Recursive Market Intelligence through Multi-Machine Geometric Modeling

Madhav Dogra

June 6, 2025

Abstract

Financial market prediction remains a formidable challenge due to its complex, dynamic, and often non-linear nature, influenced by a myriad of quantitative and qualitative factors. This paper introduces a novel framework, Recursive Market Intelligence through Multi-Machine Geometric Modeling (RMI-MMGM), designed to address these complexities. RMI-MMGM proposes a recursive, modular prediction architecture composed of specialized **machines** that (1) ingest diverse social and financial inputs, (2) generate dynamic financial surface equations, (3) project future prices with associated confidence scores, and (4) continuously learn and adapt from feedback across modules. The core hypothesis posits that market behavior can be modeled as projections on a learned, dynamic, non-linear surface defined by Time (X), Volatility (M), a real-time **Positivity Rating (S)**, and a historically-calibrated **Black Impact Factor (β)**. Price (Y) is thus given by $Y = F(X, M, S, \beta) + \varepsilon$. This paper details the conceptual architecture of RMI-MMGM, including the design of its core components: a Black Machine for extracting both the Positivity Rating and its Impact Factor, a Surface Machine for geometric financial modeling using Fourier Feature Networks, and a Projection Machine for predictive reasoning. Furthermore, it outlines the system integration, feedback loops, and self-learning mechanisms crucial for continuous adaptation and evolution. The proposed framework aims to enhance predictive accuracy, provide robust confidence estimation, and offer a more interpretable model of market dynamics.

Keywords: Financial Forecasting, Machine Learning, Geometric Deep Learning, Implicit Neural Representations, Recursive Systems, Sentiment Analysis, Multi-Factor Models, Market Microstructure, Explainable AI.

1 Introduction

The accurate prediction of financial market movements is a central pursuit in quantitative finance, offering significant economic implications. Traditional models, ranging from econometric approaches like ARIMA and GARCH to early machine learning applications, have often struggled to capture the full spectrum of market dynamics, particularly the impact of non-quantifiable information and the inherent non-stationarity of financial time series. Recent advancements in machine learning, particularly deep learning and natural language processing (NLP), have opened new avenues for developing more sophisticated and adaptive forecasting systems.

Despite these advances, existing models often operate as monolithic entities or lack robust mechanisms for integrating heterogeneous data sources and adapting to rapidly changing market regimes. There is a pressing need for architectures that can dynamically learn complex relationships, explicitly model diverse influencing factors, and provide transparent confidence assessments for their predictions.

This paper introduces Recursive Market Intelligence through Multi-Machine Geometric Modeling (RMI-MMGM), a novel framework designed to address these challenges. RMI-MMGM is built upon the Core Hypothesis that market behavior can be effectively modeled as projections on a learned, dynamic, non-linear surface $F(X, M, S, \beta)$, where:

- **X:** Time (e.g., normalized time index $t \in \mathbb{R}$) represents the temporal evolution.
- **M:** Volatility Factor (e.g., VIX, realized volatility $\sigma \in \mathbb{R}^+$) captures market uncertainty and risk.
- **S: Positivity Rating** (a scalar score $S \in [-1, 1]$) which quantifies the net real-time qualitative sentiment derived from news, social media, and other textual sources.

- β: **Black Impact Factor** (a learned scalar $\beta \in \mathbb{R}$) which measures the historical sensitivity of the asset's price to changes in the Positivity Rating, effectively calibrating the **power** of sentiment for a specific asset.
- **Y:** Price (e.g., closing price, adjusted close $P \in \mathbb{R}^+$) is the target variable.

The relationship is formally expressed as:

$$Y = F(X, M, S, \beta) + \varepsilon \tag{1}$$

where ε represents noise or unmodeled factors.

The RMI-MMGM framework is characterized by:

- **Modularity:** Composed of specialized machines, each handling a distinct aspect of the modeling process.
- **Recursion:** Incorporates continuous feedback loops for model refinement and adaptation.
- **Geometric Modeling:** Leverages techniques like Fourier Feature Networks to learn complex financial surfaces.
- **Confidence Estimation:** Explicitly aims to provide confidence scores with its predictions.

This paper details the two primary phases of RMI-MMGM development: Phase 1, focusing on component-level prototypes (Black Machine, Surface Machine, and Projection Machine), and Phase 2, addressing system integration and the implementation of self-learning feedback loops. We believe this architecture offers a significant step towards more robust, adaptive, and interpretable financial market intelligence.

2 Related Work

The RMI-MMGM framework draws inspiration from and builds upon several key research areas:

- **Financial Time Series Forecasting:** Traditional methods (ARIMA, GARCH) and more recent machine learning models (SVMs, Random Forests, LSTMs, Transformers) have been widely applied. RMI-MMGM aims to extend these by explicitly incorporating soft factors and dynamic surface modeling.
- **NLP and Sentiment Analysis in Finance:** Extracting sentiment and topical information from news articles, social media, and financial reports (e.g., earnings calls, SEC filings) has shown promise in predicting market movements. The Black Machine component directly leverages these techniques.
- **Alternative Data in Finance:** The use of non-traditional data sources, such as insider trading reports, short interest, and macro-economic indicators, is increasingly common. RMI-MMGMs Black Machine is designed to fuse such structured data with unstructured text.
- **Implicit Neural Representations (INRs) and Geometric Deep Learning:** INRs, often utilizing sinusoidal activation functions or Fourier features (e.g., SIREN, Fourier Feature Networks), have demonstrated remarkable efficacy in representing complex signals and surfaces. The Surface Machine adapts these concepts for financial modeling.
- **Modular and Multi-Agent Systems:** Breaking down complex problems into smaller, manageable modules that interact and collaborate is a well-established paradigm in AI. RMI-MMGM adopts this for creating specialized **machines**.
- **Reinforcement Learning (RL) in Finance:** RL has been explored for trading strategies and portfolio optimization. The feedback and adaptation mechanisms in RMI-MMGM could potentially leverage RL principles for policy optimization.
- **Explainable AI (XAI) in Finance:** As models become more complex, understanding their decision-making processes is crucial, especially in high-stakes domains like finance. While not its primary focus, the modular nature and the explicit modeling of factors in RMI-MMGM can aid interpretability.

RMI-MMGM differentiates itself by holistically integrating these diverse research threads into a cohesive, recursive architecture specifically tailored for financial market intelligence, with a unique emphasis on learning a dynamic geometric surface.

3 System Overview

The RMI-MMGM framework aims to build a recursive, modular prediction architecture. Its overarching goal is to:

- Ingest diverse social/financial inputs.
- Generate dynamic financial surface equations.
- Project and predict future prices with associated confidence.
- Continuously learn and adapt from feedback across modules.

The system operates on the core hypothesis $Y = F(X, M, S, \beta) + \varepsilon$, where X, M, S, β, and Y are defined as Time, Volatility Factor, Positivity Rating, Black Impact Factor, and Price, respectively. The architecture is envisioned as a collaborative system of specialized machines, as illustrated in fig. 1.

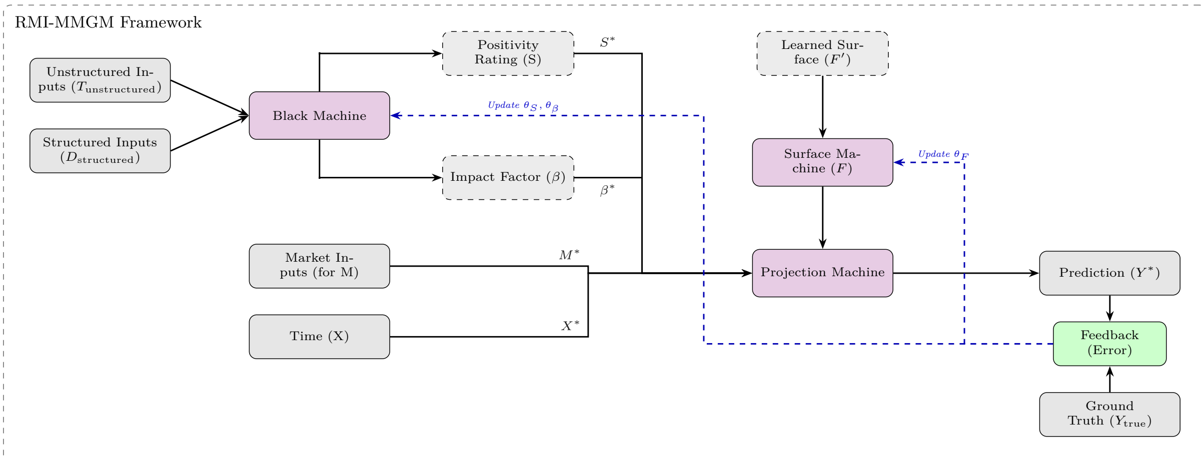


Figure 1: Updated RMI-MMGM architecture with corrected Feedback and Ground Truth placement and arrow updates.

The iterative process involves:

- The **Black Machine** processing unstructured and structured data to derive the **Positivity Rating (S)** and the **Black Impact Factor** (β).
- The **Surface Machine** learning the non-linear function F by mapping historical (X, M, S, β) tuples to corresponding prices (Y), using techniques like Fourier Feature Networks.
- The **Projection Machine** utilizing the learned surface F and estimated future inputs (X^*, M^*, S^*, β^*) to predict future prices (Y^*) and their confidence intervals.
- A **Feedback System** that uses prediction errors to refine the parameters and logic of both the Black and Surface Machines, enabling continuous adaptation.

4 Phase 1: Component-Level Prototypes

This phase focuses on the design, implementation, and individual evaluation of the core machines.

4.1 Black Machine (Social/Soft Factor & Impact Extractor)

Objective: Convert real-world unstructured text and structured data into two distinct outputs: a real-time continuous latent **Positivity Rating** (S), representing non-quantitative market influences, and a periodically calibrated **Black Impact Factor** (β).

Inputs:

- **Unstructured Text ($T_{\text{unstructured}}$):** Earnings call transcripts, SEC filings (10-K, 10-Q), news articles, social media feeds, analyst reports.

- **Structured Data ($D_{\textbf{structured}}$):** Insider trading reports, short interest data, bond yields, macroeconomic indicators (e.g., inflation, unemployment).
- **Historical Market Data ($Y_{\textbf{hist}}, M_{\textbf{hist}}$):** Required for calibrating the Impact Factor.

Outputs:

- **Positivity Rating ($S_t \in [-1, 1]$):** A scalar capturing **soft influence** on the target asset at time t.
- **Black Impact Factor ($\beta \in \mathbb{R}$):** A scalar coefficient representing the historical price sensitivity to the Positivity Rating.

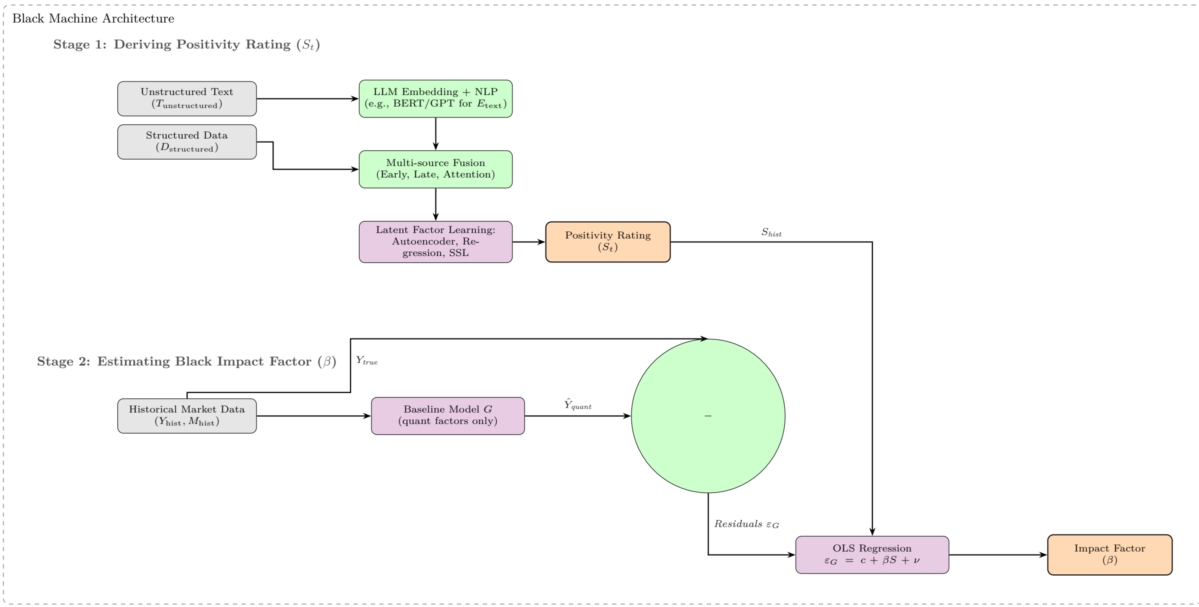


Figure 2: Detailed internal workflow of the Black Machine, showing the two-stage process for generating the Positivity Rating (S_t) and the Black Impact Factor (β).

Approach: The process is divided into two stages. Original latent factor learning techniques from the single-factor model are now re-contextualized as methods for deriving the Positivity Rating.

Stage 1: Deriving the Positivity Rating (S)

- **NLP for Text Processing:**
  - Utilize pre-trained Large Language Models (LLMs, e.g., BERT, GPT variants) to generate dense embeddings:
  $$E_{\text{text}} = \text{LLM}(T_{\text{unstructured}}) \tag{2}$$
  - Employ techniques like sentiment analysis (s_i), topic modeling (z_j), named entity recognition, and event extraction to derive structured features from text.
- **Multi-source Fusion:**
  - Let E_{text} be embeddings/features from text and E_{struc} be embeddings/features from structured data (after appropriate normalization and encoding).
  - **Early Fusion:** Concatenate raw or processed features before feeding into a learning model:
  $$E_{\text{fused}} = \text{Concat}(E_{\text{text}}, E_{\text{struc}}) \tag{3}$$
  - **Late Fusion:** Process text and structured data streams independently and then fuse their outputs: $O_{\text{text}} = \text{Process}_{\text{text}}(E_{\text{text}})$, $O_{\text{struc}} = \text{Process}_{\text{struc}}(E_{\text{struc}})$. The final rating S is a fusion of these outputs.

- **Attention Mechanisms:** If inputs are sequences (e.g., time-series of news articles or reports), an attention mechanism can compute weights α_i to produce a contextually weighted summary.

- **Latent Factor Learning for S:**
  - **Autoencoder:** Define an encoder $g_\phi : X_{\text{input}} \to \mathbb{R}^d$ and a decoder $h_\theta : \mathbb{R}^d \to X_{\text{input}}$. The latent vector can be used to derive S. The objective is to minimize reconstruction loss:

$$\min_{\phi,\theta} \|\text{Input} - h_\theta(g_\phi(\text{Input}))\|^2 \tag{4}$$

  - **Supervised Regressor:** Train a regressor $f_{\text{reg}} : X_{\text{input}} \to \mathbb{R}$ to map fused inputs to an empirically derived **soft influence** metric $S_{\text{empirical}}$ (e.g., based on expert scores or observed market anomalies). Objective:

$$\min \|S_{\text{empirical}} - f_{\text{reg}}(\text{Input})\|^2 \tag{5}$$

  - **Self-Supervised Learning (SSL):** Utilize market reaction (e.g., immediate abnormal returns AR_t post-information release) as a weak supervisory signal for S_t. The loss function could be:

$$L_{\text{SSL}} = \text{Loss}(S_t, \text{AR}_t) \tag{6}$$

    aiming to learn an S_t that correlates with or predicts AR_t.

Stage 2: Deriving the Black Impact Factor (β)

This stage uses historical data to quantify the *effect* of sentiment, separating it from other market drivers. It is performed periodically (e.g., weekly, monthly).

- **Baseline Quantitative Model:** Train a baseline forecasting model G that predicts price changes using only traditional quantitative factors.
- **Calculate Model Residuals:** Over a historical window, calculate the prediction errors (residuals) of this baseline model. These residuals, $\varepsilon_{G,i}$, represent the portion of price movement *not explained* by the quantitative model: $\varepsilon_{G,i} = Y_{i,\text{true}} - G(X_i, M_i, \dots)$.
- **Impact Factor Regression:** The Black Impact Factor β is determined by regressing the unexplained price residuals on the historical sentiment scores (S_i) computed in Stage 1.

$$\varepsilon_{G,i} = c + \beta S_i + \nu_i \tag{7}$$

  The coefficient β is estimated via Ordinary Least Squares (OLS) and captures the marginal price impact of sentiment.

Evaluation:

- **Market Movement Consistency:** Qualitative and quantitative assessment of whether S aligns with significant market events or narrative shifts.
- **Correlation Analysis:** $\text{Corr}(S, M)$ to understand interplay with volatility; $\text{Corr}(S, \Delta P)$ where ΔP is price change, to assess direct relevance.
- **Predictive Power Improvement:** Test if a baseline forecasting model $F(X, M)$ significantly improves when augmented to $F(X, M, S, \beta)$.
- **Interpretability:** Employ methods like LIME or SHAP to understand which input features contribute most to the derived S factor.

4.2 Surface Machine (Fourier Feature Network)

Objective: Generate a non-linear function $Y = F(X, M, S, \beta)$ that accurately models the historical relationship between the input factors and price.

Architecture:

- **Fourier Feature Encoder:** For each input variable v (scalar components of X, M, S, β), the encoding $\gamma(v)$ transforms it into a higher-dimensional feature vector using sinusoidal functions:

$$\gamma(v) = [a_1 \cos(2\pi\omega_1 v + \phi_1), a_1 \sin(2\pi\omega_1 v + \phi_1), \ldots, a_k \cos(2\pi\omega_k v + \phi_k), a_k \sin(2\pi\omega_k v + \phi_k)]^T \quad (8)$$

  where ω_i are frequencies and ϕ_i are phase shifts. Let $X' = \gamma_X(X)$, $M' = \gamma_M(M)$, $S' = \gamma_S(S)$, $\beta' = \gamma_\beta(\beta)$. The input to the subsequent MLP is $Z = \text{Concat}(X', M', S', \beta')$. This encoding helps the MLP learn high-frequency variations effectively.

- **Multi-Layer Perceptron (MLP) for Regression:** A deep neural network $F_{\theta_{\text{MLP}}}(Z)$ with parameters θ_{MLP} takes the Fourier-encoded features Z as input and outputs the predicted price $Y_{\text{pred}} = F_{\theta_{\text{MLP}}}(Z)$. Activation functions like ReLU, SiLU, or sinusoidal activations (for INRs like SIREN) can be used.

- **Implicit Neural Representation (INR) Perspective:** The entire network F can be viewed as an INR, implicitly defining the surface $F : (X, M, S, \beta) \to Y$.

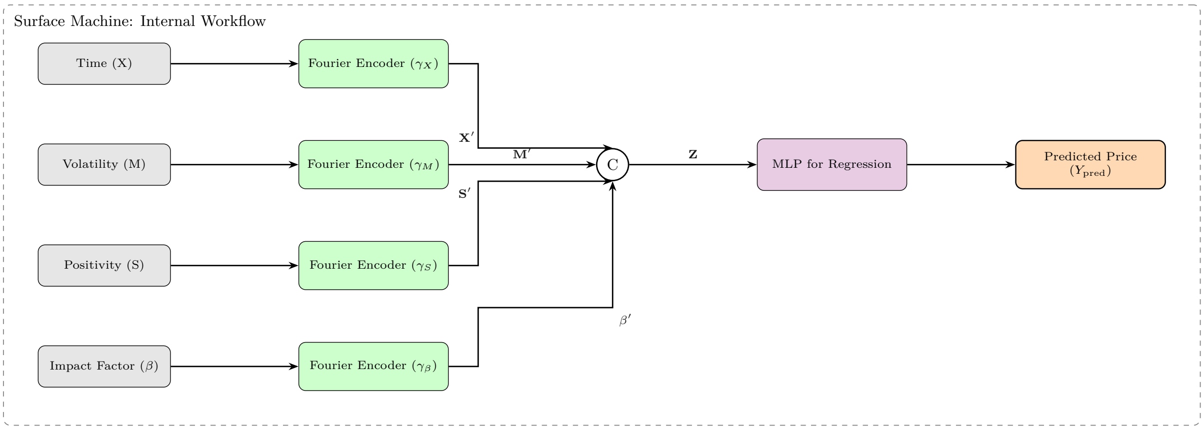


Figure 3: Detailed internal workflow of the Surface Machine (F), showing how the four input factors are individually encoded with Fourier features, concatenated, and then processed by an MLP to predict the price.

Tasks:

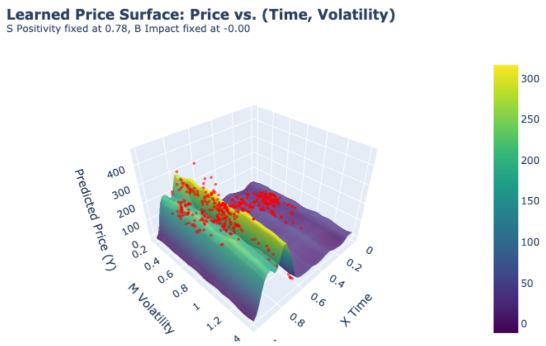
- **Dataset Creation:** Compile a historical dataset $D = \{(X_i, M_i, S_i, \beta_i, Y_i)\}_{i=1}^N$, where S_i and β_i are obtained from the Black Machine.

- **Training:** Optimize the parameters θ_{MLP} (and potentially parameters of γ) by minimizing a suitable loss function (e.g., MSE) on the training dataset.

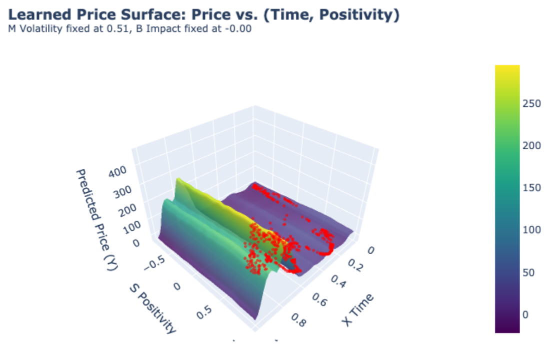
- **Surface Visualization and Evaluation:** Where dimensionality permits (e.g., fixing two variables and plotting Y against two others), visualize the learned surface.

- **Comparison with Baselines:** Compare against simpler regression models (linear regression, gradient boosting) and non-Fourier feature MLPs.

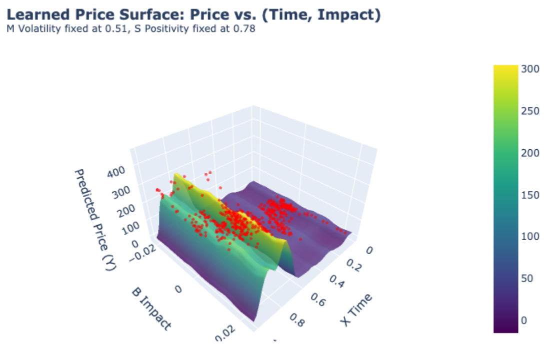
As the model learns a 4-dimensional function $Y = F(X, M, S, \beta)$, direct visualization is impossible. To inspect the learned relationships, we can create 3D surface plots by fixing two of the four input variables at their median values and plotting the price Y against the remaining two. This technique provides interpretable **slices** of the 4D hyperspace. Figure 4 shows the six possible combinations of these visualizations, generated from a model trained on historical data for a sample asset. The red dots represent the actual historical data points, showing how well the learned surface (the smooth manifold) fits the data distribution.



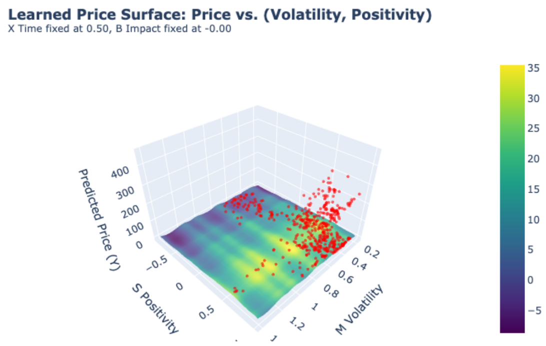
(a) Price vs. (Time, Volatility)



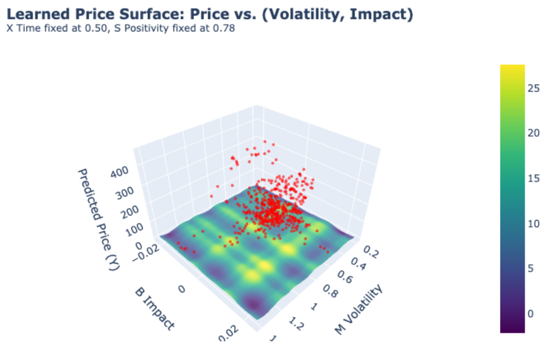
(b) Price vs. (Time, Positivity)



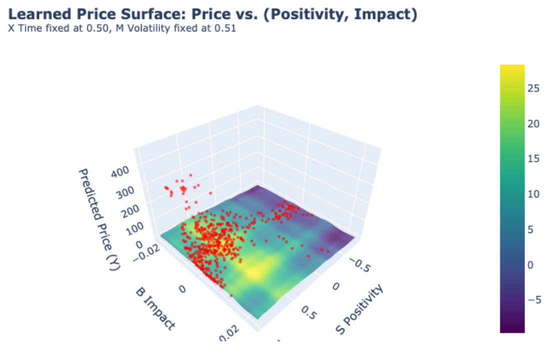
(c) Price vs. (Time, Impact)



(d) Price vs. (Volatility, Positivity)



(e) Price vs. (Volatility, Impact)



(f) Price vs. (Positivity, Impact)

Figure 4: Visualizations of the learned 4D price surface. Each plot shows a 3D slice where two variables are held constant at their median values, illustrating the complex, non-linear relationships captured by the Surface Machine. The plots correspond to the model trained on TSLA data.

The true power of the Surface Machine is not just its predictive accuracy, but its ability to construct an interpretable, multi-dimensional model of market dynamics. As the learned function $Y = F(X, M, S, \beta)$ exists in four dimensions, it cannot be visualized directly. However, by creating 3D **slices**fixing two input variables at their median historical values and plotting the Price against the remaining twowe can gain profound insights into the complex, non-linear interdependencies the model has captured.

Figure 4 presents the six primary slices of the learned 4D hyperspace for a sample asset (TSLA). The smooth, colored manifold represents the model's predicted price function, while the scattered red dots represent the ground-truth historical data. The close correspondence between the data cloud and the learned surface indicates a good fit, but the true value lies in interpreting the geometry of the surfaces themselves.

4.2.1 Interpreting the Learned Financial Surfaces

An analysis of each slice in Figure 4 reveals that the RMI-MMGM framework has learned relationships far more sophisticated than simple linear correlations:

(a) Price vs. (Time, Volatility): This plot uncovers a classic, non-linear market behavior. While the price shows a general upward trend over time (the overall slope of the surface), its relationship with volatility is far more nuanced. The model has learned a pronounced **volatility canyon**: prices are highest at low volatility, drop significantly as volatility increases to moderate levels (reflecting fear and risk-aversion), and then begin to level off or even rise slightly at extremely high volatility regimes, which could correspond to speculative frenzies or **buy-the-dip** capitulation events. A linear model could only ever learn a flat plane and would be fundamentally incapable of capturing this critical U-shaped risk response.

(b) Price vs. (Time, Positivity): This surface provides a stunningly clear validation of the Positivity factor (S). The model has learned a strong, almost linear **ramp** where price increases unequivocally with higher positivity ratings, regardless of the point in time. This demonstrates that the Black Machine is successfully extracting a signal with direct and powerful explanatory relevance to price. The steepness of this ramp is a visual proxy for the average impact of sentiment, a concept refined by the β factor.

(c) Price vs. (Time, Impact): This slice reveals a more subtle, second-order effect. The relationship is not a dramatic ramp, but a gently warped plane. This is financially intuitive: the Impact factor (β) does not, by itself, drive price. Instead, it acts as a *catalyst* for the Positivity factor. The plot shows that periods with a higher Impact factor tend to coincide with higher prices over time, suggesting the model has learned to associate **sentiment-sensitive regimes** with specific market conditions, likely periods of higher growth or narrative-driven trading.

(d) Price vs. (Volatility, Positivity): This is perhaps the most insightful plot, revealing a powerful interaction effect that is a cornerstone of behavioral finance. The model shows that the impact of volatility is conditional on sentiment. When Positivity is high ($S > 0.5$), the surface is high and relatively flat with respect to volatility; in other words, **positive narratives can make the market indifferent to risk**. Conversely, when Positivity is low or negative ($S < 0$), the surface becomes highly corrugated and sensitive to volatility; **fear and uncertainty amplify the market's reaction to risk**. This demonstrates the model's ability to move beyond simple additive factors and learn the crucial state-dependent nature of market psychology.

(e) Price vs. (Volatility, Impact): This plot further explores the system's second-order learning. The wavy surface suggests that the market's response to volatility is fundamentally different depending on the sentiment regime (captured by the Impact factor β). In high-impact regimes (where sentiment is a key driver), volatility might be associated with narrative fervor and large price moves. In low-impact regimes (where fundamentals may dominate), volatility may be treated more traditionally as pure, undiluted risk. The model has learned to differentiate between these contexts automatically.

(f) Price vs. (Positivity, Impact): This surface offers direct visual confirmation of the core hypothesis's term βS. The plot shows a tilted plane where the price rises with Positivity (S), but critically, the **steepness of this rise is amplified by the Impact factor (β)**. When β is high, the slope of the surface along the Positivity axis is steeper, meaning each unit of positive sentiment translates into a larger price increase. This is precisely the dynamic the RMI-MMGM was designed to model, and its clear emergence from the data serves as a powerful validation of the entire framework.

In summary, these visualizations demonstrate that the Surface Machine is not a black box but a sophisticated tool for discovery. It builds a holistic and intuitive model of market behavior, capturing the complex interplay, non-linearities, and conditional dependencies between quantitative and qualitative factors that govern financial markets. This level of interpretability and insight is a significant advancement over models that produce only a single point-prediction.

Evaluation:

- **Root Mean Squared Error (RMSE):**

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - Y_{\text{pred},i})^2} \tag{9}$$

- **Mean Absolute Error (MAE):**

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|Y_i - Y_{\text{pred},i}| \tag{10}$$

- **R-squared ($\mathbf{R^2}$):** Coefficient of determination.
- **Fidelity to Curve Shape:** Qualitative assessment of how well the learned surface captures known market patterns (e.g., volatility smiles if M represents moneyness and Y implied volatility, though here Y is price).
- **Generalization:** Performance on unseen test data.
- **Smoothness/Differentiability:** Check if ∇F exists and is well-behaved, which is important for sensitivity analysis and some optimization techniques. Fourier features generally ensure smoothness.

4.3 Projection Machine (Predictive Reasoner)

Objective: Given future (or hypothesized) inputs X^*, M^*, S^*, β^*, project and predict the future price Y^* with an associated confidence measure.

Inputs:

- The learned surface function F from the Surface Machine.
- Scalar or vector inputs: Future time X^*, estimated future volatility M^*, estimated future Positivity Rating S^*, and the current (or hypothesized) Black Impact Factor β^*.

Tasks:

- **Point Prediction:** Evaluate the surface at the target inputs: $Y^*_{\text{pred}} = F(X^*, M^*, S^*, \beta^*)$.
- **Sensitivity Analysis (Optional but Recommended):** Test with noisy inputs: $M^*_{\text{noisy}} = M^* + \delta M$, $S^*_{\text{noisy}} = S^* + \delta S$ to understand the predictions stability.
- **Confidence Score Generation:**
  - **Monte Carlo Dropout:** During K inference passes with dropout layers active in F (if used), obtain a distribution of predictions $\{Y^*_{\text{pred},k}\}_{k=1}^{K}$.
  - **Mean Prediction:** $\bar{Y}^*_{\text{pred}} = \frac{1}{K}\sum_{k=1}^{K} Y^*_{\text{pred},k}$.
  - **Variance (Uncertainty):**

$$\text{Var}(Y^*_{\text{pred}}) = \frac{1}{K-1}\sum_{k=1}^{K}(Y^*_{\text{pred},k} - \bar{Y}^*_{\text{pred}})^2 \tag{11}$$

    This variance can be decomposed into aleatoric and epistemic uncertainty if the model is designed accordingly.
  - **Ensemble Methods:** Train N_{ens} different Surface Machines (e.g., with different initializations or bootstrapped data). The mean of their predictions serves as $\bar{Y}^*_{\text{pred}}$, and the variance across predictions serves as an uncertainty measure.

  - **Quantile Regression:** Modify the Surface Machines loss function to predict specific quantiles of the conditional distribution of Y. For instance, predict $Q_{0.05}(Y^*|X^*, M^*, S^*, \beta^*)$ and $Q_{0.95}(Y^*|X^*, M^*, S^*, \beta^*)$ to form a 90% prediction interval.
  - **Conformal Prediction:** A model-agnostic technique that can provide statistically rigorous prediction intervals based on calibration on a hold-out set.

Evaluation:

- **Accuracy on Future Prices:** RMSE, MAE on actual future outcomes Y^*_{true}.
- **Calibration of Confidence Scores:** For a $p\%$ confidence interval, assess if the true value Y^*_{true} falls within the interval approximately $p\%$ of the time across many predictions. Plot reliability diagrams.
- **Sharpness of Prediction Intervals:** Narrower intervals are preferred, given they maintain calibration.
- **Sensitivity Analysis:** Evaluate $\partial Y^*/\partial S^*$, $\partial Y^*/\partial \beta^*$, $\partial Y^*/\partial M^*$ to understand the predicted impact of changes in input factors.
- **Timeliness:** Speed of prediction generation.

5 Phase 2: System Integration & Feedback Loops

This phase focuses on connecting the individual machines into a cohesive, adaptive system.

5.1 Integrate Three Machines into Loop

Refined Pipeline:

- **Current State Monitoring (t):** Ingest real-time inputs I_t (text, structured data, market data for M_t).
- **Black Machine Processing:** Generate current Positivity Rating $S_t = \text{BlackMachine}_{\text{Stage1}}(I_t)$. Use most recent β.
- **Future State Estimation ($t + \Delta t$):**
  - Estimate future volatility $M^*_{t+\Delta t}$.
  - Estimate future Positivity Rating $S^*_{t+\Delta t}$. This is challenging and might involve extrapolating trends in S_t, using separate forecasting models for S, or scenario-based analysis.
  - Use current Impact Factor $\beta^*_{t+\Delta t} = \beta_t$.
- **Surface Machine Maintenance:** The Surface Machine $F(X, M, S, \beta)$ is continuously maintained or updated based on new data and feedback.
- **Projection Machine Prediction:** Predict $Y^*_{t+\Delta t} = F(X_{t+\Delta t}, M^*_{t+\Delta t}, S^*_{t+\Delta t}, \beta^*_{t+\Delta t})$ along with confidence.
- **Ground Truth Arrival:** Observe the actual market price $Y_{t+\Delta t,\text{true}}$.
- **Feedback Computation:** Calculate the prediction error $E = Y_{t+\Delta t,\text{true}} - Y^*_{t+\Delta t}$.
- **Feedback Distribution & Updates:**
  - **Black Machine Update:** The error E provides a signal for the efficacy of S_t. If the path is differentiable, parameters θ_S for the sentiment model can be updated.

$$\theta_S \leftarrow \theta_S - \eta_S \frac{\partial L(E)}{\partial S_t} \frac{\partial S_t}{\partial \theta_S} \tag{12}$$

    This is complex. Alternatively, RL can be used. A simpler heuristic: if E is consistently large, it might trigger a full recalibration of β.

- **Surface Machine Update:** The parameters θ_F of the Surface Machine are updated using the new data point $(X_t, M_t, S_t, \beta_t, Y_{t,\text{true}})$. Objective: $\min_{\theta_F} L(Y_{t,\text{true}}, F_{\theta_F}(X_t, M_t, S_t, \beta_t))$. Update rule (e.g., SGD):

$$\theta_F \leftarrow \theta_F - \eta_F \nabla_{\theta_F}(Y_{t,\text{true}} - F_{\theta_F}(X_t, M_t, S_t, \beta_t))^2 \tag{13}$$

  Online learning or mini-batch learning can be applied.

- **Techniques for Feedback Implementation:**
  - **End-to-End Differentiability:** If the entire system from I_t to $Y^*_{t+\Delta t}$ can be made (approximately) differentiable with respect to a shared set of parameters Θ_{total}, then $\Theta_{\text{total}} \leftarrow \Theta_{\text{total}} - \eta \nabla_{\Theta_{\text{total}}} \text{Loss}(Y_{\text{true}}, Y_{\text{pred}})$.
  - **Reinforcement Learning (RL):** Define State $s_t = (X_t, M_t, I_t, \text{current model parameters})$, Action a_t (e.g., parameters for S_t estimation), and Reward $r_t = f(E_t)$.
  - **Adaptive Learning Rates:** Employ techniques like Adam or AdaGrad, or meta-learn learning rates.

5.2 Self-Learning & Evolution

Objective: Enable continuous improvement, adaptation to market regime changes, and robustness against concept drift.

Mechanisms:

- **Memory Buffer (Experience Replay):** Store a large buffer $\mathcal{M} = \{(I_j, X_j, M_j, S_j, \beta_j, Y_{j,\text{true}}, E_j)\}$.
- **Adaptive Retraining Triggers:**
  - **Error-based:** Retrain components if cumulative error exceeds a threshold.
  - **Drift Detection:** Monitor the statistical properties of $P(Y|X, M, S, \beta)$ or the distributions of the factors themselves.
  - **Time-based:** Periodic retraining (e.g., daily, weekly).
- **Adversarial Retraining:** To improve robustness, train the system on perturbed inputs. For the Surface Machine, this involves finding $\delta M, \delta S, \delta\beta$ that maximize prediction error:

$$L_{\text{adv}} = \text{Loss}(F(X, M + \delta M, S + \delta S, \beta + \delta\beta), Y_{\text{true}}) \tag{14}$$

  and then training F to minimize this loss. This can help create smoother, more robust surfaces.
- **Online Adaptation / Meta-Learning:**
  - **Online Learning:** Update model parameters with each new data point or small batch.
  - **Meta-Learning (e.g., MAML):** Learn model parameters θ that can be rapidly adapted:

$$\theta^* = \theta - \alpha \nabla_\theta L_{\text{new\_task}}(\theta) \tag{15}$$

- **Anomaly Detection in Inputs/Outputs:** Implement mechanisms to detect anomalies in input data or in the generated factors S_t or $Y^*_{t+\Delta t}$.
- **Model Versioning and Selection:** Maintain multiple versions of models.

6 Overall System Evaluation Strategy

Evaluating the entire RMI-MMGM framework requires a comprehensive strategy beyond individual component metrics:

- **Backtesting Framework:**
  - Rigorous out-of-sample backtesting on historical financial data across various assets and market conditions.

  - Walk-forward optimization to simulate realistic model training and deployment.
  - Careful consideration of transaction costs, slippage, and data snooping biases.

- **Performance Metrics:**
  - **Financial:** Alpha, Sharpe Ratio, Sortino Ratio, Maximum Drawdown, Profitability.
  - **Statistical:** Overall RMSE/MAE, calibration of system-level confidence intervals.

- **Benchmarking:**
  - Comparison against established baseline models (e.g., GARCH, LSTMs, simpler factor models) and potentially commercial solutions.

- **Ablation Studies:**
  - Systematically remove or simplify components (e.g., run without the β-factor by setting it to 1, remove the S-factor, use a linear surface model) to quantify the contribution of each part of RMI-MMGM.

- **Robustness and Stability:**
  - Stress testing under simulated market shocks or historically volatile periods.
  - Analysis of performance consistency over time.

- **Computational Cost and Scalability:**
  - Evaluate the resources required for training and real-time inference.

7 Discussion and Future Work

The RMI-MMGM framework presents a conceptually powerful approach to financial market intelligence. By explicitly modeling time, volatility, a real-time Positivity Rating, and a data-driven Impact Factor within a dynamic geometric surface, it has the potential to capture complex market behaviors that elude simpler models. The modular design facilitates specialized development and upgrades, while the recursive feedback loops are crucial for adaptation in ever-changing market environments.

Challenges and Limitations:

- **Data Requirements:** The Black Machine, in particular, requires access to diverse, high-quality, and timely data streams, as well as extensive historical archives for robust β calibration.
- **Factor Estimation:** Deriving a meaningful and predictive Impact Factor (β) is a significant research challenge. The estimation of $S^*_{t+\Delta t}$ (future S) is particularly speculative.
- **Computational Complexity:** Training sophisticated LLMs, Fourier Feature Networks, and managing periodic recalibration of β can be computationally intensive.
- **Interpretability:** While modularity helps, the internal workings of deep learning components can still be opaque. Continuous effort in XAI is needed.
- **Market Reflexivity:** Large-scale deployment of such a model could, in theory, influence the market it aims to predict.
- **Overfitting:** The complexity of the model requires careful regularization and validation to prevent overfitting to historical data.

Future Work:

- **Advanced Factor Modeling:** Explore dynamic models for β itself, allowing it to vary with market regimes (e.g., β might be higher in volatile markets). Explore graph neural networks for relational data in S-factor construction, incorporate more diverse alternative datasets, and develop more robust methods for $S^*_{t+\Delta t}$ forecasting.

- **Alternative Surface Parametrizations:** Investigate other geometric deep learning techniques beyond Fourier features for the Surface Machine, such as those based on differential geometry or topology.
- **Sophisticated RL for Feedback:** Develop more advanced RL agents for managing feedback and adaptation, possibly using hierarchical RL.
- **Causality-Informed Modeling:** Incorporate techniques to infer and leverage causal relationships between factors, rather than relying solely on correlations.
- **Hardware Acceleration:** Explore specialized hardware (TPUs, GPUs) and distributed computing frameworks to manage computational demands.
- **Integration with Portfolio Optimization:** Extend the framework to directly inform portfolio construction and risk management decisions.

8 Conclusion

The Recursive Market Intelligence through Multi-Machine Geometric Modeling (RMI-MMGM) framework offers a comprehensive and adaptive approach to financial forecasting. By decomposing the problem into specialized machines for soft factor extraction, dynamic surface modeling, and predictive reasoning, and by integrating these through recursive feedback loops, RMI-MMGM aims to achieve superior predictive performance and robustness. The explicit modeling of time, volatility, a real-time Positivity Rating, and a calibrated Black Impact Factor on a learned non-linear surface represents a novel paradigm in quantitative finance. While significant research and engineering challenges remain, the successful development and deployment of such a system could provide a profound leap in our ability to understand and navigate the complexities of financial markets.

A Appendix

This appendix contains the Python code for the core components of the RMI-MMGM framework. The implementations use standard data science and machine learning libraries such as PyTorch, Transformers, and Scikit-learn.

A.1 Black Machine Factor Extraction Code

The following script implements the two-stage process for the Black Machine, as described in Section 4.1.

- **Stage 1 (Positivity Rating 'S'):** The script scrapes recent news headlines for a given ticker from multiple sources (yfinance and Google News). It then uses a pre-trained financial sentiment model (FinBERT) to analyze these headlines and compute an aggregate, real-time Positivity Rating (S).
- **Stage 2 (Black Impact Factor 'β'):** It first trains a baseline quantitative model (XGBoost) on historical price and volume data to generate price predictions. The errors (residuals) of this model represent price movements not explained by simple quantitative factors. The script then fetches and analyzes the sentiment of historical news for the same period. Finally, it performs an Ordinary Least Squares (OLS) regression of the residuals against the historical sentiment scores. The coefficient of the sentiment term in this regression is the Black Impact Factor (β), quantifying the historical price sensitivity to sentiment.

```
import pandas as pd
import numpy as np
import yfinance as yf
import torch
import xgboost as xgb
import statsmodels.api as sm
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from sklearn.model_selection import train_test_split
import requests
from bs4 import BeautifulSoup
from datetime import datetime, timedelta
import time

# --- SETUP ---
print("Loading financial sentiment model (FinBERT)...")
tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert")
print("Model loaded.")


# --- PART 1: POSITIVITY RATING (S) ---
def analyze_sentiment(text_list: list) -> float:
    if not text_list: return 0.0
    inputs = tokenizer(text_list, padding=True, truncation=True, return_tensors='pt',
    max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
    scores = predictions[:, 0] - predictions[:, 1]  # pos - neg
    return np.mean([s.item() for s in scores])


def scrape_google_news(query: str, period_days: int = 2) -> list:
    """Scrapes Google News for a given query and time period."""
    headlines = []
    end_date = datetime.now()
    start_date = end_date - timedelta(days=period_days)

    # Format for Google News URL
    query = query.replace(" ", "+")
    url = f"https://news.google.com/search?q={query}&after={(start_date).strftime('%Y-%m
    -%d')}&before={(end_date).strftime('%Y-%m-%d')}&hl=en-US&gl=US&ceid=US:en"

    try:
        headers = {"User-Agent": "Mozilla/5.0"}
```

```python
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')

        # Google News articles are in 'a' tags with class 'JtKRv'
        articles = soup.find_all('a', class_='JtKRv')
        for article in articles:
            headlines.append(article.text)
    except requests.exceptions.RequestException as e:
        print(f"Error scraping Google News: {e}")

    return list(set(headlines))  # Return unique headlines


def get_positivity_rating(ticker_symbol: str) -> float:
    """Calculates the real-time Positivity Rating (S) using multiple data sources."""
    print(f"\n--- Calculating Positivity Rating (S) for {ticker_symbol} ---")
    stock = yf.Ticker(ticker_symbol)
    company_name = stock.info.get('longName', ticker_symbol)

    # Source 1: yfinance
    yfinance_headlines = []
    try:
        yfinance_headlines = [item.get('title') for item in stock.news if item.get('title')]
    except Exception as e:
        print(f"Could not fetch news from yfinance: {e}")

    # Source 2: Google News Web Scraping
    google_headlines = scrape_google_news(f"{company_name} stock", period_days=3)

    # Combine and deduplicate headlines
    all_headlines = list(set(yfinance_headlines + google_headlines))

    if not all_headlines:
        print(f"No recent news found for {ticker_symbol} from any source.")
        return 0.0

    print(f"Found {len(all_headlines)} unique headlines for '{company_name}'.")
    positivity_score = analyze_sentiment(all_headlines)
    print(f"Aggregated Positivity Rating (S) for {ticker_symbol}: {positivity_score:.4f}")
    return positivity_score


# --- PART 2: BLACK IMPACT FACTOR () ---
def train_baseline_model(hist_prices: pd.DataFrame):
    df = hist_prices.copy()
    for i in range(1, 6):
        df[f'close_lag_{i}'] = df['Close'].shift(i)
        df[f'volume_lag_{i}'] = df['Volume'].shift(i)
    df = df.dropna()
    features = [col for col in df.columns if 'lag' in col]
    X, y = df[features], df['Close']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

    model_xgb = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, random_state=42)
    model_xgb.fit(X_train, y_train)
    predictions = model_xgb.predict(X_test)
    residuals = y_test - predictions

    return pd.DataFrame({
        'Actual_Price': y_test,
        'Quant_Predicted_Price': predictions,
        'Residual_Error': residuals
    }, index=y_test.index)


def get_historical_sentiment(ticker_symbol: str, dates: pd.DatetimeIndex) -> pd.Series:
    """Fetches historical news for a list of dates and calculates sentiment."""
    print("Fetching and analyzing historical sentiment. This may take several minutes...
```

```python
    ")
    sentiments = {}
    stock = yf.Ticker(ticker_symbol)
    company_name = stock.info.get('longName', ticker_symbol)

    for date in dates:
        # To scrape for a single day, set the period to one day
        headlines = scrape_google_news(f'"{company_name}"', period_days=1)
        sentiments[date] = analyze_sentiment(headlines)
        time.sleep(0.5)  # Be polite to the server

    print("Historical sentiment analysis complete.")
    return pd.Series(sentiments)


def calculate_black_impact_factor(ticker_symbol: str) -> float:
    """Calculates the Black Impact Factor () using REAL historical data."""
    print(f"\n--- Calculating Black Impact Factor () for {ticker_symbol} ---")

    print("1. Training baseline quantitative model...")
    stock = yf.Ticker(ticker_symbol)
    hist_prices = stock.history(period="2y")  # Use a shorter period for faster historical scraping
    residuals_df = train_baseline_model(hist_prices)
    print(f"   Baseline model trained. Found {len(residuals_df)} residuals.")

    # 2. Get REAL historical sentiment for the same period
    print("2. Calculating REAL historical sentiment for each day...")
    # This is the major upgrade - no more simulation
    historical_sentiments = get_historical_sentiment(ticker_symbol, residuals_df.index)
    residuals_df['Positivity_Rating'] = historical_sentiments.reindex(residuals_df.index).fillna(0)

    print("3. Regressing residuals on sentiment to find the Impact Factor ()...")
    y_reg = residuals_df['Residual_Error']
    X_reg = sm.add_constant(residuals_df['Positivity_Rating'])

    model_ols = sm.OLS(y_reg, X_reg).fit()
    black_impact_factor = model_ols.params.get('Positivity_Rating', 0.0)
    p_value = model_ols.pvalues.get('Positivity_Rating', 1.0)

    print("\n--- Black Impact Factor Regression Results ---")
    print(model_ols.summary())

    print(f"\nThe calculated Black Impact Factor () for {ticker_symbol} is: {black_impact_factor:.4f}")
    if p_value < 0.05:
        print(f"The result is statistically significant (p-value = {p_value:.4f}).")
    else:
        print(f"The result is not statistically significant (p-value = {p_value:.4f}).")

    return black_impact_factor


if __name__ == '__main__':
    target_ticker = 'TSLA'
    current_S = get_positivity_rating(target_ticker)
    historical_beta = calculate_black_impact_factor(target_ticker)

    print("\n" + "=" * 50)
    print("      BLACK MACHINE FINAL OUTPUTS")
    print("=" * 50)
    print(f"For Ticker:             {target_ticker}")
    print(f"Current Positivity (S): {current_S:.4f}")
    print(f"Impact Factor (beta):      {historical_beta:.4f}")
    print("=" * 50)
```

Listing 1: Python code for the Black Machine implementation.

A.2 Surface Machine Training and Visualization Code

The following script provides a complete implementation for training the Surface Machine and generating the 3D surface visualizations shown in Figure 4. It uses PyTorch for the neural network, yfinance for data retrieval, and Plotly for interactive graphing.

For the purpose of this standalone script, the historical values for the `S_positivity` and `B_impact` factors are simulated using a random walk and an exponentially weighted moving average, respectively. In the full RMI-MMGM system, these historical data series would be generated by the Black Machine implementation detailed in Appendix A.1.

```python
import pandas as pd
import numpy as np
import yfinance as yf
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import plotly.graph_objects as go
from itertools import combinations
# New import for surface smoothing
from scipy.ndimage import gaussian_filter


# --- 1. ARCHITECTURE DEFINITION (Enhanced Regularization) ---

class FourierFeatureEncoder(nn.Module):
    def __init__(self, input_dims: int, embed_dims: int, scale: float = 10.0):
        super().__init__()
        b_matrix = torch.randn(input_dims, embed_dims // 2) * scale
        self.register_buffer('b_matrix', b_matrix)

    def forward(self, v: torch.Tensor) -> torch.Tensor:
        if v.dim() == 1:
            v = v.unsqueeze(-1)
        proj = 2 * np.pi * v @ self.b_matrix
        return torch.cat([torch.sin(proj), torch.cos(proj)], dim=-1)


class SurfaceMachine(nn.Module):
    """
    MODIFIED: Increased model capacity and dropout for better generalization.
    """

    def __init__(self, fourier_embed_dims: int = 64, mlp_hidden_dims: int = 256, mlp_layers: int = 4,
                 dropout_rate: float = 0.3):
        super().__init__()
        self.encoder_x = FourierFeatureEncoder(1, fourier_embed_dims)
        self.encoder_m = FourierFeatureEncoder(1, fourier_embed_dims)
        self.encoder_s = FourierFeatureEncoder(1, fourier_embed_dims)
        self.encoder_b = FourierFeatureEncoder(1, fourier_embed_dims)

        total_input_dims = 4 * fourier_embed_dims

        layers = []
        in_dims = total_input_dims
        for _ in range(mlp_layers):
            layers.append(nn.Linear(in_dims, mlp_hidden_dims))
            layers.append(nn.SiLU())
            layers.append(nn.Dropout(dropout_rate))
            in_dims = mlp_hidden_dims
        layers.append(nn.Linear(mlp_hidden_dims, 1))

        self.mlp = nn.Sequential(*layers)

    def forward(self, x, m, s, b):
        x_encoded = self.encoder_x(x)
        m_encoded = self.encoder_m(m)
        s_encoded = self.encoder_s(s)
        b_encoded = self.encoder_b(b)
```

```python
        z = torch.cat([x_encoded, m_encoded, s_encoded, b_encoded], dim=-1)
        return self.mlp(z)


# --- 2. DATASET CREATION & PREPARATION ---

def create_surface_dataset(ticker_symbol: str, period: str = "20y"):
    print(f"\n--- Creating Historical Dataset for {ticker_symbol} over {period} ---")
    stock = yf.Ticker(ticker_symbol)
    hist = stock.history(period=period)

    if len(hist) < 60:
        raise ValueError(f"Historical data for {ticker_symbol} is too short ({len(hist)}
     days).")

    df = pd.DataFrame(index=hist.index)
    df['Y_price'] = hist['Close']
    df['X_time'] = pd.Series(np.linspace(0, 1, len(hist)), index=hist.index)
    log_returns = np.log(df['Y_price'] / df['Y_price'].shift(1))
    df['M_volatility'] = log_returns.rolling(window=30).std() * np.sqrt(252)
    s_innovations = np.random.randn(len(hist)) * 0.05
    s_simulated = np.cumsum(s_innovations)
    df['S_positivity'] = pd.Series(np.tanh(s_simulated), index=hist.index)
    beta_innovations = np.random.randn(len(hist)) * 0.1
    df['B_impact'] = pd.Series(beta_innovations, index=hist.index).ewm(span=200).mean()

    df = df.dropna()
    if df.empty:
        raise RuntimeError("DataFrame is unexpectedly empty after processing.")

    original_df = df.copy()
    df = df.astype(np.float32)
    print(f"Generated dataset with {len(df)} samples.")

    scalers = {}
    for col in df.columns:
        scalers[col] = MinMaxScaler()
        df[col] = scalers[col].fit_transform(df[[col]])

    print("All features and target normalized.")
    return df, original_df, scalers


# --- 3. TRAINING AND EVALUATION ---

def train_model(model, train_loader, val_loader, epochs=50, lr=1e-3, weight_decay=1e-4):
    """
    MODIFIED: Increased weight_decay for stronger L2 regularization.
    """
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)

    print("\n--- Starting Model Training (with Enhanced Regularization) ---")
    for epoch in range(epochs):
        model.train()
        train_loss = 0.0
        for x, m, s, b, y in train_loader:
            optimizer.zero_grad()
            outputs = model(x, m, s, b)
            loss = criterion(outputs, y)
            loss.backward()
            optimizer.step()
        train_loss = np.sqrt(loss.item())  # RMSE

        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for x, m, s, b, y in val_loader:
                outputs = model(x, m, s, b)
                loss = criterion(outputs, y)
                val_loss = np.sqrt(loss.item())

        if (epoch + 1) % 10 == 0:
```

```python
            print(f"Epoch {epoch + 1}/{epochs} | Train RMSE: {train_loss:.4f} | Val RMSE: {val_loss:.4f}")

    print("--- Training Complete ---")
    return model


# --- 4. VISUALIZATION (Major Enhancements) ---

def plot_interactive_surface(model, normalized_df, original_df, scalers, x_axis_var, y_axis_var):
    """
    Generates a detailed, smoothed, and robust interactive 3D surface plot.
    """
    model.eval()

    all_vars = ['X_time', 'M_volatility', 'S_positivity', 'B_impact']
    fixed_vars = [v for v in all_vars if v not in [x_axis_var, y_axis_var]]

    # 1. Higher Resolution Grid
    grid_size = 50
    x_range = np.linspace(normalized_df[x_axis_var].min(), normalized_df[x_axis_var].max(), grid_size)
    y_range = np.linspace(normalized_df[y_axis_var].min(), normalized_df[y_axis_var].max(), grid_size)
    x_grid, y_grid = np.meshgrid(x_range, y_range)

    input_data = {}
    input_data[x_axis_var] = torch.from_numpy(x_grid.flatten()).float()
    input_data[y_axis_var] = torch.from_numpy(y_grid.flatten()).float()

    for var in fixed_vars:
        median_val = normalized_df[var].median()
        input_data[var] = torch.full_like(input_data[x_axis_var], median_val)

    with torch.no_grad():
        y_pred_normalized = model(
            input_data['X_time'].unsqueeze(1),
            input_data['M_volatility'].unsqueeze(1),
            input_data['S_positivity'].unsqueeze(1),
            input_data['B_impact'].unsqueeze(1)
        ).numpy()

    y_pred_rescaled = scalers['Y_price'].inverse_transform(y_pred_normalized).reshape(x_grid.shape)

    # 2. Surface Smoothing using Gaussian Filter
    # A sigma of 1.0 provides a good amount of smoothing to remove noise.
    smoothed_y_pred = gaussian_filter(y_pred_rescaled, sigma=1.0)

    # Rescale axes to their original values for plotting
    x_grid_rescaled = scalers[x_axis_var].inverse_transform(x_grid)
    y_grid_rescaled = scalers[y_axis_var].inverse_transform(y_grid)

    # --- Create the Plot ---
    # Trace 1: The Smoothed Surface
    surface_trace = go.Surface(
        z=smoothed_y_pred, x=x_grid_rescaled, y=y_grid_rescaled,
        colorscale='Viridis',
        opacity=0.8,
        hovertemplate=f'{x_axis_var.split("_")[1].title()}: %{{x:.2f}}<br>{y_axis_var.split("_")[1].title()}: %{{y:.2f}}<br>Price: %{{z:.2f}}<extra></extra>'
    )

    # 3. Overlaying Actual Data Points
    # Take a random sample to avoid cluttering the plot
    sample_df = original_df.sample(n=min(500, len(original_df)), random_state=42)
    scatter_trace = go.Scatter3d(
        x=sample_df[x_axis_var],
        y=sample_df[y_axis_var],
        z=sample_df['Y_price'],
        mode='markers',
        marker=dict(size=2, color='red', opacity=0.6),
```

```python
        name='Actual Data Points'
    )

    fig = go.Figure(data=[surface_trace, scatter_trace])

    # --- Final Touches ---
    fixed_vals_str = ", ".join(
        [f"{v.replace('_', ' ').title()} fixed at {original_df[v].median():.2f}" for v in fixed_vars])
    fig.update_layout(
        title=f"<b>Learned Price Surface: Price vs. ({x_axis_var.split('_')[1].title()}, {y_axis_var.split('_')[1].title()})</b><br><sup>{fixed_vals_str}</sup>",
        scene=dict(
            xaxis_title=x_axis_var.replace('_', ' ').title(),
            yaxis_title=y_axis_var.replace('_', ' ').title(),
            zaxis_title='Predicted Price (Y)',
            aspectratio=dict(x=1, y=1, z=0.7)  # Adjust aspect ratio for better viewing
        ),
        margin=dict(l=65, r=50, b=65, t=90),
        legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.1)
    )
    fig.show()


def visualize_all_surfaces(model, normalized_df, original_df, scalers):
    print("\n--- Generating 6 Interactive Surface Plots (will open in browser) ---")
    input_vars = ['X_time', 'M_volatility', 'S_positivity', 'B_impact']
    for var1, var2 in combinations(input_vars, 2):
        print(f"Plotting Price vs. ({var1}, {var2})...")
        plot_interactive_surface(model, normalized_df, original_df, scalers, var1, var2)


# --- MAIN EXECUTION BLOCK ---
if __name__ == '__main__':
    ticker = 'TSLA'
    norm_dataset, orig_dataset, scalers = create_surface_dataset(ticker)

    X_data = torch.from_numpy(norm_dataset[['X_time', 'M_volatility', 'S_positivity', 'B_impact']].values)
    y_data = torch.from_numpy(norm_dataset['Y_price'].values).unsqueeze(1)

    (x_train, x_val, y_train, y_val) = train_test_split(X_data, y_data, test_size=0.2, shuffle=False, random_state=42)

    train_dataset = TensorDataset(x_train[:, 0], x_train[:, 1], x_train[:, 2], x_train[:, 3], y_train)
    val_dataset = TensorDataset(x_val[:, 0], x_val[:, 1], x_val[:, 2], x_val[:, 3], y_val)

    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64)

    surface_model = SurfaceMachine()
    trained_model = train_model(surface_model, train_loader, val_loader)

    visualize_all_surfaces(trained_model, norm_dataset, orig_dataset, scalers)

```

Listing 2: Python code for Surface Machine implementation and visualization.

References

[1] Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control*. Wiley.

[2] Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3), 307-327.

[3] Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *Review of Financial Studies*, 33(5), 2223-2273.

[4] Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), 654-669.

[5] Loughran, T., & McDonald, B. (2011). When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks. *The Journal of Finance*, 66(1), 35-65.

[6] Kearney, C., & Liu, S. (2014). Textual sentiment in finance: A survey of methods and models. *International Review of Financial Analysis*, 33, 171-185.

[7] Corum, K., et al. (2019). *Alternative Data in Investment Management*. CFA Institute Research Foundation.

[8] Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., & Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33.

[9] Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., ... & Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33.

[10] Stone, P., & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3), 345-383.

[11] Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059.*

[12] Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., ... & Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82-115.

[13] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

[14] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[15] Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30.

[16] Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 20.

[17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

[18] Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning.*

[19] Vovk, V., Gammerman, A., & Shafer, G. (2005). *Algorithmic Learning in a Random World*. Springer.

[20] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[21] Gama, J., liobait, I., Bifet, A., Peciukait, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4), 1-37.

[22] Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*.