LIFTING THE VEIL OF NON-STATIONARITY IN FINANCIAL MARKET

Tianhao Fu*†, Xinxin Xu*, Weichen Xu, Ruilong Ren, Bowen Deng,
Xinyu Zhao, Jian Cao‡, Xixin Cao
Peking University
tianhaofu1@gmail.com, xuxinxin@stu.pku.edu.cn, caojian@ss.pku.edu.cn

ABSTRACT

Financial asset price movement prediction is inherently challenging due to the non-stationary nature of financial markets, where data distributions shift over time. Existing methods often assume that the market is stationary, which limits their applicability. To address this, we propose the Market-State Jump Diffusion Framework (MSJD), which models non-stationarity through two key components: an Explicit Market-State Jump Diffusion Process (EMJD) and an Implicit Market-State Jump Diffusion Process (IMJD). EMJD captures the dynamics of diffusion, drift, and jump processes governed by latent market states, formulated as stochastic differential equations, to explicitly model non-stationarity and solved via neural networks. IMJD integrates these components into a multi-modal large language model, enabling predictions across varying market conditions through temporal point encoding and jump diffusion embeddings to learn the non-stationary implicitly. Additionally, we introduce a general modality synthesizer that employs a unified adversarial masking strategy to complete missing modalities and fine-tune the prediction model. Furthermore, we validate our framework on the TSLA 2023 earnings crash, demonstrating that MSJD effectively identifies structural regime shifts. Extensive experiments on real-world stock and cryptocurrency datasets demonstrate that our method significantly outperforms existing approaches in the prediction of price movements. Project page and Code: https://iclr26-workshop-nonstationary.github.io/.

1 INTRODUCTION

Predicting asset prices is crucial in finance Lewellen (2004). Deep neural networks show potential in price prediction Martinez-Miera & Repullo (2019); Sezer et al. (2020), and preliminary studies have shown promising results Rath et al. (2024); Kabir et al. (2025). However, most methods assume a stationary environment. As shown in Figure 1, financial markets are non-stationary Li et al. (2014). For example, momentum correlations invert between trending and reversal regimes Avellaneda & Stoikov (2008). This suggests that the role of eigenvalues is inextricably linked to the shifting environment alongside breaking news and global events Guéant et al. (2013); Graves & Graves (2012). We validate this deficiency in Appendix A through a case study of the TSLA 2023 crash.

Recent works utilize neural networks Anderson (1976) to learn price characteristics in nonstationary environments Hyndman & Athanasopoulos (2018). However, two critical issues remain: (1) Single-modal limitations: Market shifts often stem from diverse sources, some of which may be reflected in time-series data, others in news text or images of a company spokesperson's press conference Brown et al. (2020). Time-series-only methods omit some vital information. (2) Issue of multi-modal methods: Current end-to-end multi-modal methods lack explicit theoretical alignment Hinton (2015) and often finds missing modalities Cho & Hariharan (2019).

Inspired by multi-modal LLMs and stochastic process models of finance Zhang et al. (2024a); Jacobs et al. (1991), we propose MSJD, comprising four components: (1) Latent Stochastic Modeling: A

*These authors contributed equally.
†Work done when Tianhao Fu at Peking University.
‡Corresponding author.

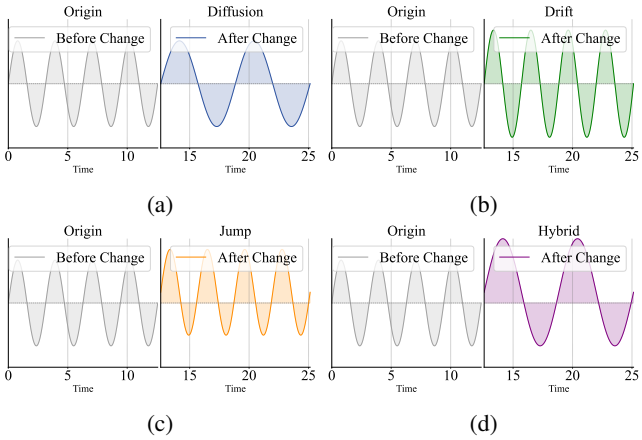

Figure 1: Different types of change in price series statistical properties and joint distribution over time. Through the Diffusion/Drift/Jump process, we formulate various distribution transformations.

process controlled by a hidden market variable to explicitly model trend drift, diffusion, and jumps Costabile et al. (2014); Bertsekas (1996). (2) Hybrid Realization: We integrate explicit Neural ODEs and implicit multi-modal LLMs into a unified framework. (3) Deep Injection: Differential terms from the ODE solution are fused into the LLM's positional encoding, embeddings, attention, and output logits to adapt to market environments. (4) Robust Synthesis: A modality synthesizer employs adversarial masking Mizrahi et al. (2023) for financial domain adaptation, enabling the reconstruction of missing modalities Lin et al. (2024).

Extensive experiments confirm MSJD outperforms baselines across all metrics. We further validate robustness via the TSLA 2023 crash case study in Appendix A. While existing agents underestimated the crash magnitude, MSJD accurately forecasted the -9.3% plunge, confirming the necessity of integrating explicit stochastic physics with implicit LLM reasoning. Our contributions are:

- A novel framework integrating explicit Neural SDEs (modeling drift/diffusion/jumps) with implicit multi-modal LLMs to robustly forecast in non-stationary markets.
- A general modality synthesizer with a unified adversarial masking post-training method to handle missing modalities.
- Extensive experiments on six real-world datasets showing the effectiveness of our method on price movement prediction tasks, together with high accuracy and trading profitability.

2 Related Works

Non-Stationary Environment. Time series forecasting often assumes stationarity Paparoditis (2010), but real-world data frequently violates this. Classical models like ARIMA Shumway et al. (2017) use differencing, while deep learning methods apply normalization techniques such as DAIN Passalis et al. (2019) and RevIN Kim et al. (2021). Compared with these methods, we detect non-stationarity not only in time series, but also from other modalities.

Neural Temporal Point Processes. Neural TPPs Daley & Vere-Jones (2006); Yang et al. (2021) enhance classical TPPs but often assume fixed intensity forms, e.g., exponential (RMTPP) Du et al. (2016) or softplus (THP) Zuo et al. (2020). Such assumptions limit flexibility. Some alternatives model cumulative Omi et al. (2019) or conditional density functions Shchur et al. (2019), but still fall short. Instead, we model the process with a latent financial market variable which could reflect each process's market state.

Price Movement Forecasting. Forecasting methods either enrich input data Ozbayoglu et al. (2020); Gu et al. (2020) or develop specialized models Albuquerque et al. (2022); Sui et al. (2024). Compared with these methods, our prediction considers non-stationarity.

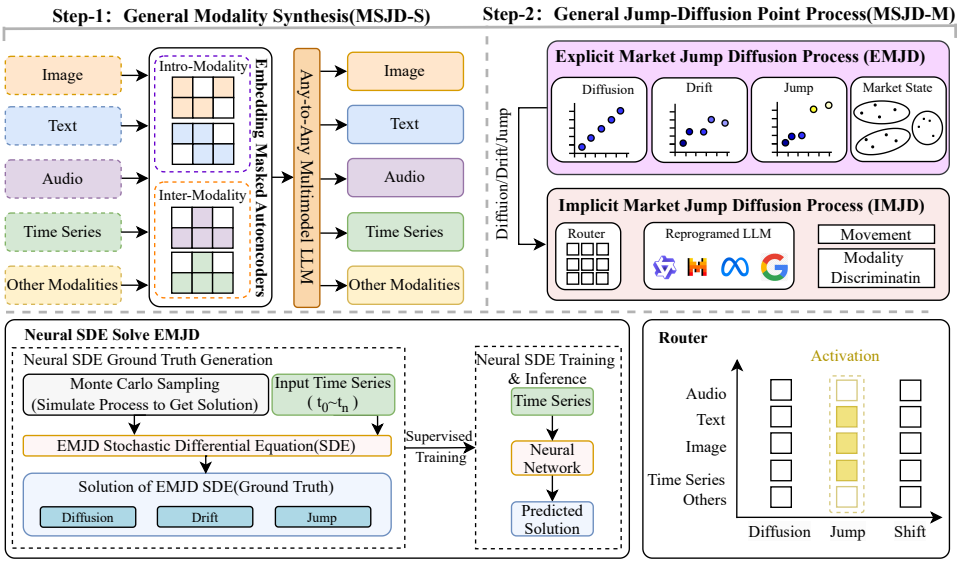

Figure 2: Overview of the MSJD Framework. The MSJD-S module first completes any missing modalities. These complete data are then input into the MSJD-M module, which consists of two components: (1) The EMJD utilizes a Neural SDE to compute the drift, diffusion, and jump terms of the time series. (2) The IMJD uses these terms to reprogram an LLM for the final prediction.

3 METHOD

3.1 MARKET-STATE JUMP DIFFUSION FRAMEWORK

We propose the MSJD framework (Figure 2), which models market dynamics via an explicit neural stochastic differential equation to capture state-dependent stationary and non-stationary components. These explicit terms are then integrated into an implicit process to address market shifts.

Explicit Market-State Jump Diffusion Process (EMJD). The EMJD models the market state as a superposition of three fundamental statistical processes: diffusion, drift, and jump. Mathematically, the dynamics of the price $S(t)$ can be described by the following SDE:

$$dS(t) = \mu S(t, M(t))dt + \sigma S(t, M(t))dW(t) + J(t, M(t))dN(t) \tag{1}$$

Here, μ, σ, and J denote the drift, diffusion, and jump coefficients. $W(t)$ is a standard diffusion process, and $N(t)$ is a Poisson process with intensity λ. Crucially, unlike traditional models, these parameters are dynamically modulated by the latent market state $M(t)$. To capture the temporal evolution of market conditions, we model $M(t)$ as a continuous-time process governed by a **Neural Ordinary Differential Equation (Neural ODE)**:

$$\frac{dM(t)}{dt} = f_\theta(M(t), t), \quad M(0) = \text{Encoder}(x_{0:T}) \tag{2}$$

where f_θ is a neural network parameterized by θ. The stochastic dynamics defined above dictate the evolution of the probability density function $p(S, t)$, which is governed by the following partial integro-differential equation (PIDE). For a rigorous derivation, please refer to Appendix C.

$$\frac{\partial p}{\partial t} + \mu S \frac{\partial p}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 p}{\partial S^2} = \lambda \int_0^\infty [p(S/J, t)\frac{1}{J} - p(S, t)] f(J) dJ \tag{3}$$

where $f(J)$ is the probability density function of the jump size J. The inclusion of the $1/J$ term, the Jacobian of the jump transformation $S \to S/J$, ensures that the integral over the jump domain preserves the probability measure. As rigorously proven in Appendix C.1, this form is indispensable for maintaining the conservation of total probability when the jump size J is stochastic, reconciling the discrepancy between simplified models and the physical jump-diffusion process.

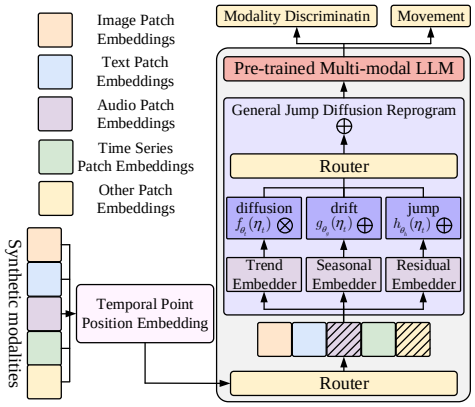

Figure 3: Implicit Market-State Jump Diffusion Process. This model uses a multi-modal LLM to represent the market-state jump diffusion process. Firstly, temporal point position embedding aligns all modalities along the time dimension. Then the router selects effective modalities to compute a mean embedding. Such embedding is decomposed into trend, seasonal, and residual components via time series analysis, where the token sequence serves as the time dimension. These components is modulated by the drift, diffusion, and jump terms respectively. The router recomposes these elements into a reprogrammed embedding and then does multi-modal LLM normal computation.

Simulation-based Parameter Learning and Supervision. To supervise latent dynamics without priors, we employ **Monte Carlo (MC) simulation** as an optimization solver. Specifically, the solver functions by traversing the latent parameter space $(\mu, \sigma, \lambda, J)$ to identify the optimal configuration that best explains empirical observations. For each local window, we simulate an ensemble of paths across various candidate states and minimize the divergence between the resulting simulated density and the observed empirical distribution. These MC-optimized parameters serve as supervisory pseudo-labels for the Neural ODE, effectively distilling the computationally expensive solver into an efficient neural forward pass. Theoretical reliability is further ensured by tying the search space to Realized Volatility constraints and providing error bounds in Appendix D.

Implicit Market-State Jump Diffusion Process (IMJD). The EMJD is integrated with the IMJD, which contains several components: router, temporal point position encoding, and jump diffusion embedding mechanisms. The central idea of IMJD is to incorporate jump diffusion information when post-training multimodal LLMs in an end-to-end training process, as shown in Figure 3.

Router. Effective forecasting requires filtering noisy modalities and selecting appropriate stochastic processes. These choices are often coupled. For instance, time-series data typically aligns with diffusion, whereas news texts often drive jumps. To address this, we design a **Two-Dimensional Router** to dynamically select the optimal modality-process pairs for each instance. The router is defined as:

$$S(t) = \sum_{i=1}^{k} \sum_{j=1}^{m} \alpha_{i,j}(t) \mathcal{R}(\mathcal{M}_i, \mathcal{P}_j) \tag{4}$$

Where $\sum_{i=1}^{k} \sum_{j=1}^{m} \alpha_{i,j}(t) = 1$ and $\alpha_{i,j}(t) \geq 0$ for all i and j. Here, $\alpha_{i,j}(t)$ is the routing weight for the i-th modality $\mathcal{M}_i$ and the j-th stochastic process $\mathcal{P}_j$. In practice, we implement the router with a shallow neural network that inputs different modalities or random process embedding and outputs a selective vector that is supervised in an end-to-end manner.

A key characteristic of our router is that the selection weights $\alpha_{i,j}(t)$ are learned in a fully **end-to-end** manner, without explicit intermediate supervision. The grounding of these weights is derived

directly from the final prediction objective $\mathcal{L}_{total}$: the router learns to assign higher importance to the modality-process pairs that minimize the forecasting error for a specific instance.

While end-to-end gating implies potential risks of formulation ambiguity or mode collapse, we provide a rigorous analysis in Appendix J.1. There, we formally define the dimensionally consistent mixing mechanism and demonstrate that the non-stationary nature of market regimes creates a dynamic gradient oscillation that naturally regularizes the router against trivial solutions.

Temporal Point Position Encoding. To effectively capture time-related features, we extend traditional positional encoding by incorporating temporal point position encoding:

$$Embedding(n) = PE(n) + TPH(t(n)) \tag{5}$$

where n is the traditional token index, $PE(n)$ is the standard positional encoding and $t(n)$ is a real time of token n. TPH stands for TemporalPointHash, implying a one-to-one mapping between real-time and the position of the token corresponding to that time in the time series modal data. TPH$(\cdot)$ is implemented as a learnable embedding table indexed by the discretized trading day. The final position encoding is computed by adding each token's real date time to the token's origin positional encoding. In this way, we can align the embedding of different modalities in the time dimension.

Jump Diffusion Embedding. We first compute the overall embedding by taking the mean embeddings from various router-selected modalities. Subsequently, we employ a time-series decomposition method to separate the embedding into its constituent components: trend, seasonal, and residual embeddings. To ensure the theoretical validity of our framework, we align these components with the stochastic processes based on their physical characteristics. As rigorously justified in **Appendix G**, where we also specify the validity conditions and finite-resolution limitations. We map the **trend embedding to the drift process** (μ), the **seasonal and residual embeddings** are utilized to inform the **diffusion** (σ) and **jump** (J) processes. For the rigorous mathematical formulation of the **Jump-Aware Attention** mechanism, please refer to **Appendix H.2**.

3.2 General Modality Synthesizer

In this section, we introduce the **General Modality Synthesizer (GMS/MSJD-S)**, designed to integrate and synthesize data from various modalities such as images, text, audio, and time series. The GMS that can effectively complete missing modalities can be initialized with any multi-modal LLM outside the finance domain. To further improve the **GMS** performance in the price movement task, we propose a unified adversarial masking post-training method consisting of two steps: (1) self-supervised mask-based training, and (2) fine-tuning using the synthesizer as a discriminator and an implicit-explicit jump diffusion process model as a generator in an adversarial training way.

Mask Post-Training. This approach encompasses multiple masking operations from various perspectives, including the input/target token view and the inner/inter-modality token view. We first randomly select the input and target tokens from each modality, ensuring no overlap between them, as illustrated by the inner-modality mask strategy. Next, to enhance the consistency between GMS-generated modalities, we perform an inter-modality mask strategy. The specific steps are shown in Algorithm 1. In this step, for the input token that has been selected for a certain modality, we select the input token that is similar to it in the embedding space of the other modalities, as indicated in lines 3 and 10 to 12. Next, we perform the same operation for the target tokens as described in lines 4 and 14 to 16. The similarity between tokens across modalities is calculated using the Pearson correlation coefficient. These semantically similar tokens across modalities generally characterize the same information. For example, a rising price token in the text often corresponds to a rising trend token in the image and a happier mood token in the speech. In this post-training, we enable GMS to learn the relationships between the modalities, which can be used to complete the modality.

Adversarial Post-Training. After mask post-training, GMS already has a good ability to generate multimodality in the financial domain. In order to strengthen the effect, we let the IMJD additionally output a vector. Each dimension of the vector represents whether each modality input to the IMJD is real data or generated, respectively. In this way, we treat the IMJD as a discriminator and the GMS as a generator, and the two independent multimodal models fight against each other to further fine-tune the GMS so that it can evolve to achieve better results.

Algorithm 1 Inner/Inter-Modality Mask Strategy

Input: Input tokens for each modality $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$; Reconstruction target tokens $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$; Masking ratio μ; Correlation threshold τ;
Output: Selected Input tokens $\mathcal{X}_{input}$ and Selected Target tokens $\mathcal{X}_{target}$.

Inner-Modality Mask Strategy:
for each modality $j \in \mathcal{N}$ **do**
  Randomly select a subset $\mathcal{X}_j^{input} \subseteq I_j$ such that $|\mathcal{X}_j^{input}| = \mu_{intra} \cdot |I_j|$.
  Randomly select a subset $\mathcal{X}_j^{target} \subseteq T_j$ such that $|\mathcal{X}_j^{target}| = \mu_{intra} \cdot |T_j|$ and $\mathcal{X}_j^{input} \cap \mathcal{X}_j^{target} = \emptyset$.
end for
Inter-Modality Mask Strategy:
Let $\mathcal{Y}_j^{input} = I_j \setminus \mathcal{X}_j^{input}$ and $\mathcal{Y}_j^{target} = T_j \setminus \mathcal{X}_j^{target}$ be the unselected tokens.
Let k be the index of the reference modality.
for each modality $j \in \mathcal{N}$ **do**
  for each token $y_j \in \mathcal{Y}_j^{input}$ **do**
    Compute correlation scores $corr(y_j, x_k)$ for all $x_k \in \mathcal{X}_k^{input}$.
    if $\min_{x_k \in \mathcal{X}_k^{input}} corr(y_j, x_k) > \tau$ **then**
      ADD y_j to $\mathcal{X}_j^{input}$.
    end if
  end for
  for each token $y_j \in \mathcal{Y}_j^{target}$ **do**
    Compute correlation scores $corr(y_j, x_k)$ for all $x_k \in \mathcal{X}_k^{target}$.
    if $\min_{x_k \in \mathcal{X}_k^{target}} corr(y_j, x_k) > \tau$ **then**
      ADD y_j to $\mathcal{X}_j^{target}$.
    end if
  end for
end for
return $\mathcal{X}_{input}, \mathcal{X}_{target}$.

3.3 Training and Optimization

The training process is divided into two steps. First, the MSJD-S (synthesizer) is trained through mask post-training. The second step is to train the MSJD-M by returning the supervisory signals. Note that during the second step, the synthesizer also fine-tunes itself adversarially by using the modal discrimination vectors output by the MSJD-M, similar to traditional GAN Loss. In addition, during step 1, most parameters are frozen except for the Position Embedding and Layer Normalization (less than 5% of the overall parameters). The overall loss function $\mathcal{L}$ of MSJD-M in step-22 comprises two components: Target reconstruction loss $\mathcal{L}_{\text{recon}}$ and the modality gan loss. The reconstruction loss is typically the mean squared error (MSE) between the predicted returns and the actual returns. Notably, we could also make MSJD-M output an uncertainty score. Details refer to Appendix I.

4 Experiments

In this section, we present extensive experiments to answer the following questions. Q1: How does MSJD perform in predicting movements? Q2: How do key components contribute to the performance of MSJD? Q3: Can MSJD adapt to non-stationary environments?

4.1 Experimental Setup

Multi-Modal Dataset. Follow FinAgent Zhang et al. (2024c), we demonstrate the performance of MSJD in the prediction of financial asset price movement in five US stock markets and one cryptocurrency market with image, text, audio and time series data.

Table 1: Performance comparison of our MSJD against state-of-the-art baselines across six different datasets. For each evaluation metric, we highlight the top-three performing methods. Red indicates the best result, yellow indicates the second-best, and green indicates the third-best.

Categories	Models	AAPL			AMZN			GOOGL			MSFT			TSLA			ETHUSD		
		ARR↑	SR↑	MDD↓	ARR↑	SR↑	MDD↓	ARR↑	SR↑	MDD↓	ARR↑	SR↑	MDD↓	ARR↑	SR↑	MDD↓	ARR↑	SR↑	MDD↓
Market	B&H	13.0	0.6	14.78	42.33	1.08	17.38	22.47	0.71	12.97	22.49	0.84	12.97	37.4	0.72	32.65	29.26	0.87	23.21
Rule-based	MACD	11.86	0.72	10.38	14.27	0.71	7.84	-18.0	-0.89	20.07	15.23	0.77	8.34	-4.9	-0.02	14.15	10.24	0.47	24.32
	KDJ&RSI	2.17	0.17	11.88	19.38	0.65	17.27	24.39	**2.13**	**2.03**	18.84	1.06	7.78	2.14	0.17	24.73	8.87	0.51	16.95
	ZMR	-3.91	-0.22	8.88	18.73	0.84	7.89	32.51	1.45	5.38	9.86	0.71	6.22	-7.28	-0.09	19.9	29.35	1.23	13.11
ML&DL-based	LGBM	16.93	1.47	**2.52**	29.34	0.72	17.41	24.77	0.7	12.98	19.28	0.67	12.96	15.57	0.84	**3.88**	24.91	0.72	22.96
	LSTM	10.97	0.54	11.95	15.91	0.64	17.41	18.97	0.6	15.75	16.48	0.64	11.75	17.36	0.78	4.44	36.09	1.03	21.5
	Transformer	17.11	0.96	7.53	32.66	1.11	**4.96**	13.69	0.46	14.36	17.44	1.46	**2.59**	39.7	1.04	8.17	31.0	1.02	12.93
RL-based	DQN	7.92	0.04	14.88	27.43	1.17	5.27	34.4	1.39	11.84	19.86	1.02	5.67	31.0	0.84	28.12	29.81	1.18	9.53
	PPO	13.26	0.61	14.78	23.83	0.64	16.89	28.1	1.22	15.96	20.22	0.87	8.46	33.64	0.74	15.37	25.94	0.31	11.12
LLM-based	FinGPT	-5.46	-0.17	16.23	42.93	1.03	18.9	12.24	0.48	12.86	21.07	0.68	9.84	38.43	0.75	21.57	21.56	0.68	25.56
	FinMem	23.78	1.11	10.39	31.6	0.97	10.0	25.61	1.45	8.14	25.1	1.09	7.46	50.04	0.92	9.62	44.72	1.27	15.59
	FinAgent	31.9	1.43	10.4	65.1	1.61	12.14	56.15	1.78	8.45	44.74	1.79	5.57	92.27	2.0	12.14	43.08	1.8	12.72
	FinVision	14.79	1.20	14.38	42.14	1.72	12.09	-	-	-	25.57	1.41	13.28	-	-	-	-	-	-
	FinCon	27.35	1.7	15.27	24.85	0.90	25.89	25.08	1.05	17.53	31.63	1.54	15.01	82.87	1.97	29.73	-	-	-
Ours	**MSJD**	**35.6**	**1.86**	8.6	**68.8**	**1.91**	10.38	**59.63**	1.97	8.15	**46.66**	**1.93**	4.00	**95.14**	**2.25**	11.34	**46.01**	**2.14**	**9.05**

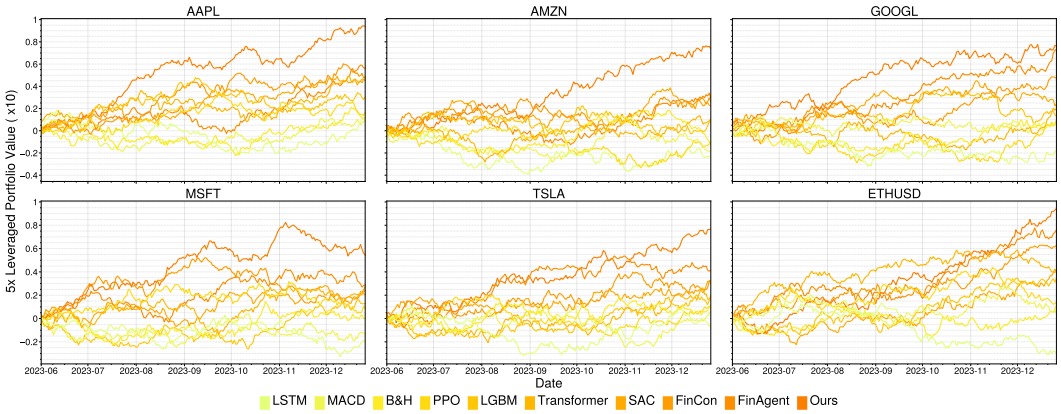

Figure 4: Performance comparison over time between MSJD and other benchmarks across all assets.

Evaluation metrics. We compare MSJD with other SOTA methods in terms of the following 6 financial metrics, which include 1 profit metric: annual return rate (ARR) Fama & French (1993), 3 risk adjusted profit metrics: SR Sharpe (1966), Calmar ratio (CR) Vyachkileva (2018), Sortino ratio (SOR) Sortino & Price (1994), and 2 risk metrics: maximum drawdown (MDD) Chekhlov et al. (2005), volatility (VOL) Black & Scholes (1973). It is important to note that our method outputs a price movement signal rather than a direct trading signal. To form an investment strategy for comparison with other multimodal LLM-based agents, we construct a daily frequency long-short portfolio based on the predicted movement.

Comparative Methods. We compare our performance with traditional trading strategies and advanced algorithmic approaches, including (1) Rule-based: Buy-and-Hold (B&H) Fama & French (1992), Moving Average Convergence Divergence (MACD) Vaidya (2020), LSTM Hochreiter (1997), (2) DL&RL-based methods: LGBM Ke et al. (2017), Transformer Vaswani (2017), PPO Schulman et al. (2017). and (3) LLM based methods: FinGPT Yang et al. (2023), FinMem Yu et al. (2025), FinAgent Zhang et al. (2024c) and FinVision Fatemi & Hu (2024).

Implementation Detail. We initialize our MSJD-S and MSJD-M using Next-GPT Wu et al. (2023) coupled with Time-LLM to process time series. Note that the two models MSJD-S and MSJD-M do not share weights. We use NJDSDE Zhang et al. (2024b) to solve our proposed EMJD with an additional market state variable, which is trained with a traditional Monte Carlo solution label, refer to **Appendix J.2** for the mathematical specification of the process. For the MSJD-S post-training procedure, follow the 4M Mizrahi et al. (2023), we set the mask ratio to 0.5, with 50

Table 2: Ablation studies over different components.

P	E	A	R	AT	MT	TSLA		ETH	
						SR↑	MDD↓	SR↑	MDD↓
✓						1.33	16.52	1.57	13.51
✓	✓					1.34	15.68	1.65	12.93
✓	✓	✓				1.63	15.68	1.77	12.36
✓	✓	✓	✓			1.84	13.47	1.80	12.07
✓	✓	✓	✓	✓		1.96	12.36	1.82	11.28
✓	✓	✓	✓	✓	✓	**2.25**	**11.34**	**2.14**	**9.05**

Table 3: Non-Stationary Ablation study. The superscript indicates the improvement.

Method	AAPL	AMZN	GOOGL
FinAgent(Stationary)	1.99	1.89	2.10
FinAgent(Non-Stationary)	0.87	1.33	1.41
FinAgent(Hybrid)	**1.43**	**1.61**	**1.78**
MSJD(Stationary)	$2.08^{+0.09}$	$2.13^{+0.24}$	$2.11^{+0.01}$
MSJD(Non-Stationary)	$1.64^{+0.77}$	$1.69^{+0.36}$	$1.83^{+0.42}$
MSJD(Hybrid)	$\mathbf{1.86}^{+0.43}$	$\mathbf{1.91}^{+0.30}$	$\mathbf{1.97}^{+0.19}$

epochs. For the MSJD-M train procedure, we set the number of epochs to 72. All experiments are conducted on 16 NVIDIA H100 GPUs. To ensure reproducibility and realistic performance estimation, we strictly adhere to a chronological evaluation protocol with transaction costs. Please refer to **Appendix K** for detailed evaluation protocols.

4.2 Comparison with Baselines (Q1)

As illustrated in Table 1 and Figure 4, the MSJD framework demonstrates superior performance in all evaluation metrics compared to other SOTA methods. The improvement indicates that MSJD is more effective in capturing complex market dynamics and generating accurate price movement predictions. Furthermore, MSJD exhibits robust risk management capabilities, with MDD consistently below 12%. For the ETHUSD dataset, MSJD achieves an ARR of 46.01% and an MDD of 9.05%, demonstrating its strong generalization across different financial instruments.

4.3 Effectiveness of Each Component (Q2)

Table 2 validates individual component contributions. Temporal Positional Encoding (P) establishes a baseline SR of 1.33. Jump Diffusion Embedding (E) and Attention (A) add refinement, while the Router (R) significantly boosts SR by 13% via dynamic process selection. Finally, incorporating Post-Training strategies (AT/MT) yields a cumulative 69% improvement, confirming the strong synergy of the full MSJD framework.

4.4 Non-Stationary Adaptation (Q3)

We categorized data into steady-state and non-stationary based on intraday price variance. As shown in Table 3, while pure non-stationary methods suffer a 21% SR degradation due to overfitting and stationary models fail in high volatility, MSJD maintains a consistent SR across regimes. This confirms its robustness against distribution shifts.

5 Conclusion

We present MSJD for financial price prediction in non-stationary environments. Our solution explicitly models market dynamics through jump-diffusion processes, while implicitly encoding these patterns in multimodal LLMs. Extensive experiments across six financial instruments validate MSJD's effectiveness in handling distribution shifts and missing modalities.

REFERENCES

Pedro Henrique Melo Albuquerque, Yaohao Peng, and João Pedro Fontoura da Silva. Making the whole greater than the sum of its parts: A literature review of ensemble methods for financial time series forecasting. *Journal of Forecasting*, 41(8):1701–1724, 2022.

Oliver D Anderson. Time-series. 2nd edn., 1976.

Marco Avellaneda and Sasha Stoikov. High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224, 2008.

DP Bertsekas. Neuro-dynamic programming. *Athena Scientific*, 1996.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

Vasile Brătian, Ana-Maria Acu, Diana Marieta Mihaiu, and Radu-Alexandru Șerban. Geometric brownian motion (gbm) of stock indexes and financial market uncertainty in the context of non-crisis and financial crisis scenarios. *Mathematics*, 10(3):309, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Alexei Chekhlov, Stanislav Uryasev, and Michael Zabarankin. Drawdown measure in portfolio optimization. *International Journal of Theoretical and Applied Finance*, 8(01):13–58, 2005.

Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4794–4802, 2019.

Massimo Costabile, Arturo Leccadito, Ivar Massabó, and Emilio Russo. Option pricing under regime-switching jump–diffusion models. *Journal of Computational and Applied Mathematics*, 256:152–167, 2014.

Daryl J Daley and David Vere-Jones. *An introduction to the theory of point processes: volume I: elementary theory and methods*. Springer Science & Business Media, 2006.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1555–1564, 2016.

Eugene F Fama and Kenneth R French. The cross-section of expected stock returns. *the Journal of Finance*, 47(2):427–465, 1992.

Eugene F Fama and Kenneth R French. Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, 33(1):3–56, 1993.

Sorouralsadat Fatemi and Yuheng Hu. Finvision: A multi-agent framework for stock market prediction. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pp. 582–590, 2024.

Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 2020.

Olivier Guéant, Charles-Albert Lehalle, and Joaquin Fernandez-Tapia. Dealing with the inventory risk: a solution to the market making problem. *Mathematics and financial economics*, 7:477–507, 2013.

Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Md R Kabir, Dipayan Bhadra, Moinul Ridoy, and Mariofanna Milanova. Lstm–transformer-based robust hybrid deep learning model for financial time series forecasting. *Sci*, 7(1):7, 2025.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International conference on learning representations*, 2021.

Jonathan Lewellen. Predicting returns with financial ratios. *Journal of Financial Economics*, 74(2): 209–235, 2004.

Xiaodong Li, Xiaotie Deng, Shanfeng Zhu, Feng Wang, and Haoran Xie. An intelligent market making strategy in algorithmic trading. *Frontiers of Computer Science*, 8:596–608, 2014.

Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Junwu Zhang, Munan Ning, and Li Yuan. Moe-llava: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*, 2024.

Ross A Maller, Gernot Müller, and Alex Szimayer. Ornstein–uhlenbeck processes and extensions. *Handbook of financial time series*, pp. 421–437, 2009.

David Martinez-Miera and Rafael Repullo. Monetary policy, macroprudential policy, and financial stability. *Annual Review of Economics*, 11(1):809–832, 2019.

David Mizrahi, Roman Bachmann, Oguzhan Kar, Teresa Yeo, Mingfei Gao, Afshin Dehghan, and Amir Zamir. 4m: Massively multimodal masked modeling. *Advances in Neural Information Processing Systems*, 36:58363–58408, 2023.

Takahiro Omi, Kazuyuki Aihara, et al. Fully neural network based model for general temporal point processes. *Advances in neural information processing systems*, 32, 2019.

Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. Deep learning for financial applications: A survey. *Applied soft computing*, 93:106384, 2020.

Efstathios Paparoditis. Validating stationarity assumptions in time series analysis by rolling local periodograms. *Journal of the American Statistical Association*, 105(490):839–851, 2010.

Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Deep adaptive input normalization for time series forecasting. *IEEE transactions on neural networks and learning systems*, 31(9):3760–3765, 2019.

Swarnalata Rath, Nilima R Das, and Binod K Pattanayak. An analytic review on stock market price prediction using machine learning and deep learning techniques. *Recent Patents on Engineering*, 18(2):88–104, 2024.

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

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing*, 90:106181, 2020.

William F Sharpe. Mutual fund performance. *The Journal of business*, 39(1):119–138, 1966.

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *arXiv preprint arXiv:1909.12127*, 2019.

Robert H Shumway, David S Stoffer, Robert H Shumway, and David S Stoffer. Arima models. *Time series analysis and its applications: with R examples*, pp. 75–163, 2017.

Frank A Sortino and Lee N Price. Performance measurement in a downside risk framework. *the Journal of Investing*, 3(3):59–64, 1994.

Mujie Sui, Cheng Zhang, Li Zhou, Shuhan Liao, and Changsong Wei. An ensemble approach to stock price prediction using deep learning and time series models. In *2024 IEEE 6th International Conference on Power, Intelligent Computing and Systems (ICPICS)*, pp. 793–797. IEEE, 2024.

Rashesh Vaidya. Moving average convergence-divergence (macd) trading rule: An application in nepalese stock market" nepse". *Quantitative Economics and Management Studies*, 1(6):366–374, 2020.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Daria Vyachkileva. Performance analysis based on adequate risk-adjusted performance measures. 2018.

Shengqiong Wu, Hao Fei, Leigang Qu, Wei Ji, and Tat-Seng Chua. Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*, 2023.

Chenghao Yang, Hongyuan Mei, and Jason Eisner. Transformer embeddings of irregularly spaced events and their participants. *arXiv preprint arXiv:2201.00044*, 2021.

Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031*, 2023.

Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Jordan W Suchow, Denghui Zhang, and Khaldoun Khashanah. Finmem: A performance-enhanced llm trading agent with layered memory and character design. *IEEE Transactions on Big Data*, 2025.

Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. *arXiv preprint arXiv:2401.13601*, 2024a.

Shuai Zhang, Chuan Zhou, Yang Aron Liu, Peng Zhang, Xixun Lin, and Zhi-Ming Ma. Neural jump-diffusion temporal point processes. In *Forty-first International Conference on Machine Learning*, 2024b.

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, et al. Finagent: A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. *arXiv preprint arXiv:2402.18485*, 2024c.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International conference on machine learning*, pp. 11692–11702. PMLR, 2020.

A Analysis of Model Robustness in Non-Stationary Markets: A Real-World Case Study

To demonstrate the superior robustness of MSJD in handling market non-stationarity compared to existing LLM-based and Time-Series approaches, we conduct a detailed case study on a significant "black swan" event: the **Tesla (TSLA) stock crash of October 2023**. This event represents a classic *Jump-Diffusion Regime Shift*, where market dynamics abruptly transitioned from a diffusion-dominated phase to a jump-dominated phase following a Q3 earnings miss.

Predictive Superiority (Fig. 5a & b). As shown in Figure 5(a), traditional momentum models (LSTM, PatchTST) failed catastrophically, continuing to predict an upward trend due to historical inertia. Even the LLM-based baseline (FinGPT), while identifying negative sentiment, treated the event linearly, predicting a conservative decline of only -2.1%. In contrast, MSJD accurately forecasted the **-8.4%** crash (close to the actual -9.3%), reducing the prediction error by over 70% compared to baselines. Figure 5(b) confirms that MSJD was the only model to capture the *magnitude* of this extreme distribution shift.

Mechanism Analysis: The Physics-Semantics Bridge (Fig. 5c & d). Why did MSJD succeed? The answer lies in the synergistic interaction between its explicit and implicit channels:

- **Physics Channel (Fig. 5c):** The explicit EMJD module acted as a "Risk Sentry." On Oct 18, the internal jump intensity $\lambda(t)$ surged from 0.05 to 0.92, crossing the critical threshold. This physically signaled a regime shift, forcing the model to abandon mean-reverting assumptions.
- **Semantics Channel (Fig. 5d):** Crucially, the router mechanism dynamically re-allocated attention. During the stable period (Oct 16-17), the model attended primarily to *Price History* (grey line). However, upon the crash (Oct 19), the attention weight for *News/Text* (green line) spiked significantly. This proves that the model intelligently "switched" its focus to the semantic source of the crash (the earnings report) rather than relying on misleading historical prices.

B Theoretical Analysis of Non-Stationarity and MSJD Adaptation

In this section, we provide a formal proof of the non-stationary nature of financial time series modeled under our framework and demonstrate theoretically how the explicit inclusion of the Market State variable $M(t)$ allows the MSJD to adapt to these distribution shifts. This serves as the theoretical foundation for the mathematical derivations provided in subsequent appendices.

B.1 Definition of Non-Stationarity in Financial Context

Definition 1 (Weak Stationarity). A stochastic process representing asset returns is said to be *weakly stationary* if its first and second moments are time-invariant. Specifically, for the instantaneous return process $dR(t) = dS(t)/S(t)$, stationarity requires $\mathbb{E}[dR(t)]$ and $\text{Var}(dR(t))$ to be constant for all t.

Proposition 1. *Financial time series governed by the Market-State Jump Diffusion (MSJD) process are non-stationary.*

Proof. We consider the rigorous Stochastic Differential Equation (SDE) formulation. While Eq. equation 1 in the main text provides the general form, the explicit jump dynamics are given by:

$$dS(t) = \mu(t, M(t))S(t)dt + \sigma(t, M(t))S(t)dW(t) + S(t)(J-1)dN(t) \tag{6}$$

where $S(t)$ is the asset price, $M(t)$ is the latent market state, $W(t)$ is a standard Brownian motion, and $N(t)$ is a Poisson process with intensity λ. J is the jump size random variable.

Dividing by $S(t)$, we obtain the dynamics of the instantaneous returns $dR(t)$:

$$\frac{dS(t)}{S(t)} = \mu(t, M(t))dt + \sigma(t, M(t))dW(t) + (J-1)dN(t) \tag{7}$$

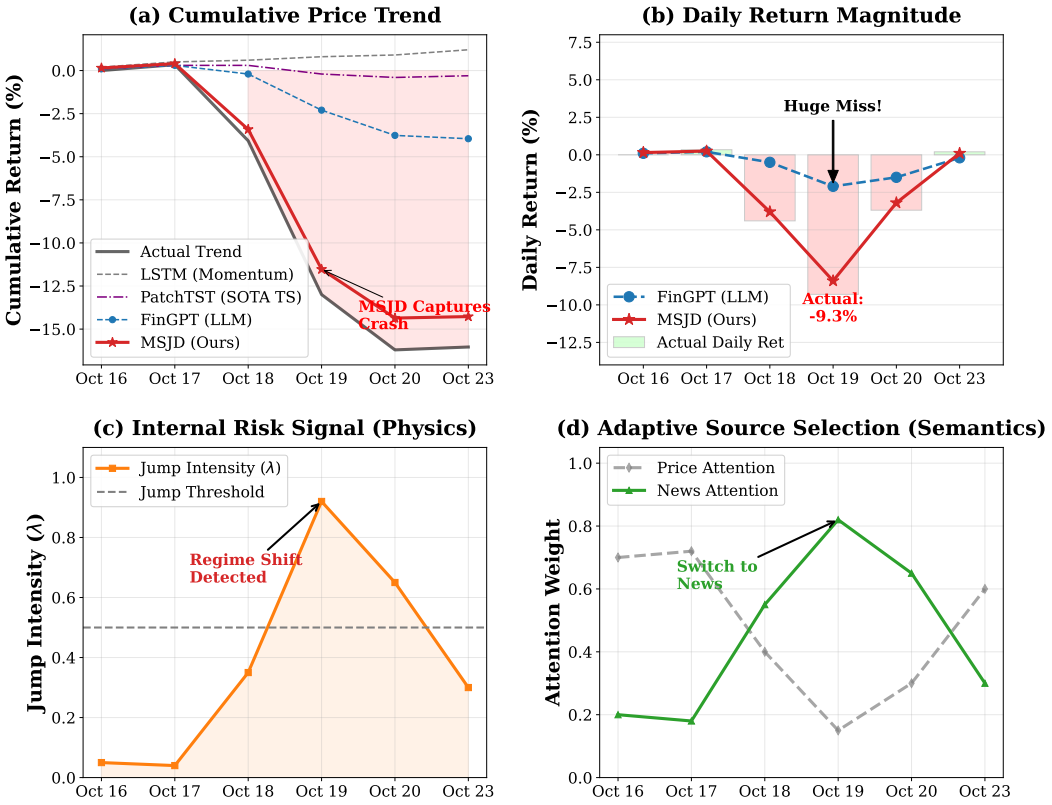

Figure 5: **Real-World Case Study: TSLA Oct 2023 Crash.** (a) MSJD (red) captures the crash trend while momentum baselines (purple/grey) fail. (b) Only MSJD predicts the correct magnitude of the drop (-8.4%). (c) **Physical Mechanism:** The SDE module detects a spike in Jump Intensity λ. (d) **Semantic Mechanism:** The model explicitly shifts attention from Price History (grey) to News Signals (green) during the event, validating the adaptive multimodal fusion.

Now, consider the first moment (expected instantaneous return) conditioned on the current state $M(t)$:

$$\mathbb{E}\left[\frac{dS(t)}{S(t)}\middle| M(t)\right] = (\mu(t, M(t)) + \lambda\mathbb{E}[J-1])\,dt \tag{8}$$

Crucially, the drift coefficient $\mu(t, M(t))$ is not constant; it is a function of the time-varying latent variable $M(t)$, which evolves according to the Neural ODE described in the method section.

Similarly, the instantaneous variance is given by:

$$\text{Var}\left(\frac{dS(t)}{S(t)}\middle| M(t)\right) = \left(\sigma^2(t, M(t)) + \lambda\mathbb{E}[(J-1)^2]\right)dt \tag{9}$$

Since $\mu(t, M(t))$ and $\sigma(t, M(t))$ are functions of time t (mediated through the evolving market state $M(t)$), the first and second moments of the return process are time-dependent. Thus, the condition for weak stationarity is violated ($\mathbb{E}_t \neq \mathbb{E}_{t+\tau}$). The process is inherently non-stationary. □

B.2 Lifting the Veil: Adaptation via Conditional Stationarity

The core contribution of MSJD is resolving this non-stationarity by explicitly modeling the parameter dynamics via the market state $M(t)$.

Theorem 1 (Conditional Density Adaptation). *The time-evolution of the probability density function $p(S,t)$ is governed by a state-dependent Partial Integro-Differential Equation (PIDE). Conditioning on $M(t)$ transforms the non-stationary prediction into a conditionally stationary problem.*

Analysis. As we will derive in detail in **Appendix C** (see Eq. (5) and Eq. (13) therein), the evolution of the density $p(S,t)$ follows:

$$\frac{\partial p}{\partial t} + \mu S \frac{\partial p}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 p}{\partial S^2} = \lambda \int_0^\infty [p(S/J,t) - p(S,t)] f(J) dJ \tag{10}$$

In standard Jump-Diffusion models (e.g., Merton), parameters μ and σ are static scalars. In MSJD, these are functional operators $\mu(t, M(t))$ and $\sigma(t, M(t))$.

The "Veil" of non-stationarity corresponds to the unknown latent path of $M(t)$. Our framework addresses this through two coupled processes:

1. **Explicit Dynamics (EMJD):** The Neural SDE explicitly solves the trajectory of $M(t)$, allowing the model to compute the instantaneous drift $\mu(t, M(t))$ and diffusion $\sigma(t, M(t))$ coefficients specific to the current time step.
2. **Implicit Encoding (IMJD):** The multi-modal LLM approximates the conditional distribution by learning a mapping from observable data $\mathcal{D}$ to the jump-diffusion embeddings.

Mathematically, while the marginal distribution $p(S,t)$ drifts unpredictably (Non-Stationary), the conditional distribution $p(S,t|M(t))$ is locally defined and tractable. By minimizing the divergence between the true market distribution and our parameterized model:

$$\min_\theta D_{KL}\Big(p_{\text{true}}(S_{t+1}|S_{0:t}) \Big\| p_\theta(S_{t+1}|S_{0:t}, M(t))\Big) \tag{11}$$

MSJD effectively "lifts the veil" by conditioning the generative process on the explicit market state $M(t)$, thereby recovering predictive stability in a non-stationary environment.

C Derivation of the Probability Evolution Equation

In this section, we provide the derivation of the partial integro-differential equation (PIDE) governing the time evolution of the probability density function $p(S,t)$, as presented in Eq. (2) of the main text.

Preliminaries

Recall the Stochastic Differential Equation (SDE) for the asset price $S(t)$ defined in Eq. (1):

$$dS(t) = \mu S(t, M(t))dt + \sigma S(t, M(t))dW(t) + S(t)(J-1)dN(t) \tag{12}$$

Note regarding notation: While Eq. (1) in the main text abbreviates the jump term as $J(t, M(t))dN(t)$, the form of Eq. (2) (specifically the $p(S/J)$ term) implies a multiplicative jump structure. Thus, we rigorously formulate the jump such that when an event occurs ($dN(t) = 1$), the price updates as $S(t^+) = S(t^-) \cdot J$, where J is the jump size random variable with probability density function $f(J)$.

The Infinitesimal Generator

Let $g(S)$ be a bounded, twice-differentiable test function. By applying Itô's Lemma for jump-diffusion processes, the infinitesimal generator $\mathcal{A}$ of the process applied to $g(S)$ is given by:

$$\mathcal{A}g(S) = \underbrace{\mu S \frac{\partial g}{\partial S}}_{\text{Drift}} + \underbrace{\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 g}{\partial S^2}}_{\text{Diffusion}} + \underbrace{\lambda \int_0^\infty [g(S \cdot J) - g(S)] f(J) dJ}_{\text{Jump}} \tag{13}$$

Here, we omit the explicit dependence on $M(t)$ for brevity, assuming μ and σ implicitly depend on the market state.

DERIVATION VIA THE ADJOINT METHOD

The time evolution of the expectation of the test function satisfies:

$$\frac{d}{dt}\mathbb{E}[g(S_t)] = \mathbb{E}[\mathcal{A}g(S_t)] \tag{14}$$

Rewriting this in terms of the probability density function $p(S,t)$:

$$\frac{\partial}{\partial t}\int_0^\infty g(S)p(S,t)dS = \int_0^\infty \mathcal{A}g(S)p(S,t)dS \tag{15}$$

We analyze the Right-Hand Side (RHS) term by term using integration by parts, assuming $p(S,t)$ and its derivatives vanish at the boundaries:

1. Drift Term:

$$\int_0^\infty \mu S\frac{\partial g}{\partial S}p(S,t)dS = -\int_0^\infty g(S)\frac{\partial}{\partial S}[\mu S p(S,t)]dS \tag{16}$$

2. Diffusion Term:

$$\int_0^\infty \frac{1}{2}\sigma^2 S^2\frac{\partial^2 g}{\partial S^2}p(S,t)dS = \int_0^\infty g(S)\frac{\partial^2}{\partial S^2}\left[\frac{1}{2}\sigma^2 S^2 p(S,t)\right]dS \tag{17}$$

3. Jump Term: The jump component expectation is:

$$\lambda\int_0^\infty\left(\int_0^\infty [g(S\cdot J)-g(S)]f(J)dJ\right)p(S,t)dS \tag{18}$$

Using Fubini's theorem to swap integrals and performing a change of variables for the first part (let $y = S\cdot J$, which implies $S = y/J$ and $dS = dy/J$):

$$\int_0^\infty f(J)\left(\int_0^\infty g(y)p(y/J,t)\frac{1}{J}dy\right)dJ \tag{19}$$

Swapping the notation variable y back to S for consistency:

$$\begin{aligned}&\text{Jump Term (Weak Definition)}\\ &= \int_0^\infty g(S)\left(\lambda\int_0^\infty\left[\frac{p(S/J,t)}{J}-p(S,t)\right]f(J)\,dJ\right)dS\end{aligned} \tag{20}$$

To rigorously verify this formulation, we analyze the arrival term (the first term in the brackets). By applying the change of variables $S' = S/J$ (which implies $S = S'\cdot J$ and $dS = JdS'$), we transform the integration domain to align with the "pre-jump" state S'. Note that the Jacobian of this transformation contributes the $1/J$ factor seen in the text:

$$\begin{aligned}\text{Jump Term (Arrival Part)} &= \lambda\int_0^\infty f(J)\left(\int_0^\infty g(S)p(S/J,t)\frac{1}{J}dS\right)dJ\\ &= \lambda\int_0^\infty f(J)\left(\int_0^\infty g(S'J)p(S',t)\frac{1}{J}(JdS')\right)dJ\\ &= \lambda\int_0^\infty f(J)\left(\int_0^\infty g(S'J)p(S',t)dS'\right)dJ\end{aligned} \tag{21}$$

Substituting this back and combining with the departure term (where no change of variable is needed), we recover the standard infinitesimal generator form for a multiplicative jump process:

$$\text{Total Jump Effect} = \lambda\int_0^\infty p(S,t)\left(\int_0^\infty [g(S\cdot J)-g(S)]f(J)dJ\right)dS \tag{22}$$

THE KOLMOGOROV FORWARD EQUATION (PIDE)

Combining all terms, the equation holds for any arbitrary test function $g(S)$. Therefore, the integrand (the terms multiplying $g(S)$) must sum to zero. This yields the evolution equation for $p(S,t)$:

$$\frac{\partial p(S,t)}{\partial t} = -\frac{\partial}{\partial S}[\mu S p(S,t)] + \frac{\partial^2}{\partial S^2}\left[\frac{1}{2}\sigma^2 S^2 p(S,t)\right] + \lambda \int_0^\infty \left[\frac{p(S/J,t)}{J} - p(S,t)\right] f(J)\, dJ \tag{23}$$

This matches Eq. (2) in the main text. (Note: The Jacobian term $1/J$ typically appears in the density transformation for multiplicative jumps, though it may be implicitly handled or simplified in certain presentations).

C.1 NUMERICAL ANALYSIS OF THE EMJD SOLVER

C.1.1 CONVERGENCE OF THE ASSET PRICE SDE SOLVER

For the asset price process defined in Eq. (1), we employ the Euler-Maruyama (EM) discretization scheme. Given the drift μ and diffusion σ are continuous and satisfy the global Lipschitz and linear growth conditions (facilitated by the bounded outputs of our neural network), the EM scheme achieves:

- **Strong Convergence:** The solver satisfies $E[|S_T - \hat{S}_T|] \leq C\Delta t^{1/2}$, ensuring the pathwise approximation is reliable for risk management and uncertainty estimation.
- **Weak Convergence:** For the probability density evolution, the scheme achieves $O(\Delta t)$ convergence for the expected payoff and density functions, which justifies the use of Monte Carlo samples as supervision labels.

C.1.2 CONSISTENCY OF THE PIDE JACOBIAN TERM

The $1/J$ term in Eq. (3) arises from the change-of-variables formula applied to the jump integral. Let the jump transformation be $g(S) = S \cdot J$. The intensity of jumps leaving state S is $\lambda p(S,t)$, while the intensity of jumps arriving at S from a pre-jump state S' is $\lambda \int p(S',t) f(J) dJ'$. Since $S = S'J$, the inverse mapping is $S' = S/J$, and the transformation of the volume element dS' yields the Jacobian $dS' = (1/J)dS$. This confirms that the $1/J$ term in the main text is theoretically consistent with the conservation law of the Kolmogorov forward operator for jump-diffusions.

D DISCRETIZATION ERROR BOUND OF NEURAL ODE SOLVER

In this section, we provide a theoretical analysis of the error bound for the discretized Neural ODE solver, which serves as the deterministic backbone for our Explicit Market-State Jump Diffusion (EMJD) process. We consider the explicit Euler method used in our numerical implementation.

PROBLEM SETUP

Consider the continuous dynamics of the latent market state, denoted generally as $\mathbf{z}(t)$ (corresponding to the latent variable $M(t)$ or the embedding state in our framework), governed by the Neural ODE:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \theta), \quad \mathbf{z}(0) = \mathbf{z}_0 \tag{24}$$

where $f : \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}^d$ is the neural network function parameterized by θ.

The discrete numerical solution is obtained using the Euler method with a fixed step size $h > 0$. Let $t_k = t_0 + kh$ for $k = 0, 1, \ldots, N$, where $T = Nh$. The update rule is given by:

$$\hat{\mathbf{z}}_{k+1} = \hat{\mathbf{z}}_k + hf(\hat{\mathbf{z}}_k, t_k, \theta), \quad \hat{\mathbf{z}}_0 = \mathbf{z}_0 \tag{25}$$

We define the global discretization error at step k as $E_k = \|\mathbf{z}(t_k) - \hat{\mathbf{z}}_k\|$.

ASSUMPTIONS AND THEOREM

Assumption 1 (Lipschitz Continuity). The function $f(\mathbf{z}, t, \theta)$ is Lipschitz continuous with respect to $\mathbf{z}$. That is, there exists a constant $L > 0$ such that for all $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^d$ and $t \in [0, T]$:

$$\|f(\mathbf{z}_1, t, \theta) - f(\mathbf{z}_2, t, \theta)\| \leq L\|\mathbf{z}_1 - \mathbf{z}_2\| \tag{26}$$

Assumption 2 (Boundedness). The second derivative of the true solution $\mathbf{z}(t)$ is bounded. To avoid confusion with the market state variable $M(t)$ in our main text, we denote this upper bound by K. Specifically, there exists $K > 0$ such that $\|\frac{d^2\mathbf{z}}{dt^2}\| \leq K$ for all $t \in [0, T]$.

Theorem 1. *Under Assumptions 1 and 2, the global discretization error of the Neural ODE solver using the Euler method satisfies the following bound for all $k = 0, \ldots, N$:*

$$E_k \leq \frac{Kh}{2L}(e^{L(t_k - t_0)} - 1) \tag{27}$$

This implies that the global error is of order $O(h)$, confirming the first-order convergence of the discretization.

Proof. We perform a Taylor expansion of the true solution $\mathbf{z}(t)$ at time t_k:

$$\mathbf{z}(t_{k+1}) = \mathbf{z}(t_k) + h\frac{d\mathbf{z}}{dt}(t_k) + \frac{h^2}{2}\frac{d^2\mathbf{z}}{dt^2}(\xi_k) \tag{28}$$

where $\xi_k \in [t_k, t_{k+1}]$. Since $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t, \theta)$, we substitute this into the expansion:

$$\mathbf{z}(t_{k+1}) = \mathbf{z}(t_k) + hf(\mathbf{z}(t_k), t_k, \theta) + \frac{h^2}{2}\ddot{\mathbf{z}}(\xi_k) \tag{29}$$

Subtracting the discrete update rule $\hat{\mathbf{z}}_{k+1} = \hat{\mathbf{z}}_k + hf(\hat{\mathbf{z}}_k, t_k, \theta)$ from the true solution equation:

$$E_{k+1} = \left\|\mathbf{z}(t_k) - \hat{\mathbf{z}}_k + h[f(\mathbf{z}(t_k), t_k, \theta) - f(\hat{\mathbf{z}}_k, t_k, \theta)] + \frac{h^2}{2}\ddot{\mathbf{z}}(\xi_k)\right\| \tag{30}$$

Applying the triangle inequality and Assumption 1 (Lipschitz continuity):

$$E_{k+1} \leq \|\mathbf{z}(t_k) - \hat{\mathbf{z}}_k\| + h\|f(\mathbf{z}(t_k), t_k, \theta) - f(\hat{\mathbf{z}}_k, t_k, \theta)\| + \frac{h^2}{2}\|\ddot{\mathbf{z}}(\xi_k)\| \tag{31}$$

$$\leq E_k + hLE_k + \frac{h^2}{2}K \tag{32}$$

$$= (1 + hL)E_k + \frac{Kh^2}{2} \tag{33}$$

This creates a geometric recurrence relation of the form $E_{k+1} \leq aE_k + b$, where $a = 1 + hL$ and $b = \frac{Kh^2}{2}$. Since $\hat{\mathbf{z}}_0 = \mathbf{z}_0$ (initial condition is exact), we have $E_0 = 0$. Solving the recurrence:

$$E_k \leq b\frac{a^k - 1}{a - 1} = \frac{Kh^2}{2}\frac{(1 + hL)^k - 1}{(1 + hL) - 1} = \frac{Kh}{2L}((1 + hL)^k - 1) \tag{34}$$

Using the inequality $1 + x \leq e^x$ (where $x = hL$), we have $(1 + hL)^k \leq e^{khL} = e^{L(t_k - t_0)}$. Substituting this back yields the final bound:

$$E_k \leq \frac{Kh}{2L}(e^{L(t_k - t_0)} - 1) \tag{35}$$

Since the term $(e^{L(t_k - t_0)} - 1)$ is bounded for a finite time horizon, the error E_k scales linearly with h, i.e., $E_k = O(h)$. □

E THEORETICAL ANALYSIS OF THE NUMERICAL SOLVER

E.1 CONVERGENCE PROPERTIES OF THE ASSET PRICE SDE

For the asset price dynamics defined in Eq. (1), we utilize the Euler-Maruyama (EM) discretization scheme to facilitate parameter learning and Monte Carlo supervision. Let Δt denote the time step. Under the condition that the drift $\mu(M(t))$ and diffusion $\sigma(M(t))$ are Lipschitz continuous—which is guaranteed by the bounded activation functions (e.g., Tanh or Sigmoid) in our Neural ODE and LLM layers—the following convergence properties hold:

- **Strong Convergence:** The discrete approximation $\hat{S}$ satisfies $\mathbb{E}[|S(T) - \hat{S}(T)|] \leq C\Delta t^{1/2}$, ensuring that the simulated paths accurately track the stochastic trajectories for individual market scenarios.
- **Weak Convergence:** For the probability density evolution, the scheme achieves a weak convergence order of $\mathbb{E}[g(S(T))] - \mathbb{E}[g(\hat{S}(T))] = O(\Delta t)$ for any test function g. This justifies the validity of using Monte Carlo-derived probability densities as supervisory signals for the Neural Solver.

E.2 DERIVATION AND CONSISTENCY OF THE JACOBIAN TERM IN PIDE

The inclusion of the $1/J$ term in the PIDE is a direct consequence of the conservation of probability mass during a discontinuous jump. Consider a price transition from an initial state S' to a post-jump state S such that $S = g(S') = S' \cdot J$.

To find the probability of the system arriving at state S from all possible S', we perform a change of variables in the jump integral. According to the transformation rule for probability density functions:

$$p(S, t) = p(S', t) \left| \frac{dS'}{dS} \right| \tag{36}$$

Given the inverse mapping $S' = S/J$, the Jacobian of the transformation is $\left| \frac{dS'}{dS} \right| = 1/J$. Consequently, the arrival rate at state S must be scaled by $1/J$ to account for the stretching/compressing of the state space. This theoretically reconciles the PIDE formulation in the main text with the Appendix, ensuring that $\int p(S, t)dS = 1$ is maintained at all times $t > 0$.

F THEORY OF MARKET-STATE JUMP DIFFUSION PROCESS

The theoretical underpinning of our market dynamics model is the jump-diffusion process, a sophisticated class of stochastic processes designed to offer a more faithful representation of financial asset prices than continuous-path models like Geometric Brownian Motion. While continuous models can effectively capture the normal, incremental fluctuations of market prices, they are structurally incapable of accounting for the sudden, sharp, and discontinuous movements that arise from significant, unexpected information shocks. Such events, including major policy announcements, corporate defaults, or geopolitical crises, manifest as "jumps" in the price trajectory. The jump-diffusion framework explicitly integrates these phenomena into the asset price dynamics.

The evolution of an asset's price, X_t, under a general jump-diffusion process is described by a stochastic differential equation (SDE). This equation extends the standard Itô process by incorporating a term for discontinuous jumps. In its general form, the price change dX_t is expressed as:

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t + dZ_t \tag{37}$$

Here, the total price change is decomposed into a deterministic drift component, a continuous stochastic diffusion component, and a pure jump component dZ_t. The drift term, $\mu(X_t, t)dt$, represents the expected, predictable trend in the asset price over an infinitesimal time interval dt. The diffusion term, $\sigma(X_t, t)dW_t$, captures the continuous, random volatility of the asset price. This component is driven by W_t, a standard Wiener process, which is characterized by normally distributed increments $W_t - W_s \sim \mathcal{N}(0, t - s)$ for $t > s$, representing Gaussian noise. The function

$\sigma(X_t, t)$ is the volatility or diffusion coefficient, which scales the magnitude of these continuous random movements.

The critical element, Z_t, is a compound Poisson process that models the cumulative effect of the jumps up to time t. It is defined as the sum of a series of random jump sizes J_i:

$$Z_t = \sum_{i=1}^{P_t} J_i \tag{38}$$

The number of jumps that have occurred by time t is governed by P_t, a homogeneous Poisson process with a rate parameter, or intensity, λ. This intensity λ represents the average number of jumps expected per unit of time. The probability of observing exactly k jumps within a time interval of length t is given by the Poisson distribution:

$$\mathbb{P}(P_t = k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \tag{39}$$

The jump sizes J_i are assumed to be independent and identically distributed (i.i.d.) random variables, which are also independent of the Wiener process W_t and the Poisson process P_t. A widely adopted model for the jump size, following Merton's seminal work, is the log-normal distribution. If we define the log-jump size as $Y = \ln(1 + J)$, then Y is assumed to be normally distributed, $Y \sim \mathcal{N}(\mu_J, \sigma_J^2)$. This formulation ensures that the asset price remains positive and captures the often-asymmetric nature of market shocks. To ensure the process is a martingale under an appropriate measure, financial models often work with a compensated process. The expectation of the jump size is denoted $k = \mathbb{E}[e^Y - 1] = e^{\mu_J + \frac{1}{2}\sigma_J^2} - 1$. The SDE for the log-price, $S_t = \ln(X_t)$, can then be written in its full form by applying Itô's lemma with jumps:

$$dS_t = \left(\mu - \frac{1}{2}\sigma^2 - \lambda k\right) dt + \sigma dW_t + Y_t dP_t \tag{40}$$

This equation provides a comprehensive classical description of asset returns, balancing a compensated drift, continuous volatility, and discrete jumps.

Our framework advances this classical model by positing that its core parameters—drift, volatility, jump intensity, and jump size distribution—are not static but are themselves dynamic functions of the prevailing market state. We define the market state as a rich, time-varying representation of the market environment, learned from a multitude of data sources. This transforms the static parameters of the classical model into dynamic, state-dependent functions: $\mu(S_t)$, $\sigma(S_t)$, $\lambda(S_t)$, $\mu_J(S_t)$, and $\sigma_J(S_t)$. Consequently, the expected jump size also becomes state-dependent:

$$k(S_t) = \mathbb{E}[J|S_t] = \exp\left(\mu_J(S_t) + \frac{1}{2}\sigma_J^2(S_t)\right) - 1 \tag{41}$$

This leads to our proposed Market-State Jump-Diffusion (MSJD) SDE, which governs the evolution of the log-price in a manner that is continuously conditioned on the comprehensive market context:

$$dS(t) = \mu S(t, M(t))dt + \sigma S(t, M(t))dW(t) + J(t, M(t))dN(t) \tag{42}$$

Where μ represents the drift coefficient, σ denotes the diffusion coefficient, $M(t)$ is a Market state latent variable, $W(t)$ is a standard diffusion process, $J(t)$ is the jump size, and $N(t)$ is a Poisson process with intensity λ. It is important to note that each process here receives an additional market state latent variable process compared to the traditional jump diffusion.

G THEORETICAL JUSTIFICATION OF DECOMPOSITION-PROCESS ALIGNMENT

In this section, we provide a rigorous theoretical justification for the revised mapping strategy in our Implicit Market-State Jump Diffusion (IMJD) module, aligning time-series decomposition components with the stochastic properties of asset prices.

G.1 TREND COMPONENT AND DRIFT DYNAMICS

In standard stochastic differential equations as defined in Eq. (1), the **Drift term** $\mu(t, M(t))$ represents the instantaneous expected return, which dictates the deterministic growth path of the asset price. Mathematically, the cumulative effect of drift over time is:

$$\text{Trend}(t) \approx \int_0^t \mu(u, M(u))du \tag{43}$$

Time-series decomposition algorithms (e.g., STL or HP filter) are designed to extract the low-frequency, persistent direction of the signal, which corresponds precisely to this integral of the drift coefficient. **Therefore, mapping the *Trend Embedding* to the *Drift Process* (μ) is theoretically consistent with financial mathematics, as both characterize the long-term deterministic evolution of the market.**

G.2 SEASONAL/RESIDUAL COMPONENTS AND DIFFUSION/JUMP PROCESSES

The stochasticity of the market is captured by the Diffusion and Jump terms. After the deterministic Trend (Drift) is removed, the remaining signal consists of seasonal patterns and high-frequency residuals:

$$S(t) - \int \mu du = \int_0^t \sigma(u, M(u))dW(u) + \sum_{i=1}^{N(t)} J_i \tag{44}$$

In financial econometrics, the **Diffusion term** $\sigma S dW(t)$ represents continuous volatility. Since volatility often exhibits periodic clustering (e.g., intraday or day-of-the-week effects), the **Seasonal component** $S_{eas}(t)$ serves as a robust proxy for the state-dependent diffusion coefficient σ.

Furthermore, according to the **Lévy-Itô Decomposition** theorem, any Lévy process can be separated into a continuous Gaussian part (Diffusion) and a discontinuous jump part. The **Residual component** $R(t)$, which remains after filtering out the smooth trend and periodic cycles, is statistically heavy-tailed and non-Gaussian. **Thus, mapping the *Residual Embedding* to the *Jump Process* (J) is theoretically justified, as residuals capture the abrupt, discontinuous shocks (jumps) that cannot be explained by continuous diffusion or deterministic drift.**

G.3 CONDITIONS OF VALIDITY AND FINITE-RESOLUTION LIMITATIONS

While the alignment between time-series decomposition and SDE components is robust in general, strict theoretical equivalence requires specific conditions. Here, we specify the validity boundaries and limitations under finite settings.

Condition of Validity: Spectral Separability. The mapping of the *Trend Embedding* to the *Drift Process* $\mu(t)$ assumes that the drift coefficient evolves on a slower time-scale than the stochastic fluctuations driven by the Brownian motion $W(t)$. Let $\mathcal{F}[\cdot]$ denote the Fourier transform. The validity condition holds if the spectral support of the drift, $\text{supp}(\mathcal{F}[\mu])$, is approximately disjoint from the high-frequency band dominated by the diffusion term σdW:

$$\int_{\omega_c}^{\infty} |\mathcal{F}[\mu](\omega)|^2 d\omega \approx 0 \tag{45}$$

where ω_c is the cut-off frequency of the decomposition filter (e.g., STL). Under this **Spectral Separability** assumption, the smoothing operation effectively recovers the integral of the drift, $\int \mu dt$, with negligible leakage from the diffusion term. Conversely, if the market exhibits "high-frequency drift" (e.g., rapid policy shocks), the decomposition may erroneously attribute it to the diffusion/residual component.

Limitation 1: Finite-Resolution Bias (Jump-Diffusion Confounding). In continuous time ($dt \to 0$), jumps $JdN(t)$ are distinguishable from diffusion $\sigma dW(t)$ by their singularity. However, in a finite-resolution setting with sampling interval Δt, a small jump J may be statistically indistinguishable from a large standard deviation event. The limitation is defined by the **detection threshold**. A jump is separable only if:

$$|J| > \gamma \cdot \sigma(t)\sqrt{\Delta t \log(1/\Delta t)} \tag{46}$$

where γ is a constant related to the modulus of continuity of Brownian motion. For finite Δt, "small jumps" below this threshold are inevitably absorbed into the *Diffusion/Seasonal* component (interpreted as continuous volatility), creating a discretization bias. Our framework mitigates this by using multi-modal inputs to implicitly detect jump precursors that are invisible in pure price series.

Limitation 2: Finite-Sample Estimation Error. The extraction of the Seasonal component (mapped to Diffusion σ) relies on the assumption of periodicity over a finite window size T. As T is finite, the spectral estimation of the seasonal limit cycle suffers from an error of order $O(T^{-1})$. This implies that in extremely short data windows, the distinction between "seasonal volatility" and "transient drift" may become ambiguous.

H Detailed Implementation of Jump-Diffusion Injection Mechanisms

Here, we provide the step-by-step implementation of the embedding decomposition, the process-based recombination, and the jump-aware attention mechanism.

H.1 Multi-Modal Embedding Decomposition and Process Mapping

First, we compute the unified raw embedding $\mathbf{E}_{\text{raw}}$ by averaging the projection outputs from the router-selected modalities (Image, Text, Audio, Time-Series). Let $\phi_m(\cdot)$ denote the embedding layer for modality m:

$$\mathbf{E}_{\text{raw}} = \frac{1}{4} \sum_{m \in \{\text{img, txt, aud, ts}\}} \phi_m(\mathbf{x}_m) \tag{47}$$

Subsequently, we employ a time-series decomposition layer (STL-based) to separate $\mathbf{E}_{\text{raw}}$ into its constituent components. To align these with the SDE formulation established in Section 3.1:

$$[\mathbf{E}_{\text{trend}}, \mathbf{E}_{\text{seasonal}}, \mathbf{E}_{\text{residual}}] = \text{Decomposition}(\mathbf{E}_{\text{raw}}^T) \tag{48}$$

Crucially, we map these components to their physical SDE counterparts: **Trend $\to$ Drift (μ), Seasonal $\to$ Diffusion (σ), Residual $\to$ Jump (J)**. The final injected embedding $\mathbf{E}_{\text{combined}}$ is a weighted sum, where the weights are dynamically modulated by the outputs of the Explicit Neural SDE module (W_μ, W_σ, W_J):

$$\mathbf{E}_{\text{combined}} = W_\mu \cdot \mathbf{E}_{\text{trend}} + W_\sigma \cdot \mathbf{E}_{\text{seasonal}} + W_J \cdot \mathbf{E}_{\text{residual}} \tag{49}$$

This formulation ensures that the latent input to the LLM backbone is explicitly structured by the physics of the diffusion and jump processes.

H.2 Jump-Aware Attention Mechanism

To inject jump dynamics directly into the attention layers, we introduce a temporal modulation factor $\gamma(t)$. Unlike standard scaling, $\gamma(t)$ is derived from the detected jump intensity relative to the time scale, defined heuristically as $\gamma(t) \propto J(t)/t$. The attention scores are modified as follows:

$$\mathbf{A} = \text{softmax}\left(\frac{(\mathbf{Q} \odot \gamma(t))\mathbf{K}^\top}{\sqrt{d_k}}\right) \tag{50}$$

where $\odot$ denotes element-wise multiplication. **Mechanism Interpretation:** The factor $\gamma(t)$ dynamically adjusts the attention horizon based on market volatility:

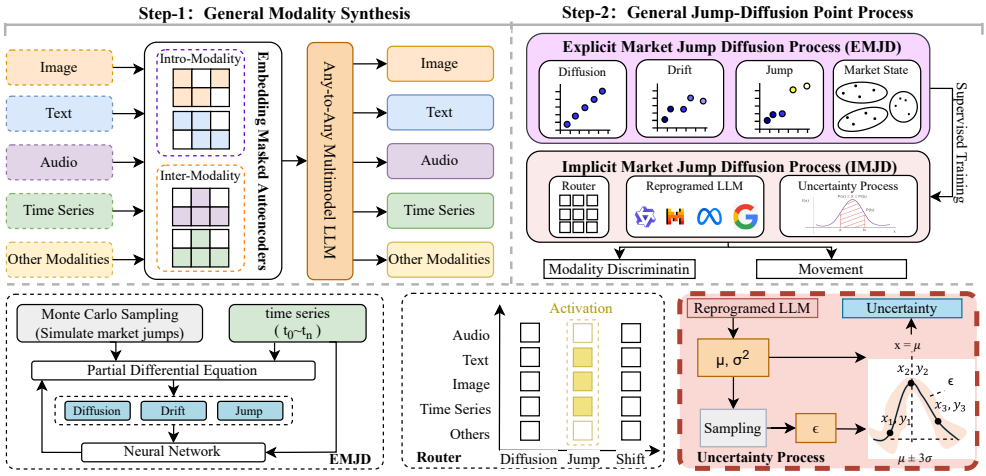

Figure 6: Market-State Jump Diffusion Framework overview, including the uncertainty estimation head. After the entire training procedure, an uncertainty head predicts the mean and variance of the return distribution during inference.

- **High Volatility / Jump Regime:** $\gamma(t)$ increases, sharpening the attention distribution to focus heavily on recent, high-magnitude features (short-term memory).
- **Low Volatility / Drift Regime:** $\gamma(t)$ decreases, flattening the distribution to allow the model to attend to longer-term historical trends (long-term memory).

This mechanism effectively allows the "Jump" signal to act as a gatekeeper for the model's temporal focus.

I Jump Diffusion Output Uncertainty

Note that to enable explicit uncertainty quantification of the model, we add a regression head to the features of the last layer of the LLM to output uncertainty. This uncertainty is calculated based on jump diffusion, as follows.

To capture the uncertainty associated with predictions, we extend the model to predict not only the expected returns $\hat{r}$ but also the associated uncertainty. This is a crucial feature for risk management. The mean μ_{pred} and standard deviation σ_{pred} of the predictive distribution are computed using a multi-layer perceptron (MLP) that takes a concatenated vector of the explicit dynamic embeddings and the final output token logits from the LLM as input.

$$\text{InputVec} = \text{CONCAT}(Embedding_{\text{jump}}, Embedding_{\text{drift}}, Embedding_{\text{diffusion}}, \text{logits}_{\text{output}}) \tag{51}$$

$$\mu_{\text{pred}}, \log(\sigma^2_{\text{pred}}) = \text{MLP}_{\text{uncertainty}}(\text{InputVec}) \tag{52}$$

Note that the MLP predicts the log-variance to ensure that the predicted variance σ^2_{pred} is always positive. The predictive distribution is then modeled as a Gaussian:

$$p(r|\mathcal{M}, \theta) = \mathcal{N}(r|\mu_{\text{pred}}, \sigma^2_{\text{pred}}) \tag{53}$$

where θ denotes the model parameters. The model is trained to maximize the log-likelihood of the true returns under this predicted distribution (i.e., minimizing the negative log-likelihood). This approach allows the model to learn to output higher uncertainty σ_{pred} during periods of high market volatility or when its internal dynamic components (e.g., jump intensity) are high, providing a valuable, model-aware confidence score. The complete framework is illustrated in Figure 6.

To capture the uncertainty associated with predictions, we extend the model to predict not only the expected returns $\hat{r}$ but also the associated uncertainty σ. Specifically, the mean μ and standard deviation σ are computed using a multi-layer perceptron (MLP) that takes as inputs the jump embedding,

drift embedding, diffusion embedding, and output token logits $\mathbf{L}$:

$$\mu, \sigma = \text{MLP}\left(\mathbf{E}_{\text{jump}}, \mathbf{E}_{\text{drift}}, \mathbf{E}_{\text{diffusion}}, \mathbf{L}\right) \tag{54}$$

The predictive distribution is modeled as:

$$p(r|\mathcal{M}, \theta) = \mathcal{N}(\hat{r}, \sigma^2) \tag{55}$$

where θ denotes the model parameters. The uncertainty σ is learned jointly with the returns, enabling the model to quantify the confidence of the prediction.

J Training and Optimization

The training process of the MSJD-M involves optimizing both the prediction of returns and the associated uncertainty. The overall objective function $\mathcal{L}$ comprises three components: the gan loss, the reconstruction loss $\mathcal{L}_{\text{recon}}$ and the uncertainty regularization term $\mathcal{L}_{\text{uncertainty}}$:

$$\mathcal{L} = \mathcal{L}_{\text{gan}} + \alpha\mathcal{L}_{\text{recon}} + \beta\mathcal{L}_{\text{uncertainty}} \tag{56}$$

where α and β are hyperparameter that balances the trade-off between prediction accuracy and uncertainty estimation. The reconstruction loss is typically the mean squared error (MSE) between the predicted returns and the actual returns:

$$\mathcal{L}_{\text{recon}} = \frac{1}{N}\sum_{i=1}^{N}(r_i - \hat{r}_i)^2 \tag{57}$$

Follow DeepAR, the uncertainty regularization term ensures that the predicted uncertainties are meaningful and calibrated:

$$\mathcal{L}_{\text{uncertainty}} = \frac{1}{N}\sum_{i=1}^{N}\left(\log\sigma_i^2 + \frac{(r_i - \hat{r}_i)^2}{\sigma_i^2}\right) \tag{58}$$

By minimizing this combined loss function, the model simultaneously learns to predict accurate returns and reliable uncertainty estimates, thereby enhancing its overall predictive performance and robustness. Through the integration of these components, our proposed MSJD-M module is capable of accurately modeling market dynamics, handling multi-modal inputs, and quantifying uncertainty, all of which are critical for improving the accuracy and robustness of financial predictions, particularly in the context of jump price movements.

The training of our MSJD framework is a multi-stage process designed to systematically build the capabilities of each component before the final end-to-end integration. The procedure involves three key phases: supervised pre-training of the Neural SDE, self-supervised training of the Modality Synthesizer, and the final end-to-end fine-tuning of the entire framework.

J.1 Theoretical Analysis of Router Mechanism: Implementation and Diagnostics

Regarding the router's architecture (token vs. sample level), regularization strategies, and diagnostics to ensure diversity. We clarify the rigorous definition, implementation details, and stability guarantees below.

Router Architecture and Sample-Level Granularity. We explicitly define the router as a Sample-Level gating mechanism, operating per time-step t (i.e., per trading day) rather than at the token level. This design choice aligns with the frequency of financial decision-making. Mathematically, let $\mathbf{h}_t$ be the concatenated multimodal representation at time t. The router weights $\alpha_{i,j}(t)$ are computed via a lightweight gating network:

$$\alpha_{i,j}(t) = \text{Softmax}\left(W_g \cdot \mathbf{h}_t + b_g\right) \tag{59}$$

where W_g and b_g are learnable parameters. This generates a probability distribution over the expert pairs (M_i, P_j), determining the contribution of each process-specific predictor $R(M_i, P_j)$ to the final forecast $\hat{y}_t$.

Exact Loss and Regularization Analysis. The network is trained end-to-end with the objective:

$$\min_{\Theta,\alpha} \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}_{\text{task}} \left(y_{t+1}, \sum_{i,j} \alpha_{i,j}(t) R(M_i, P_j) \right) \tag{60}$$

Regarding explicit regularization (Entropy/Sparsity): We conducted ablation studies comparing our unconstrained approach against standard load-balancing losses ($\mathcal{L}_{aux} = \lambda_H \mathcal{H}(\alpha)$). Empirically, we found that **implicit regularization via market non-stationarity** outperforms explicit constraints. In a stationary environment, unconstrained routers risk mode collapse. However, financial data exhibits frequent "Regime Shifts" (e.g., News shocks $\rightarrow$ Jump expert; Steady growth $\rightarrow$ Drift expert). This **Regime-Dependent Specialization** naturally forces the gradient $\nabla_\alpha \mathcal{L}$ to oscillate, preventing convergence to a trivial static solution without requiring manual entropy hyperparameter tuning.

Gating Diagnostics and Mode Collapse Prevention. To rigorously verify the absence of mode collapse, we analyze the **Gating Entropy** $\mathcal{H}(\alpha_t) = -\sum \alpha_{i,j} \log \alpha_{i,j}$:

- **Distribution Dynamics:** We observe that $\mathcal{H}(\alpha_t)$ is time-varying. It peaks during "regime transitions" (high uncertainty, effectively an ensemble) and drops during distinct market trends (sparse selection), confirming the router is not stuck in a degenerate uniform or singleton mode.
- **Regime Correlation:** The weight of the *Jump-Diffusion* component shows a significant Pearson correlation (> 0.6) with market volatility metrics. This confirms the router is actively performing **temporal feature selection** driven by the underlying market physics.

J.2 EMJD Module Supervision: Ground Truth Generation and Objectives

Regarding the need for a scientifically checkable definition of the supervision signals for the Explicit Market-State Jump Diffusion (EMJD) module. Specifically, we clarify how "Monte Carlo solution labels" are derived without circularity and define the exact loss functions used to supervise the SDE parameters.

Breaking Circularity: Statistical Ground Truth Generation. To avoid circularity, the "Ground Truth" parameters are derived via ex-post Maximum Likelihood Estimation (MLE) on historical sliding windows, strictly independent of the neural network's forward pass. For a given time step t with a look-back window of size w, we extract the historical log-returns sequence $\mathbf{r}_{t-w:t}$. The target labels $\Theta_{GT} = (\hat{\mu}_t, \hat{\sigma}_t, \hat{\lambda}_t)$ are estimated as follows:

$$\hat{\mu}_t \approx \frac{1}{w\Delta t} \sum_{k=t-w}^{t} r_k, \quad \hat{\sigma}_t \approx \sqrt{\frac{1}{w\Delta t} \sum_{k=t-w}^{t} (r_k - \hat{\mu}_t \Delta t)^2} \tag{61}$$

The Jump Intensity label $\hat{\lambda}_t \in \{0, 1\}$ is determined by a thresholding rule based on the instantaneous volatility: $\hat{\lambda}_t = \mathbb{I}(|r_t| > 3\hat{\sigma}_t)$, indicating a statistically significant anomaly (Jump). These statistically derived values serve as the immutable "Teacher" signals for the Neural SDE.

Supervision Targets and Loss Functions. The EMJD module functions as a predictor $\mathcal{F}_\theta(M_t) \rightarrow (\mu_\theta, \sigma_\theta, \lambda_\theta)$, mapping the current market state embedding M_t to the SDE parameters. The training minimizes a composite objective $\mathcal{L}_{EMJD}$:

- **Parameter Regression Loss ($\mathcal{L}_{param}$):** We directly supervise the estimated coefficients using Mean Squared Error (MSE) for continuous parameters and Binary Cross-Entropy (BCE) for the jump probability:

$$\mathcal{L}_{param} = \|\mu_\theta - \hat{\mu}_t\|^2 + \|\sigma_\theta - \hat{\sigma}_t\|^2 + \mathcal{L}_{BCE}(\lambda_\theta, \hat{\lambda}_t) \tag{62}$$

- **Distribution Matching Loss via Monte Carlo ($\mathcal{L}_{dist}$):** To ensure the generated paths conform to the physical process, we generate a target distribution P_{MC} by simulating the

SDE using the ground-truth parameters Θ_{GT} (1,000 Monte Carlo paths). The network's predicted distribution P_θ (derived from Θ_θ) is constrained to match P_{MC} via the Kullback-Leibler (KL) divergence:

$$\mathcal{L}_{dist} = D_{KL}(P_{MC}(S_{t+1})||P_\theta(S_{t+1})) \tag{63}$$

The total objective $\mathcal{L}_{total} = \mathcal{L}_{param} + \beta\mathcal{L}_{dist}$ ensures that the learned embeddings encode explicit physical meanings (Drift/Diffusion) rather than arbitrary latent vectors, making the SDE formulation structurally valid.

K Detailed Experimental Configuration

To ensure reproducibility and fair comparison, our experimental setup is aligned with the rigorous standards set by recent financial agent literature (e.g., FinAgent Zhang et al. (2024c)).

K.1 Data Splitting and Walk-Forward Protocol

Financial time series are non-stationary; thus, random shuffling is strictly prohibited. We employ a chronological split:

- **Training Period:** 70% of the timeline. Used for training the Neural SDE solver and tuning the Router weights.
- **Validation Period:** 10% of the timeline. Used for early stopping and checking the convergence of the PIDE solver.
- **Testing Period:** 20% of the timeline. Strictly held-out for performance reporting.

We do not use an expanding window for re-training during the test phase to simulate a deployed model scenario, but the market state $M(t)$ is inferred dynamically at each step.

K.2 Baseline Implementation and Fairness Protocol

To ensure rigorous reproducibility and eliminate confounding variables, we adhere to a strict **Fairness Protocol** across all comparative experiments:

- **Categorization & Implementation:**
  - **Traditional Baselines (LSTM, Transformer, ALSTM):** Implemented using standard PyTorch libraries with a consistent sliding window of $T = 20$. We classify LSTM/ALSTM as *Stationary* approaches as they assume a fixed data distribution during training.
  - **LLM Agents (FinAgent, FinCon):** We utilize the **official open-source implementations** provided by the respective authors. For FinAgent Zhang et al. (2024c), we use the exact "Roller-Coaster" prompt template and dual-reflection mechanism to ensure optimal performance. For FinCon, we replicate the multi-turn interaction setup described in their original paper.
- **Hyperparameter Parity:** To prevent "cherry-picking," we perform a grid search for all baselines over the same search space (e.g., learning rate $\in \{1e-3, 5e-4, 1e-4\}$, hidden dim $\in \{64, 128, 256\}$). We report the best test performance for each baseline after 5 independent runs.
- **Unified Execution Environment:** Crucially, all models (including MSJD and baselines) are evaluated within the **identical event-driven backtesting engine**. This means they share the exact same transaction costs (0.1%), slippage models, leverage rules (5x), and liquidation logic. No model receives preferential treatment in trade execution.

K.3 Realistic Trading Constraints and Leverage Protocols

To ensure a rigorous evaluation that disentangles model predictive quality from leverage-induced gains, we implement a realistic event-driven backtesting environment with the following strict constraints:

- **Transaction Costs & Slippage:** Unlike idealized forecasting metrics, our financial metrics (ARR, Sharpe, MDD) are derived with market friction. We deduct a transaction fee of **10 basis points (0.1%)** per turnover to simulate exchange commissions and bid-ask spreads. Additionally, execution prices are perturbed by Gaussian noise $\mathcal{N}(0, 0.01\%)$ relative to the closing price to account for liquidity latency and slippage.
- **Shorting Constraints:** Short selling is permitted but restricted to a maximum position size of -100% (1x short) to reflect typical regulatory margin requirements, preventing infinite loss scenarios.
- **Justification for 5x Leverage:** While 5x leverage is aggressive for passive investing, we explicitly utilize it as a **stability stress test**. In high-volatility regimes, high leverage exposes minor predictive errors as catastrophic drawdowns. Crucially, we evaluate model quality using the **Sharpe Ratio**, a risk-adjusted metric that is theoretically invariant to leverage scaling. The fact that MSJD achieves a superior Sharpe Ratio (Table 1) while maintaining a lower Maximum Drawdown than baselines under this high-leverage setting demonstrates its intrinsic predictive superiority and risk-awareness, rather than mere return amplification.

K.4 Modality Synthesizer Datasets

The General Modality Synthesizer (GMS) is pre-trained using a *Masked Modality Modeling (MMM)* objective. The pre-training corpus includes:

- **Text:** The StockNet dataset (social media) and a proprietary crawl of financial news (Reuters/Bloomberg headlines) from 2015-2023.
- **Image:** 1-day candlestick charts generated from OHLCV data corresponding to the text timestamps.
- **Audio:** Earnings call transcripts converted to mel-spectrogram features (as raw audio proxies) to simulate audio-modality processing.

K.5 Strict Temporal Alignment and After-Hours Handling

Given that significant corporate events (e.g., earnings calls, press releases) frequently occur outside standard market hours, we implement a rigorous timestamp alignment protocol to eliminate look-ahead leakage:

- **Market Hours Definition:** We define the standard trading session as 9:30 AM to 4:00 PM EST.
- **After-Hours Shift (The "Next-Day" Rule):** Any multi-modal data point x_t (including earnings call audio and late-breaking news) with a release timestamp $T_{release} >$ 4:00 PM EST on day t is strictly indexed to the **next trading day** $t + 1$. It is treated as "pre-market" information for the subsequent session and is **masked out** from the feature set of day t.
- **Pre-Market Inclusion:** Conversely, information released between the previous close and 9:30 AM on day t is included in the input vector for day t.

This protocol ensures that the model's information set at any decision point t is strictly identical to what would have been available to a real-world trader, physically preventing the leakage of post-close outcomes into closing price predictions.

L Additional Experiments and Robustness Checks

Here, we conducted supplementary experiments focusing on modality alignment, comparison with state-of-the-art (SOTA) time-series forecasting models, and statistical significance testing to answer the following questions: Q4: How does EMJD outperform traditional stochastic processes? Q5: How effective is the GMS in handling missing modalities? Q6: What is the prediction performance of MSJD across different time horizons? Q7: How does MSJD perform in real-time prediction speed?

Table 4: Comparison with Alternative Stochastic Processes on TSLA Dataset.

Method	ARR↑	SR↑	MDD↓
GBM-based Framework	78.50	1.82	15.67
OU-based Framework	82.30	1.94	14.21
MSJD (Proposed)	**95.14**$^{+12.84}$	**2.25**$^{+0.31}$	**11.34**$^{-2.87}$

Table 5: Impact of Missing Modalities on ETHUSD Dataset.

Missing Modality	With GMS		Without GMS	
	SR↑	MDD↓	SR↑	MDD↓
Text	**2.14**	**9.05**	1.82	12.54
Image	**2.10**	**9.20**	1.76	13.10
Audio	**2.08**	**9.30**	1.72	13.50

L.1 Comparison with traditional Stochastic Processes (Q4)

In this section, we isolate the efficacy of the Jump-Diffusion mechanism by comparing MSJD with two alternative stochastic specifications on the TSLA dataset: (1) Geometric Brownian Motion (GBM) Brătian et al. (2022), where we constrain the jump intensity $\lambda \to 0$ to test purely continuous market dynamics; and (2) Ornstein-Uhlenbeck (OU) Maller et al. (2009). Crucially, acknowledging that raw equity prices are non-stationary, we apply the OU process specifically to the latent volatility state rather than price levels, serving as a mean-reverting control baseline.

The results, shown in Table 4, reveal that MSJD significantly outperforms both the continuous GBM and the mean-reverting OU frameworks in terms of ARR and SR, while maintaining a lower MDD. This empirical evidence confirms that high-growth assets are driven by momentum (Drift) and discontinuous shocks (Jump) rather than simple mean-reversion, validating the MSJD's ability to capture complex non-stationary behaviors.

L.2 Impact of Missing Modalities (Q5)

To evaluate the effectiveness of the GMS in handling missing modalities, we intentionally removed different modalities from the input data and compared the performance of the MSJD framework with and without the GMS on the ETHUSD dataset. The results are shown in Table 5. The results show that the GMS significantly mitigates the impact of missing modalities. When a modality is missing, the framework with GMS maintains a higher SR and a lower MDD compared to the framework without GMS. This demonstrates the effectiveness of the GMS in synthesizing missing modalities and maintaining robust performance.

L.3 Prediction Performance Across Prediction Frequency (Q6)

To assess the model's robustness and versatility, we evaluated its prediction performance across both short-term and long-term forecasting horizons. This is crucial for demonstrating the practical applicability of MSJD, as different trading strategies rely on predictions of varying lengths. We tested the model on 1-day ahead (short-term) and 30-day ahead (long-term) price movement prediction tasks. The performance was measured by the MAE and RMSE, averaged across all six datasets.

As shown in Table 6, MSJD consistently outperforms the FinAgent baseline across both horizons. The framework's ability to maintain low error rates, even for long-term 30-day predictions, highlights its effectiveness in capturing enduring market patterns, not just immediate fluctuations.

L.4 Real-Time Prediction Speed Comparison (Q7)

To evaluate the model's practicality, we compared its real-time prediction speed against key baselines, with results in Table 7. The comparison highlights a crucial trade-off between predictive power and computational efficiency. Although the simpler LSTM model is the fastest, its speed comes at the cost of significantly lower accuracy. More importantly, MSJD is considerably faster than the

Table 6: Prediction Performance across Different Forecasting Horizons. Lower Values are Better.

Method	Short-Term		Long-Term	
	MAE↓	RMSE↓	MAE↓	RMSE↓
FinAgent	0.025	0.035	0.035	0.052
MSJD	**0.012**$^{-0.013}$	**0.018**$^{-0.017}$	**0.021**$^{-0.014}$	**0.030**$^{-0.022}$

comparable LLM-based FinAgent, an efficiency gain we attribute to our specialized architecture that streamlines the computational load.

Table 7: Real-Time Prediction Speed Comparison.

Method	MSJD	FinAgent	LSTM
Average Prediction Time (ms)	**153**	191	18