. Conversely, if the SHAP value has decreased, it indicates a decrease in the feature's contribution to the model's output.

20.2 Enhancing Evaluated Pricing Engines and Recommended Quoting Systems for Traders

SHAP values provide valuable insights into the decision-making process of deep neural network-based financial models, such as evaluated pricing engines and recommended quoting systems. By quantifying the contributions of individual input features to the model's predictions, SHAP enables traders to combine their domain knowledge with the model's evaluation of the market, leading to better-informed decisions and improved trading strategies. In this subsection, we discuss how the integration of SHAP values can benefit sell-side and buy-side traders.

20.2.1 Sell-Side Traders

Sell-side traders are responsible for creating liquidity in the market by providing quotes for financial instruments. They need to accurately price these instruments to maximize their revenue while minimizing risk. SHAP values can help sell-side traders in the following ways:

1. Identify key drivers of pricing: By ranking the features according to their SHAP values, traders can identify the most influential factors driving the evaluated pricing engine's predictions. This allows them to focus on the most relevant market information and make informed decisions about quoting.

2. Incorporate domain knowledge: SHAP values enable traders to understand which aspects of the market the model is taking into account and how it's weighting their importance. Traders can then combine their knowledge of market trends, news, or other factors that the model might not consider, to adjust their pricing strategy accordingly.

3. Validate model predictions: By analyzing the SHAP values, traders can validate the model's predictions against their market expertise. If the model's evaluations

are consistent with their understanding of the market, it provides confidence in the model's performance. Conversely, if there's a discrepancy, it may indicate a need for model refinement or additional investigation.

20.2.2 Buy-Side Traders

Buy-side traders are responsible for executing orders on behalf of their clients or firm. They must evaluate the best prices and timings for their trades to maximize returns and minimize transaction costs. DeepSHAP can assist buy-side traders in the following ways:

1. Improve trade execution: By understanding the key factors driving the recommended quoting system, traders can better evaluate the model's suggestions and decide when to execute their trades. This can help them optimize their trading strategy and minimize adverse price impacts.

2. Enhance decision-making: With SHAP insights, traders can incorporate their market knowledge and intuition, alongside the model's evaluation, to make more informed decisions. This helps them identify potential mispricings or market inefficiencies that the model may not fully capture.

3. Evaluate investment opportunities: By analyzing the SHAP values, traders can gain insights into the factors that drive the valuation of different financial instruments. This information can be used to evaluate potential investment opportunities and construct portfolios that align with their investment objectives.

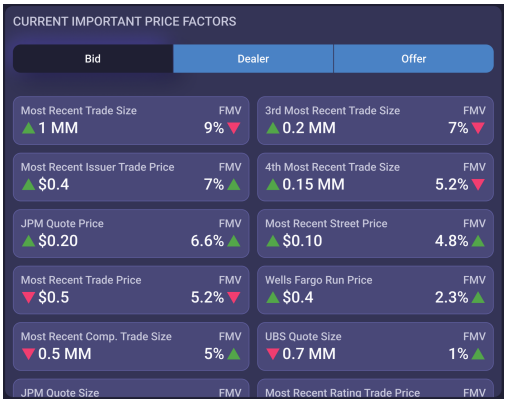
In conclusion, the integration of SHAP values into evaluated pricing engines and recommended quoting systems can greatly benefit both sell-side and buy-side traders. By offering a transparent view of the model's decision-making process, SHAP allows traders to combine their domain knowledge with the model's evaluation of the market, leading to more informed decisions and improved trading strategies. This powerful combination can result in better pricing, enhanced trade execution, and ultimately, increased profitability for traders and their clients.

20.2.3 UI Display for Ranking Features

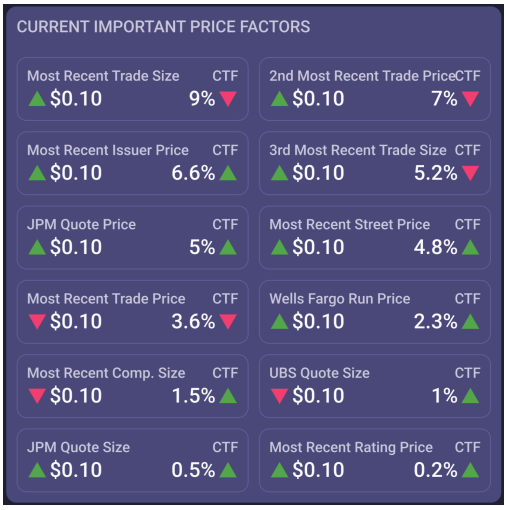
A user interface (UI) display can be designed to showcase the most important features ranked by their SHAP values. In this UI, each feature will be displayed with its SHAP value, allowing users to easily identify the features with the highest impact on the model's predictions. Additionally, the UI can display the change in the feature value from the last trading day to today, providing a comprehensive understanding of how the feature values have evolved over time.

The UI display can include the following elements:

1. A ranked list of features, sorted by their SHAP values in descending order. This list allows users to quickly identify the most influential features in the model.



(a) Fair Market Value (FMV) Model Feature Importance Display Example



(b) Chance to Fill (CTF) Model Feature Importance Display Example

Figure 10: User Interfaces displaying ranked features by SHAP values, including the percent change in feature values from the last trading day for the (a) Fair Market Value (FMV) Model and (b) Chance to Fill (CTF) Model.

2. A visual representation of the SHAP values for each feature, such as bar charts or heatmap. This representation can help users to easily compare the importance of different features.

3. A display of the percent change in the feature value from the last trading day to today, alongside the feature's SHAP value. This information helps users understand the relationship between the feature's importance and its value change over time.

4. An interactive component, such as a filter or search functionality, that enables users to focus on specific features or groups of features, making it easier to analyze the information.

By using SHAP values in conjunction with a background sample, financial professionals can gain a deeper understanding of how changes in feature values influence the output of a deep neural network-based regression model. This knowledge can help them make more informed decisions and fine-tune their strategies based on the most important factors driving the model's predictions.

20.2.4 Advanced UI Enabled by SHAP Feature Importance

In addition to the basic UI elements mentioned earlier, the display can be further enhanced to provide more insights and improve the user experience. Some advanced features that can be added to the UI display are:

1. Temporal Analysis: Incorporate a time series visualization that shows the SHAP values and feature changes over a selected period. This enables users to track how the importance of features and their values have evolved over time, which can help identify trends, seasonality, or sudden shifts in the model's behavior.

2. Correlation Analysis: Display a correlation matrix or scatter plots between the features, showcasing the relationships between the most important features. This can help users understand how the features interact with each other, identify multicollinearity, and make better-informed decisions when adjusting their strategies.

3. Feature Clustering: Group similar features based on their SHAP values or other similarity measures, providing an organized view of the data. This can help users identify clusters of features that contribute similarly to the model's output and potentially discover new relationships or patterns.

4. Sensitivity Analysis: Provide a tool that allows users to simulate changes in feature values and observe how these changes affect the model's output. This can help users understand the impact of specific features on the model's predictions and explore potential scenarios for different market conditions.

5. Interpretation Guidance: Offer contextual information and explanations for the SHAP values and feature changes, helping users interpret the results more effectively. This can include domain-specific insights, such as the implications of a particular feature's change on the market or the expected behavior of certain features under specific circumstances.

6. Customization Options: Allow users to customize the UI display according to their preferences or needs, such as choosing different time frames for the background sample, selecting specific features to focus on, or adjusting the visualization styles.

By incorporating these advanced features into the UI display, financial professionals can gain a more comprehensive understanding of the deep neural network-based regression model and the factors driving its predictions. This increased level of insight can lead to more informed decisions and better risk management, ultimately helping to improve the overall performance of their financial strategies.

In conclusion, SHAP values are a powerful technique for interpreting the importance of features in financial regression models built with deep neural networks. By comparing SHAP values relative to a background sample, such as the features from the last trading day, users can understand the impact of feature changes on the model's output. A well-designed UI display that ranks features by their SHAP values and highlights the changes in feature values can help financial professionals make more informed decisions and optimize their strategies based on the most influential factors driving the model's predictions.

20.3 Calculating SHAP Feature Importance Values

SHAP values are based on cooperative game theory, specifically Shapley values, which offer a fair way to distribute the contribution of individual features to the output of a machine learning model.

Before diving into the mathematical discussion, let's briefly review some key concepts:

Deep neural networks consist of layers of interconnected neurons, where each connection has an associated weight. The network learns these weights during the training process to minimize the error between its predictions and the true target values.

The DeepLIFT algorithm computes the contribution of each neuron's activation to the output by comparing the neuron's activation with a reference activation. These contributions are then backpropagated through the network to obtain the contribution of each input feature. Now, let's discuss the mathematics of how SHAP values can be implemented with deep neural networks using the DeepSHAP method.

Given a deep neural network model, let's denote the prediction function as $f(x)$, where x is the input feature vector. The goal is to explain the output of the model, f(x), in terms of the contributions of individual input features. The DeepSHAP method aims to compute the SHAP values for each input feature i, denoted as $\phi_i(f, x)$.

To compute the SHAP values, we need a set of background samples, denoted as X_b, which can be drawn from the training dataset or created by using a reference value for each feature. In our typical use case, it is the set of features used to make the same prediction at the same time of day the last trading day (but the choice of how far in the past is actually up for the trader to choose), which we designate as x_ref.

Given a prediction function f and a single background instance x_{ref}, SHAP values explain the output of the model $f(x)$ for a specific input instance x by attributing contributions to each input feature. Mathematically, the SHAP value for the i-th feature is defined as:

$$\phi_i(f, x) = \sum_{S \subseteq N \setminus i} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [f_N(x_N) - f_S(x_S)] \tag{98}$$

where:

N is the set of all input features, S is a subset of features not containing the i-th feature, x_S denotes the input instance x with only features in the set S present and the rest replaced by their corresponding values in the background instance x_{ref}, $f_{S \cup i}$ is the prediction function f with only features in the set $S \cup i$ being considered, and $|S|$ and $|N|$ represent the cardinalities of sets S and N, respectively. The SHAP value $\phi_i(f, x)$ represents the average contribution of the i-th feature to the difference between the model's prediction for x and the expected prediction for the background instance x_{ref}.

In the case of deep neural networks, SHAP values can be computed using a variety of approaches. One of the most popular methods is the DeepSHAP or DeepLIFT [30]-based SHAP, which combines the principles of Shapley values with the DeepLIFT algorithm.

For more details on how SHAP values are computed for deep neural networks, see [3].

20.4 Important Properties of DeepSHAP Values

We can evaluate DeepSHAP on three important properties:

1. Local Accuracy
2. Missingness
3. Consistency.

The original SHAP framework, which DeepSHAP is based on, has already proven these properties. Here, we briefly discuss how these properties apply to DeepSHAP.

20.4.1 Local Accuracy

This property ensures that the sum of SHAP values for each input feature, plus the expected output for the background samples, equals the actual output of the model for the given input. Mathematically, it's represented as:

$$f(x) = E[f(X_b)] + \sum_{i=1}^{N} \phi_i(x) \tag{99}$$

DeepSHAP adheres to this property because, by design, it computes the contributions of input features relative to the background samples' expected output. The algorithm backpropagates these contributions through the network, preserving the local accuracy property.

20.4.2 Missingness

This property states that if a feature is missing or has no effect on the output, its corresponding SHAP value should be zero.

DeepSHAP satisfies this property because if a feature is missing or doesn't impact the output, its contribution to the output difference (calculated by DeepLIFT) will be zero. As a result, the SHAP value for that feature will also be zero.

20.4.3 Consistency

The consistency property ensures that if a model relies more on a specific feature, the SHAP value for that feature should not decrease when moving from one model to another.

Let $\hat{f}(x(z')) = f(h_x(z'))$, where z' is a binary vector of size M with elements $z'_j \in 0, 1$, and h_x is a mapping function from z' to the input features x. The notation z'_{-j} indicates that $z'_j = 0$. For any two models f and f' that satisfy:

$$f'(x(z')) - f'(x(z'-j)) \geq f(x(z')) - f(x(z'-j)) \tag{100}$$

for all inputs $z' \in 0, 1^M$, then:

$$\phi_j(f', x) \geq \phi_j(f, x) \tag{101}$$

The consistency property states that if a model changes such that the marginal contribution of a feature value increases or stays the same, regardless of other features, the Shapley value for that feature also increases or stays the same. The Shapley properties, Linearity, Dummy, and Symmetry, follow from the consistency property, as described in the appendix of Lundberg and Lee.

DeepSHAP satisfies the consistency property because it leverages the DeepLIFT algorithm, which computes the contributions by comparing the neuron's activation with a reference activation. When the model relies more on a specific feature, the contribution of that feature to the output difference will increase. As a result, the SHAP value for that feature will also increase.

20.4.4 Conclusion

In conclusion, the DeepSHAP algorithm adheres to the Local Accuracy, Missingness, and Consistency properties, which are essential for ensuring the reliability of the SHAP values. By satisfying these properties, DeepSHAP provides a robust method for interpreting and explaining the predictions of deep neural network models, including financial regression models.

References

[1] Finra seeks comment on expanded trace reporting for government securities. https://www.mondaq.com/unitedstates/financial-services/1021438/finra-seeks-comment-on-expanded-trace-reporting-for-government-securities-. Accessed: 2021-01-15.

[2] Grmela Aleš. Connections in the human brain, 2001.

[3] Marco Ancona, Cengiz Oztireli, and Markus Gross. Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. In *International Conference on Machine Learning*, pages 272–281. PMLR, 2019.

[4] J Brownlee. Probability for machine learning. discover how to harness uncertainty with python. *Machine Learning Mastery*, 2019.

[5] Isidro Cortes-Ciriano and Andreas Bender. Reliable prediction errors for deep neural networks using test-time dropout. *Journal of chemical information and modeling*, 59(7):3330–3339, 2019.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[7] Rinaldo D D'Souza, Quanxin Wang, Weiqing Ji, Andrew M Meier, Henry Kennedy, Kenneth Knoblauch, and Andreas Burkhalter. Hierarchical and nonhierarchical features of the mouse visual cortical network. *Nature communications*, 13(1):503, 2022.

[8] Craig K Enders. *Applied missing data analysis*. Guilford press, 2010.

[9] Jean-David Fermanian, Olivier Guéant, and Jiang Pu. The behavior of dealers and clients on the european corporate bond market: the case of multi-dealer-to-client platforms. *Market microstructure and liquidity*, 2(03n04):1750004, 2016.

[10] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7(10):992, 2019.

[14] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[15] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[16] Christopher C Johnson. Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, 27(78):1–9, 2014.

[17] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237–245, 2019.

[18] Ray Kurzweil. How to make a mind. *The Futurist*, 47(2):14, 2013.

[19] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[21] Run-Ran Liu, Chun-Xiao Jia, Tao Zhou, Duo Sun, and Bing-Hong Wang. Personal recommendation via modified collaborative filtering. *Physica A: Statistical Mechanics and its Applications*, 388(4):462–468, 2009.

[22] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[23] King Ma and Henry Leung. A novel lstm approach for asynchronous multivariate time series prediction. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2019.

[24] R Preston McAfee and John McMillan. Auctions with entry. *Economics Letters*, 23(4):343–347, 1987.

[25] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 06–11 Aug 2017.

[31] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

[32] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[33] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[34] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[37] Wenjie Xu, Xiuqiong Chen, and Stephen S-T Yau. Recurrent neural networks are universal filters. 2019.

[38] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[39] Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

A Objective Function Alpha Parameter Python Simulation Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# ======================================================================
# Parameters
# ======================================================================
num_rfq     = 10000
alphas      = np.round(np.arange(0.00, 1.01, 0.05), 2)
sigma_true  = 0.01
sigma_noise = 0.015
k           = 100.0
m           = 0.01
b           = -1.0          # sell-side
p_fmv       = 100.0
spread_grid = np.linspace(0.0, 0.05, 4000)

# ======================================================================
# Pre-compute CTF
# ======================================================================
ctf_grid = 1.0 / (1.0 + np.exp(k * (spread_grid - m)))   # shape (4000,)

```

```
# ======================================================================
# Vectorized simulation for one alpha value
# ======================================================================
def simulate_alpha(alpha):
    true_dp = np.random.normal(0.0, sigma_true, num_rfq)
    pred_dp = true_dp + np.random.normal(0.0, sigma_noise, num_rfq)

    # Bias from the alpha signal (fixed with b sign for sell-side)
    bias = b * alpha * pred_dp                      # shape (num_rfq,)

    # Objective for every spread and every RFQ
    objectives = ctf_grid[:, np.newaxis] * (spread_grid[:, np.newaxis] +
        bias[np.newaxis, :])

    # Optimal spread per RFQ
    best_idxs = np.argmax(objectives, axis=0)
    s_opt_all = spread_grid[best_idxs]
    c_opt_all = ctf_grid[best_idxs]

    # Simulate fills
    fills = np.random.random(num_rfq) < c_opt_all
    n_fills = np.sum(fills)

    if n_fills == 0:
        return (alpha, 0.0, 0.0, 0, 0.0)

    spread_capture = s_opt_all[fills]          # positive spread capture
    carry = b * true_dp[fills]                 # positive if good for
        position
    avg_spread = np.mean(spread_capture)
    avg_carry  = np.mean(carry)
    total_pnl  = avg_spread + avg_carry

    return (alpha, avg_spread, avg_carry, int(n_fills), total_pnl)

# ======================================================================
# Run simulation (reproducible)
# ======================================================================
np.random.seed(42)
results = [simulate_alpha(alpha) for alpha in alphas]

# ======================================================================
# Print table
# ======================================================================
print("alpha␣|␣spread␣␣␣|␣carry␣␣␣␣|␣fills␣|␣total_pnl")
for r in results:
    print(f"{r[0]:.2f}␣␣␣|␣{r[1]:.5f}␣|␣{r[2]:.5f}␣|␣{r[3]:5d}␣|␣{r[4]:.5f
        }")

# ======================================================================
# Generate plots
# ======================================================================
def quad(x, a, b, c):
```

```
    return a * x**2 + b * x + c

# P&L vs
a_vals   = [r[0] for r in results]
pnl_vals = [r[4] for r in results]
popt, _  = curve_fit(quad, a_vals, pnl_vals)
plt.figure(figsize=(8,5))
plt.scatter(a_vals, pnl_vals, label='Simulation', color='tab:blue')
plt.plot(a_vals, quad(np.array(a_vals), *popt), 'r-', label='Quadratic␣
    fit')
plt.xlabel(r'$\alpha$')
plt.ylabel('Total␣P&L␣(%␣par)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('pnl_vs_alpha.png', dpi=300)
plt.close()

# Spread vs Carry frontier
carry_vals  = [r[2] for r in results]
spread_vals = [r[1] for r in results]
popt2, _    = curve_fit(quad, carry_vals, spread_vals)
plt.figure(figsize=(8,5))
plt.scatter(carry_vals, spread_vals, label='Simulation', color='tab:blue'
    )
xfit = np.linspace(min(carry_vals), max(carry_vals), 300)
plt.plot(xfit, quad(xfit, *popt2), 'r-', label='Quadratic␣fit')
plt.xlabel('Avg␣Post-Trade␣Carry␣(%␣par)')
plt.ylabel('Avg␣Spread␣Capture␣(%␣par)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('spread_vs_carry.png', dpi=300)
plt.close()

print("\nDone!␣Table␣printed␣above.␣PNGs␣saved.")
```

You can now drop the subsection directly after Section 11.6 (or wherever you prefer) and the appendix at the very end of the document. The two figures referenced (`pnl_vs_alpha.png` and `spread_vs_carry.png`) are automatically generated by the Python script when it is executed.