Arbitrage-Free Implied Volatility Surfaces using Conditional Variational Autoencoders

Regime-Aware Generation with Multi-Constraint Optimization

Aditya Chauhan (251270601)[1] and Lavanya Pareek (241270041)[2]

[1]Department of Economics — IIT, Kanpur
[2]Department of Economics — IIT, Kanpur

November 2025

Abstract

This technical report presents a comprehensive framework for generating arbitrage-free implied volatility (IV) surfaces using Conditional Variational Autoencoders (CVAE). The methodology combines the Heston stochastic volatility model with deep generative modeling, conditioned on market and macroeconomic variables. We calibrate single Heston models to NIFTY 50 options data using a two-stage optimization procedure with Wasserstein distance penalties for density matching. The calibrated parameters are then used to train a CVAE that learns the conditional distribution $p(\theta|c)$, where θ represents Heston parameters and c denotes conditioning variables including India VIX, crude oil prices, USD/INR exchange rate, US Treasury yields, and geopolitical unrest indices. The trained model enables regime-aware IV surface generation for stress testing, scenario analysis, and risk management applications. Our implementation enforces arbitrage-free constraints through Feller condition penalties and explicit static/calendar arbitrage checks during training.

Contents

1 Introduction

1.1 Motivation and Background

The pricing and risk management of options contracts fundamentally rely on the concept of implied volatility (IV), which represents the market's expectation of future asset price volatility. Unlike the constant volatility assumption in the Black-Scholes-Merton (BSM) framework (1), empirical observations consistently reveal that implied volatilities vary across strike prices and maturities, forming a complex surface structure known as the *volatility surface* or *volatility smile/skew*.
The Black-Scholes Model has several limitations:

- The BSM model assumes that volatility σ is constant over time and across strikes, which contradicts market observations.
- Volatility Smile: Market prices exhibit systematic patterns where out-of-the-money (OTM) puts and calls trade at higher implied volatilities than at-the-money (ATM) options.
- Volatility Skew: For equity indices, implied volatility typically decreases with strike price, reflecting the market's perception of crash risk.
- Term Structure: Implied volatility varies across maturities, with short-dated options often exhibiting higher volatility than long-dated options during stress periods.

To address these limitations, stochastic volatility models allow volatility itself to be a random process. The Heston model (2) is particularly attractive because:

1. It provides semi-analytical pricing formulas via characteristic functions
2. It captures the leverage effect (negative correlation between asset returns and volatility)
3. It generates realistic volatility smile patterns
4. It satisfies mean-reversion properties observed in volatility dynamics

1.2 Problem Statement

Generating realistic, arbitrage-free IV surfaces is crucial for Risk Management (Accurate valuation of option portfolios and computation of Greeks), Stress Testing (Simulating extreme market scenarios for regulatory compliance), Scenario Analysis(Evaluating portfolio performance under specific market conditions), Model Calibration (Providing consistent initial conditions for pricing exotic derivatives), Market Making , etc.
Two broad classes of prior approaches:

- SDE-based (parametric) models: local volatility, stochastic volatility (Heston, SABR-like families), stochastic-local volatility, etc.
  - Pro: Can be set up to be arbitrage-free by construction.
  - Con: Assumptions about the underlying asset dynamics, which may not match empirical historical dynamics.
- Nonparametric / ML approaches: Regressors (SVMs), neural networks, factor-based methods (PCA + neural SDE), etc.
  - Pro: Very flexible — they can approximate complex surfaces without specifying an SDE for the underlying.
  - Con: Enforcing arbitrage-free constraints is nontrivial.

- Standard models fail to account for market regime shifts (low vol vs. high vol, crisis vs. calm)

Thus, We want to generate implied volatility (IV) surfaces that are both: Faithful to historical behaviour (i.e., they reflect the patterns and dynamics observed in market data), and Arbitrage-free (so the generated surface corresponds to a valid set of option prices under some risk-neutral measure).

1.3 Proposed Solution

We propose a framework combining:

1. **Heston Model Calibration**: Fit single Heston models to daily NIFTY 50 options data using a two-stage optimization with Wasserstein distance penalties
2. **Conditional VAE**: Train a deep generative model to learn $p(\theta|c)$, the distribution of Heston parameters conditioned on market/macro variables
3. **Arbitrage-Free Generation**: Enforce constraints during training to ensure generated surfaces satisfy no-arbitrage conditions
4. **Regime-Aware Sampling**: Generate scenario-specific IV surfaces by conditioning on desired market states

Key Innovations:

- Conditional Generation: Unlike standard VAEs that learn $p(\theta)$, our CVAE learns $p(\theta|c)$, enabling targeted scenario generation
- Wasserstein Density Matching: Ensures model-implied risk-neutral densities match market densities beyond just price fitting
- Multi-Constraint Optimization: Simultaneously enforces Feller condition, static arbitrage, and butterfly arbitrage constraints
- Feature Engineering: Incorporates rolling windows (7d, 30d, quarterly) and geopolitical risk indices (GDELT).

1.4 Datasets used

- Asset: NIFTY 50 Index Options (NSE, India)
- Time Period: 2015-01-01 to 2025-11-10 (approximately 500 trading days after filtering)
- Maturities: 8 tenures (1M, 2M, 3M, 6M, 9M, 12M, 18M, 24M)
- Strikes: 21 log-moneyness points from -20% to +20%
- Risk-Free Rate: $r = 0.067$ (6.7%, representative Indian rate)
- Dividend Yield: $q = 0.0$ (index dividends assumed reinvested)
- Conditioning Variables (8-dimensional):
    1. `crude_oil_30d_mean`: 30-day rolling mean of Brent crude oil prices
    2. `crude_oil_7d_mean`: 7-day rolling mean of crude oil prices
    3. `unrest_index_yearly`: GDELT-based geopolitical unrest index (yearly rolling)

4. `crude_oil`: Current crude oil price
5. `usdinr_quarterly_mean`: Quarterly mean of USD/INR exchange rate
6. `india_vix_30d_mean`: 30-day rolling mean of India VIX
7. `india_vix_7d_mean`: 7-day rolling mean of India VIX
8. `us_10y_yield`: US 10-year Treasury yield

2 Mathematical Foundations

2.1 The Heston Stochastic Volatility Model

The Heston model (2) describes the joint dynamics of an asset price S_t and its instantaneous variance v_t under the risk-neutral measure $\mathbb{Q}$:

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{v_t} S_t dW_t^S \\ dv_t &= \kappa(\theta - v_t)dt + \sigma_v \sqrt{v_t} dW_t^v \\ d\langle W^S, W^v \rangle_t &= \rho \, dt \end{aligned}$$

Parameters: $\kappa > 0$ (Mean reversion speed of variance), $\theta > 0$ (Long-term mean variance level), $\sigma_v > 0$ (Volatility of volatility/vol-of-vol), $\rho \in [-1, 1]$ (Correlation between asset returns and variance innovations), $v_0 > 0$ (Initial variance at time $t = 0$), r (Risk-free interest rate), q (Dividend yield).
Key Properties of Heston Model:

1. Mean Reversion: Variance v_t reverts to long-term mean θ at rate κ.
2. Leverage Effect: Negative ρ captures the empirical observation that volatility increases when prices fall
3. Volatility Clustering: The stochastic variance process generates time-varying volatility
4. Fat Tails: The model produces return distributions with heavier tails than the normal distribution

2.1.1 Feller Condition

To ensure that the variance process v_t remains strictly positive, the Feller condition must be satisfied:

$$2\kappa\theta > \sigma_v^2$$

Interpretation: The drift term $\kappa(\theta - v_t)$ must be strong enough relative to the diffusion term $\sigma_v\sqrt{v_t}$ to prevent v_t from reaching zero. When the Feller condition holds, $v_t > 0$ almost surely for all $t > 0$ given $v_0 > 0$.
Feller condition has various Practical Implications:

- Violations lead to numerical instabilities in pricing algorithms
- Negative variances are economically meaningless
- Calibration procedures should penalize Feller violations

2.1.2 Characteristic Function and Option Pricing

The Heston model admits a semi-closed-form solution for European option prices via the characteristic function. The characteristic function of $\log(S_T)$ under the risk-neutral measure is:

$$\phi(\omega; S_t, v_t, \tau) = \mathbb{E}^{\mathbb{Q}}\left[e^{i\omega \log(S_T)} \mid S_t, v_t\right] = e^{C(\tau;\omega)+D(\tau;\omega)v_t+i\omega\log(S_t)}$$

where $\tau = T - t$ is the time to maturity, and $C(\tau;\omega)$ and $D(\tau;\omega)$ are complex-valued functions satisfying Riccati equations:

$$\begin{aligned} D(\tau;\omega) &= \frac{(\kappa - i\rho\sigma_v\omega - d)}{sigma_v^2}\left(\frac{1-e^{-d\tau}}{1-ge^{-d\tau}}\right) \\ C(\tau;\omega) &= (r-q)i\omega\tau + \frac{\kappa\theta}{\sigma_v^2}\left[(\kappa - i\rho\sigma_v\omega - d)\tau - 2\log\left(\frac{1-ge^{-d\tau}}{1-g}\right)\right] \end{aligned}$$

with

$$\begin{aligned} d &= \sqrt{(\kappa - i\rho\sigma_v\omega)^2 + \sigma_v^2(\omega^2 + i\omega)} \\ g &= \frac{\kappa - i\rho\sigma_v\omega - d}{\kappa - i\rho\sigma_v\omega + d} \end{aligned}$$

European call option prices can be computed using the Carr-Madan formula or direct Fourier inversion. In our implementation, we use **QuantLib's** `AnalyticHestonEngine` for efficient pricing.

2.2 Variational Autoencoders (VAE)

A Variational Autoencoder (4) is a generative model that learns to encode data $\boldsymbol{x}$ into a low-dimensional latent representation $\boldsymbol{z}$ and decode it back to reconstruct $\boldsymbol{x}$.

2.2.1 Model Specification

Generative Model (Decoder):

$$\begin{aligned} p_\theta(\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I}) \quad \text{(prior)} \\ p_\theta(\boldsymbol{x}|\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_\theta(\boldsymbol{z}), \boldsymbol{\Sigma}_\theta(\boldsymbol{z})) \quad \text{(likelihood)} \end{aligned}$$

Inference Model (Encoder):

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\Sigma}_\phi(\boldsymbol{x}))$$

where $\boldsymbol{\mu}_\phi(\cdot)$ and $\boldsymbol{\Sigma}_\phi(\cdot)$ are neural networks parameterized by ϕ, and $\boldsymbol{\mu}_\theta(\cdot)$ and $\boldsymbol{\Sigma}_\theta(\cdot)$ are neural networks parameterized by θ.

2.2.2 Evidence Lower Bound (ELBO)

The VAE maximizes the Evidence Lower Bound on the log-likelihood:

$$\begin{aligned} \log p_\theta(\boldsymbol{x}) &\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p_\theta(\boldsymbol{z})\right) \\ &= \mathrm{ELBO}(\boldsymbol{x}; \theta, \phi) \end{aligned}$$

The ELBO consists of two terms:

1. Reconstruction Term: $\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]$ encourages accurate reconstruction

2. KL Regularization: $\mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)$ regularizes the latent distribution to match the prior

For Gaussian encoder and decoder, the KL divergence has a closed form:

$$\mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right) = \frac{1}{2}\sum_{j=1}^{J}\left(\mu_j^2 + \sigma_j^2 - \log\sigma_j^2 - 1\right)$$

where J is the latent dimension.

2.2.3 Reparameterization Trick

To enable backpropagation through stochastic sampling, the reparameterization trick expresses $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$ as:

$$\boldsymbol{z} = \boldsymbol{\mu}_\phi(\boldsymbol{x}) + \boldsymbol{\sigma}_\phi(\boldsymbol{x}) \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

This separates the stochastic component $\boldsymbol{\epsilon}$ from the learnable parameters ϕ, allowing gradients to flow through $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\sigma}_\phi$.

2.3 Conditional Variational Autoencoders (CVAE)

The Conditional VAE (5) extends the standard VAE by conditioning both the encoder and decoder on additional information $\boldsymbol{c}$ (conditioning variables).

2.3.1 Conditional Model Specification

Conditional Generative Model:

$$\begin{aligned} p_\theta(\boldsymbol{z}|\boldsymbol{c}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I}) \quad \text{(conditional prior, can be extended)} \\ p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c}) &= \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_\theta(\boldsymbol{z}, \boldsymbol{c}), \boldsymbol{\Sigma}_\theta(\boldsymbol{z}, \boldsymbol{c})) \quad \text{(conditional likelihood)} \end{aligned}$$

Conditional Inference Model:

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_\phi(\boldsymbol{x}, \boldsymbol{c}), \boldsymbol{\Sigma}_\phi(\boldsymbol{x}, \boldsymbol{c}))$$

2.3.2 Conditional ELBO Derivation

The conditional log-likelihood can be lower-bounded as:

$$\begin{aligned} \log p_\theta(\boldsymbol{x}|\boldsymbol{c}) &= \log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{c}) d\boldsymbol{z} \\ &= \log \int p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c}) p_\theta(\boldsymbol{z}|\boldsymbol{c}) d\boldsymbol{z} \\ &= \log \int \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})} p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c}) p_\theta(\boldsymbol{z}|\boldsymbol{c}) d\boldsymbol{z} \\ &= \log \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})}\left[\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c}) p_\theta(\boldsymbol{z}|\boldsymbol{c})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})}\right] \end{aligned}$$

By Jensen's inequality:

$$\begin{aligned} \log p_\theta(\boldsymbol{x}|\boldsymbol{c}) &\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})}\left[\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c}) p_\theta(\boldsymbol{z}|\boldsymbol{c})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})}\right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})\right] - \mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})\|p_\theta(\boldsymbol{z}|\boldsymbol{c})\right) \\ &= \mathrm{ELBO}(\boldsymbol{x}, \boldsymbol{c}; \theta, \phi) \end{aligned}$$

Key Difference from Standard VAE:

- The encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})$ takes both data $\boldsymbol{x}$ and conditioning $\boldsymbol{c}$ as input
- The decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})$ generates $\boldsymbol{x}$ conditioned on both latent $\boldsymbol{z}$ and conditioning $\boldsymbol{c}$
- The model learns $p(\boldsymbol{x}|\boldsymbol{c})$ instead of $p(\boldsymbol{x})$, enabling targeted generation

2.3.3 Training Objective

The CVAE is trained by maximizing the conditional ELBO:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{c})\sim p_{data}} \left[\text{ELBO}(\boldsymbol{x}, \boldsymbol{c}; \theta, \phi)\right]$$

In practice, we use Monte Carlo estimation with the reparameterization trick:

$$\mathcal{L}(\theta, \phi) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)}, \boldsymbol{c}^{(i)}) - \text{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)}, \boldsymbol{c}^{(i)}) \| p_\theta(\boldsymbol{z}|\boldsymbol{c}^{(i)})\right)\right]$$

where $\boldsymbol{z}^{(i)} = \boldsymbol{\mu}_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{c}^{(i)}) + \boldsymbol{\sigma}_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{c}^{(i)}) \odot \boldsymbol{\epsilon}^{(i)}$ with $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

3 Heston Model Calibration with Wasserstein Penalties

3.1 Calibration Problem Formulation

For each trading day t, we observe market implied volatilities $\{IV_{t,i,j}\}$ across strikes $i \in \{1, \ldots, N_K\}$ and maturities $j \in \{1, \ldots, N_T\}$. Our goal is to find a single set of Heston parameters $\boldsymbol{\theta} = (\kappa, \theta, \sigma_v, \rho, v_0)$ that best explains the entire surface.
Single Heston vs. Time-Varying Heston: In Single Heston, one parameter is set per day across all maturities (5 parameters); In Time-Varying Heston: Different parameters for each maturity ($5 \times 8 = 40$ parameters).
We adopt the single Heston approach for:

1. Parsimony: Fewer parameters reduce overfitting risk
2. Interpretability: Each parameter has clear economic meaning
3. VAE Training: Lower-dimensional input space (5D vs. 40D)
4. Stability: More robust parameter trajectories over time

3.2 Loss Function Design

The calibration loss function combines multiple objectives:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\text{price}}(\boldsymbol{\theta}) + \lambda_W \mathcal{L}_{\text{Wass}}(\boldsymbol{\theta}) + \lambda_D \mathcal{L}_{\text{drift}}(\boldsymbol{\theta}) + \lambda_F \mathcal{L}_{\text{Feller}}(\boldsymbol{\theta})$$

3.2.1 Price RMSE Term

The primary objective is to minimize the root mean squared error of option prices:

$$\mathcal{L}_{\text{price}}(\boldsymbol{\theta}) = \sqrt{\frac{1}{N} \sum_{i,j} \left(C_{i,j}^{\text{model}}(\boldsymbol{\theta}) - C_{i,j}^{\text{market}}\right)^2}$$

where $C_{i,j}^{\text{model}}(\boldsymbol{\theta})$ is the Heston model price and $C_{i,j}^{\text{market}}$ is the market price (computed from implied volatility using Black-Scholes).

Price Ratio Formulation: To improve numerical stability, we work with normalized price ratios:

$$PR_{i,j} = \frac{C_{i,j}}{S_t e^{-q\tau_j}}$$

This normalization removes the dependence on spot price and discounting, making the optimization more stable.

3.2.2 Wasserstein Distance Penalty

The Wasserstein distance measures the discrepancy between model-implied and market-implied risk-neutral densities.

Risk-Neutral Density Extraction:
By the Breeden-Litzenberger formula (3), the risk-neutral density can be recovered from option prices:

$$f(K,T) = e^{rT} \left. \frac{\partial^2 C}{\partial K^2} \right|_{K,T}$$

In practice, we use central finite differences on a fine grid of strikes:

$$f(K_i,T) \approx e^{rT} \frac{C(K_{i+1},T) - 2C(K_i,T) + C(K_{i-1},T)}{(\Delta K)^2}$$

Wasserstein-1 Distance:
The 1-Wasserstein distance (Earth Mover's Distance) between two probability densities f and g is:

$$W_1(f,g) = \int_{-\infty}^{\infty} |F(x) - G(x)| dx$$

where F and G are the cumulative distribution functions of f and g, respectively.

Computational Procedure:

1. Create a fine grid of $M = 50$ log-moneyness points: $\{\ell_1, \ldots, \ell_M\}$
2. Interpolate market and model price ratios to this grid
3. Compute second derivatives via finite differences to obtain densities
4. Normalize densities to integrate to 1
5. Compute Wasserstein distance using `scipy.stats.wasserstein_distance`
6. Average across all maturities

Rationale:

- Price fitting alone may overfit sparse data
- Wasserstein distance enforces distributional consistency
- Captures tail behavior beyond observed strikes
- More robust to outliers than L2 distance

3.2.3 Drift Penalty

To ensure accurate ATM pricing, we add a drift penalty:

$$\mathcal{L}_{\text{drift}}(\boldsymbol{\theta}) = \left| \frac{1}{N_{\text{ATM}}} \sum_{(i,j)\in\text{ATM}} PR_{i,j}^{\text{model}}(\boldsymbol{\theta}) - \frac{1}{N_{\text{ATM}}} \sum_{(i,j)\in\text{ATM}} PR_{i,j}^{\text{market}} \right|$$

where ATM denotes strikes with $|\log(K/S)| < 0.05$.

3.2.4 Feller Condition Penalty

To encourage satisfaction of the Feller condition, we add a soft penalty:

$$\mathcal{L}_{\text{Feller}}(\boldsymbol{\theta}) = \max(0, \sigma_v^2 - 2\kappa\theta)$$

This penalizes violations but allows the optimizer to explore the boundary if necessary for fit quality.

3.3 Two-Stage Optimization Procedure

We employ a two-stage calibration strategy to balance speed and accuracy:

3.3.1 Stage 1: Fast Calibration

Objective:

$$\boldsymbol{\theta}^{(1)} = \arg\min_{\boldsymbol{\theta}} \left[\mathcal{L}_{\text{price}}(\boldsymbol{\theta}) + \lambda_D \mathcal{L}_{\text{drift}}(\boldsymbol{\theta}) + \lambda_F \mathcal{L}_{\text{Feller}}(\boldsymbol{\theta}) \right]$$

Configuration:

- No Wasserstein computation (fast)
- 10 random initializations for robustness
- L-BFGS-B optimizer with tolerance 10^{-7}
- Maximum 500 iterations

Parameter Bounds: $\kappa \in [10^{-3}, 20.0]$, $\theta \in [10^{-3}, 2.0]$, $\sigma_v \in [10^{-3}, 5.0]$, $\rho \in [-0.999, 0.999]$, and $v_0 \in [10^{-3}, 2.0]$.

3.3.2 Stage 2: Wasserstein Refinement

Objective:

$$\boldsymbol{\theta}^{(2)} = \arg\min_{\boldsymbol{\theta}} \left[\mathcal{L}_{\text{price}}(\boldsymbol{\theta}) + \lambda_W \mathcal{L}_{\text{Wass}}(\boldsymbol{\theta}) + \lambda_D \mathcal{L}_{\text{drift}}(\boldsymbol{\theta}) + \lambda_F \mathcal{L}_{\text{Feller}}(\boldsymbol{\theta}) \right]$$

Configuration:

- Initialize with $\boldsymbol{\theta}^{(1)}$ (Stage 1 result)
- Include Wasserstein penalty with $\lambda_W = 1.0$
- L-BFGS-B optimizer
- Maximum 500 iterations
- Typical runtime: 20-25 seconds per day

Loss Weight Configuration:

$$\begin{aligned} \lambda_W &= 1.0 \quad \text{(Wasserstein weight)} \\ \lambda_D &= 1.0 \quad \text{(Drift penalty weight)} \\ \lambda_F &= 5.0 \quad \text{(Feller penalty weight)} \end{aligned}$$

3.4 Optimization Algorithm: L-BFGS-B

We use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Box constraints (L-BFGS-B) algorithm (6) for calibration.
Advantages:

- Quasi-Newton Method: Approximates Hessian using gradient history
- Limited Memory: Stores only recent gradient information (memory efficient)
- Box Constraints: Naturally handles parameter bounds
- Fast Convergence: Typically converges in 50-200 iterations
- Robust: Works well for non-convex problems

Convergence Criteria:

$$\frac{|\mathcal{L}(\boldsymbol{\theta}^{(k+1)}) - \mathcal{L}(\boldsymbol{\theta}^{(k)})|}{|\mathcal{L}(\boldsymbol{\theta}^{(k)})| + \epsilon} < \text{ftol} = 10^{-7}$$

3.5 Failure Handling and Temporal Continuity

Failure Criteria: A calibration is considered failed if:

- Final loss exceeds threshold: $\mathcal{L}(\boldsymbol{\theta}) > 0.3$
- Optimizer does not converge after maximum iterations
- Heston pricing engine throws exceptions (e.g., extreme parameters)

Fallback Strategy: If calibration fails for day t:

1. Use parameters from day $t-1$: $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$
2. Mark as fallback in metadata
3. Maintain temporal continuity
4. Preserve sample size for VAE training

Success Rate: Typically 85-90% of days calibrate successfully.

3.6 Implementation Code Snippet

Listing 1: Two-Stage Heston Calibration

```
def fit_single_heston(date, iv_surface, spot_price, prev_params=None):
    """
    Fit single Heston model using two-stage optimization.
    """
    # Stage 1: Fast calibration (no Wasserstein)
    def objective_stage1(params):
        kappa, theta, sigma_v, rho, v0 = params
        model = HestonModelQL(kappa, theta, sigma_v, rho, v0, r, q)

        # Price RMSE
        model_pr = compute_price_ratios(model, logm_grid, tau_grid)
        pr_rmse = np.sqrt(np.mean((model_pr - market_pr)**2))

        # Drift penalty (ATM matching)
```

```
        atm_mask = np.abs(logm_grid) < 0.05
        drift_penalty = np.abs(np.mean(model_pr[atm_mask]) -
                               np.mean(market_pr[atm_mask]))

        # Feller penalty
        feller_violation = max(0, sigma_v**2 - 2*kappa*theta)
        feller_penalty = 5.0 * feller_violation

        return pr_rmse + drift_penalty + feller_penalty

    # Optimize Stage 1 with multiple random starts
    best_result = None
    best_loss = np.inf
    for attempt in range(10):
        x0 = initialize_params(prev_params, attempt)
        result = minimize(objective_stage1, x0, method='L-BFGS-B',
                          bounds=bounds, options={'ftol': 1e-7})
        if result.fun < best_loss:
            best_loss = result.fun
            best_result = result

    stage1_params = best_result.x

    # Stage 2: Wasserstein refinement
    def objective_stage2(params):
        kappa, theta, sigma_v, rho, v0 = params
        model = HestonModelQL(kappa, theta, sigma_v, rho, v0, r, q)

        # Price RMSE + Drift + Feller (same as Stage 1)
        pr_rmse = compute_price_rmse(model, market_data)
        drift_penalty = compute_drift_penalty(model, market_data)
        feller_penalty = 5.0 * max(0, sigma_v**2 - 2*kappa*theta)

        # Wasserstein distance (averaged across maturities)
        wass_distances = []
        for tau in tau_grid:
            wass = compute_wasserstein_distance(model, market_data, tau
                )
            wass_distances.append(wass)
        avg_wass = np.mean(wass_distances)

        return pr_rmse + drift_penalty + feller_penalty + avg_wass

    # Optimize Stage 2 starting from Stage 1 result
    result_stage2 = minimize(objective_stage2, stage1_params,
                             method='L-BFGS-B', bounds=bounds,
                             options={'ftol': 1e-5, 'maxiter': 500})

    final_params = result_stage2.x
    return final_params
```

4 Conditional VAE Architecture and Training

4.1 Network Architecture

Our Conditional VAE takes as input:

- Parameters $\boldsymbol{\theta} \in \mathbb{R}^5$: Heston parameters $[\kappa, \theta, \sigma_v, \rho, v_0]$
- Conditioning $\boldsymbol{c} \in \mathbb{R}^8$: Market/macro variables

And outputs reconstructed parameters $\hat{\boldsymbol{\theta}} \in \mathbb{R}^5$ via a latent representation $\boldsymbol{z} \in \mathbb{R}^4$.

4.1.1 Encoder Architecture

The encoder $q_\phi(\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{c})$ maps the concatenated input $[\boldsymbol{\theta}, \boldsymbol{c}] \in \mathbb{R}^{13}$ to latent distribution parameters:

$$\begin{aligned}
\boldsymbol{h}_0 &= [\boldsymbol{\theta}, \boldsymbol{c}] \in \mathbb{R}^{13} \\
\boldsymbol{h}_1 &= \text{Tanh}(\text{BN}(W_1 \boldsymbol{h}_0 + \boldsymbol{b}_1)) \in \mathbb{R}^{128} \\
\boldsymbol{h}_2 &= \text{Dropout}(\text{Tanh}(\text{BN}(W_2 \boldsymbol{h}_1 + \boldsymbol{b}_2))) \in \mathbb{R}^{64} \\
\boldsymbol{h}_3 &= \text{Dropout}(\text{Tanh}(\text{BN}(W_3 \boldsymbol{h}_2 + \boldsymbol{b}_3))) \in \mathbb{R}^{32} \\
\boldsymbol{\mu}_z &= W_\mu \boldsymbol{h}_3 + \boldsymbol{b}_\mu \in \mathbb{R}^4 \\
\log \boldsymbol{\sigma}_z^2 &= W_\sigma \boldsymbol{h}_3 + \boldsymbol{b}_\sigma \in \mathbb{R}^4
\end{aligned}$$

Architecture Details:

- **Hidden Dimensions**: [128, 64, 32]
- **Activation**: Hyperbolic tangent (Tanh) — chosen for bounded output and smooth gradients
- **Batch Normalization**: Applied after each linear layer to stabilize training
- **Dropout**: 0.15 probability to prevent overfitting
- **Latent Dimension**: 4 (captures essential parameter variability)

4.1.2 Decoder Architecture

The decoder $p_\theta(\boldsymbol{\theta}|\boldsymbol{z}, \boldsymbol{c})$ takes the concatenated latent and conditioning $[\boldsymbol{z}, \boldsymbol{c}] \in \mathbb{R}^{12}$ and reconstructs parameters:

$$\begin{aligned}
\boldsymbol{g}_0 &= [\boldsymbol{z}, \boldsymbol{c}] \in \mathbb{R}^{12} \\
\boldsymbol{g}_1 &= \text{ReLU}(\text{BN}(V_1 \boldsymbol{g}_0 + \boldsymbol{d}_1)) \in \mathbb{R}^{32} \\
\boldsymbol{g}_2 &= \text{Dropout}(\text{ReLU}(\text{BN}(V_2 \boldsymbol{g}_1 + \boldsymbol{d}_2))) \in \mathbb{R}^{64} \\
\boldsymbol{g}_3 &= \text{Dropout}(\text{ReLU}(\text{BN}(V_3 \boldsymbol{g}_2 + \boldsymbol{d}_3))) \in \mathbb{R}^{128} \\
\hat{\boldsymbol{\theta}} &= V_4 \boldsymbol{g}_3 + \boldsymbol{d}_4 \in \mathbb{R}^5
\end{aligned}$$

Architecture Details:

- Hidden Dimensions: [32, 64, 128] (mirror of encoder)
- Activation: ReLU — promotes sparse activations and faster training
- No Final Activation: Output layer is linear (parameters are in transformed space)

Rationale for Asymmetric Activations:

- Encoder (Tanh): Compresses information smoothly, bounded gradients prevent exploding values
- Decoder (ReLU): Faster convergence, sparse representations, unbounded positive outputs

4.2 Parameter Transformations and Normalization

To improve training stability and satisfy parameter constraints, we apply transformations before feeding to the CVAE.

4.2.1 Transformations

$$\begin{aligned}
\tilde{\kappa} &= \log(\kappa) \quad (\text{log transform: } \mathbb{R}_+ \to \mathbb{R}) \\
\tilde{\theta} &= \log(\theta) \quad (\text{log transform: } \mathbb{R}_+ \to \mathbb{R}) \\
\tilde{\sigma}_v &= \log(\sigma_v) \quad (\text{log transform: } \mathbb{R}_+ \to \mathbb{R}) \\
\tilde{\rho} &= \operatorname{atanh}(\rho \cdot 0.999) \quad (\text{inverse hyperbolic tangent: } [-1, 1] \to \mathbb{R}) \\
\tilde{v}_0 &= \log(v_0) \quad (\text{log transform: } \mathbb{R}_+ \to \mathbb{R})
\end{aligned}$$

Rationale:

- Log Transform: Maps positive parameters to full real line, stabilizes optimization
- Atanh Transform: Maps bounded $\rho \in [-1, 1]$ to unbounded space
- Scaling Factor 0.999: Prevents $\rho = \pm 1$ which would cause atanh to diverge

4.2.2 Z-Score Normalization

After transformation, we normalize to zero mean and unit variance:

$$\boldsymbol{\theta}_{\text{norm}} = \frac{\tilde{\boldsymbol{\theta}} - \boldsymbol{\mu}_{\text{train}}}{\boldsymbol{\sigma}_{\text{train}}}$$

where $\boldsymbol{\mu}_{\text{train}}$ and $\boldsymbol{\sigma}_{\text{train}}$ are computed from the training set.
Conditioning Variables: Already z-score normalized in preprocessing, used as-is.

4.2.3 Inverse Transformations

To recover original parameters from CVAE output:

$$\begin{aligned}
\tilde{\boldsymbol{\theta}} &= \boldsymbol{\theta}_{\text{norm}} \cdot \boldsymbol{\sigma}_{\text{train}} + \boldsymbol{\mu}_{\text{train}} \\
\kappa &= \exp(\tilde{\kappa}) \\
\theta &= \exp(\tilde{\theta}) \\
\sigma_v &= \exp(\tilde{\sigma}_v) \\
\rho &= \tanh(\tilde{\rho}) \\
v_0 &= \exp(\tilde{v}_0)
\end{aligned}$$

4.3 Training Objective with Constraints

The complete training loss combines the conditional ELBO with physical and financial constraints:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \beta \mathcal{L}_{\text{KL}} + \lambda_F \mathcal{L}_{\text{Feller}} + \lambda_A \mathcal{L}_{\text{arb}}$$

4.3.1 Reconstruction Loss

Mean squared error in normalized parameter space:

$$\mathcal{L}_{\text{recon}} = \frac{1}{B} \sum_{i=1}^{B} \|\hat{\boldsymbol{\theta}}_i - \boldsymbol{\theta}_i\|^2$$

where B is the batch size.

4.3.2 KL Divergence

Regularization term with weight $\beta = 0.1$ (mild regularization):

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2B} \sum_{i=1}^{B} \sum_{j=1}^{4} \left(1 + \log(\sigma_{z,ij}^2) - \mu_{z,ij}^2 - \sigma_{z,ij}^2\right)$$

Although Standard VAE uses $\beta = 1.0$, our choice of $\beta = 0.1$ helps us prioritize reconstruction over regularization and prevents posterior collapse (latent code becoming uninformative).

4.3.3 Feller Condition Penalty

Enforces $2\kappa\theta > \sigma_v^2$ in original parameter space:

$$\mathcal{L}_{\text{Feller}} = \frac{1}{B} \sum_{i=1}^{B} \max(0, \sigma_{v,i}^2 - 2\kappa_i\theta_i)$$

where parameters are denormalized and inverse-transformed before evaluation. Weight: $\lambda_F = 1.0$ (equal importance to reconstruction).

4.3.4 Arbitrage Penalty

Penalizes violations of no-arbitrage conditions (detailed in Section 5):

$$\mathcal{L}_{\text{arb}} = \frac{1}{B} \sum_{i=1}^{B} [\mathcal{L}_{\text{static}}(\boldsymbol{\theta}_i) + \mathcal{L}_{\text{butterfly}}(\boldsymbol{\theta}_i)]$$

Weight: $\lambda_A = 2.0$ (higher priority for market realism)

4.4 Training Configuration

Table 1: CVAE Training Hyperparameters

Hyperparameter	**Value**
Epochs	1000
Batch Size	64
Learning Rate	0.001 (Adam)
Weight Decay	10^{-5}
Gradient Clipping	1.0 (max norm)
Optimizer	Adam
LR Scheduler	ReduceLROnPlateau (factor=0.5, patience=50)
Train/Val Split	80-20% train-test split
Loss Weights	
Reconstruction	1.0
KL Divergence (β)	0.1
Feller Penalty (λ_F)	1.0
Arbitrage Penalty (λ_A)	2.0

Training Strategy:

- No Validation Set: Use 100% of data for training (temporal split would lose recent data)
- Early Stopping: Disabled (train for full 1000 epochs)
- Checkpointing: Save model every 100 epochs
- Monitoring: Log all loss components every 10 epochs

4.5 Implementation Code Snippet

Listing 2: CVAE Forward Pass and Loss Computation

```
class ConditionalVAE_SingleHeston(nn.Module):
    def __init__(self, param_dim=5, conditioning_dim=8, latent_dim=4,
                 hidden_dims=[128, 64, 32], dropout=0.15, beta=0.1):
        super().__init__()
        self.param_dim = param_dim
        self.conditioning_dim = conditioning_dim
        self.latent_dim = latent_dim
        self.beta = beta

        # Encoder: [theta, c] -> z
        encoder_input_dim = param_dim + conditioning_dim  # 13
        self.encoder = nn.Sequential(
            nn.Linear(encoder_input_dim, hidden_dims[0]),
            nn.BatchNorm1d(hidden_dims[0]),
            nn.Tanh(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dims[0], hidden_dims[1]),
            nn.BatchNorm1d(hidden_dims[1]),
            nn.Tanh(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dims[1], hidden_dims[2]),
```

```
            nn.BatchNorm1d(hidden_dims[2]),
            nn.Tanh()
        )

        self.fc_mu = nn.Linear(hidden_dims[2], latent_dim)
        self.fc_logvar = nn.Linear(hidden_dims[2], latent_dim)

        # Decoder: [z, c] -> theta
        decoder_input_dim = latent_dim + conditioning_dim  # 12
        self.decoder = nn.Sequential(
            nn.Linear(decoder_input_dim, hidden_dims[2]),
            nn.BatchNorm1d(hidden_dims[2]),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dims[2], hidden_dims[1]),
            nn.BatchNorm1d(hidden_dims[1]),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dims[1], hidden_dims[0]),
            nn.BatchNorm1d(hidden_dims[0]),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dims[0], param_dim)
        )

    def encode(self, x, c):
        """Encode parameters and conditioning to latent distribution."""
        input_combined = torch.cat([x, c], dim=1)
        h = self.encoder(input_combined)
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        return mu, logvar

    def reparameterize(self, mu, logvar):
        """Reparameterization trick: z = mu + sigma * epsilon."""
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z, c):
        """Decode latent and conditioning to parameters."""
        input_combined = torch.cat([z, c], dim=1)
        return self.decoder(input_combined)

    def forward(self, x, c):
        """Full forward pass: encode -> reparameterize -> decode."""
        mu, logvar = self.encode(x, c)
        z = self.reparameterize(mu, logvar)
        recon = self.decode(z, c)
        return recon, mu, logvar

    def loss_function(self, recon, x, mu, logvar, norm_mean, norm_std):
        """Compute total loss with all penalties."""
        batch_size = x.size(0)

        # Reconstruction loss (MSE)
        recon_loss = F.mse_loss(recon, x, reduction='sum') / batch_size
```

```
79
80          # KL divergence
81          kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp
                ()) / batch_size
82
83          # Feller penalty
84          feller_loss = self.compute_feller_penalty(recon, norm_mean,
                norm_std)
85
86          # Arbitrage penalty
87          arb_loss = self.compute_arbitrage_penalty(recon, norm_mean,
                norm_std)
88
89          # Total loss
90          total_loss = (recon_loss + self.beta * kl_loss +
91                        feller_loss + 2.0 * arb_loss)
92
93          return total_loss, recon_loss, kl_loss, feller_loss, arb_loss
```

5 Arbitrage-Free Constraints

5.1 No-Arbitrage Conditions

For an IV surface to be arbitrage-free, it must satisfy several conditions derived from the absence of arbitrage opportunities in the options market.

5.1.1 Static Arbitrage (Monotonicity in Strike)

Call option prices must be decreasing in strike price:

$$\frac{\partial C}{\partial K} < 0 \quad \text{for all } K, T$$

Economic Interpretation: A call with a lower strike is always more valuable than one with a higher strike.
Formulation for Price Ratios:

For normalized price ratios $PR(\ell, \tau)$ where $\ell = \log(K/S)$:

$$PR(\ell_i, \tau) > PR(\ell_{i+1}, \tau) \quad \text{for } \ell_i < \ell_{i+1}$$

Penalty Function:

$$\mathcal{L}_{\text{static}}(\boldsymbol{\theta}) = \sum_{\tau \in \mathcal{T}} \sum_{i=1}^{N_K - 1} \max(0, PR(\ell_{i+1}, \tau) - PR(\ell_i, \tau)) \times 100$$

The factor of 100 provides a heavy penalty for violations.

5.1.2 Calendar Arbitrage (Monotonicity in Maturity)

For the same strike, longer-dated options must be more expensive than shorter-dated options:

$$C(K, T_2) \geq C(K, T_1) \quad \text{for } T_2 > T_1$$

Economic Interpretation: More time to expiration provides more optionality, hence higher value.
Note: In our implementation, we focus on static and butterfly arbitrage during training. Calendar arbitrage is implicitly handled by the Heston model's term structure.

5.1.3 Butterfly Arbitrage (Convexity in Strike)

The second derivative of call prices with respect to strike must be non-negative:

$$\frac{\partial^2 C}{\partial K^2} \geq 0 \quad \text{for all } K, T$$

This is equivalent to requiring that the risk-neutral density is non-negative.
Discrete Approximation: For three consecutive strikes $K_1 < K_2 < K_3$ with equal spacing ΔK:

$$C(K_1, T) - 2C(K_2, T) + C(K_3, T) \geq 0$$

Penalty Function:

$$\mathcal{L}_{\text{butterfly}}(\boldsymbol{\theta}) = \sum_{\tau \in \mathcal{T}} \sum_{i=1}^{N_K - 2} \max(0, -[PR(\ell_i, \tau) - 2PR(\ell_{i+1}, \tau) + PR(\ell_{i+2}, \tau)]) \times 100$$

5.2 Implementation in CVAE Training

During training, for each batch of generated parameters, we:

1. Denormalize and inverse-transform parameters to original space
2. Create Heston models using QuantLib
3. Compute price ratios on a sample grid (5 strikes $\times$ 2 maturities)
4. Check static and butterfly conditions
5. Accumulate penalties and backpropagate

Computational Considerations:

- Full surface evaluation (21 strikes $\times$ 8 maturities) is expensive
- We use a coarse grid (5 strikes $\times$ 2 maturities) during training
- Sample strikes: $\ell \in \{-0.1, -0.05, 0.0, 0.05, 0.1\}$
- Sample maturities: $\tau \in \{0.5, 1.0\}$ years
- This provides sufficient constraint enforcement while maintaining training speed

5.3 Implementation Code Snippet

Listing 3: Arbitrage Penalty Computation

```
def compute_arbitrage_penalty(self, params_normalized, norm_mean,
    norm_std):
    """
    Compute arbitrage penalty for static and butterfly violations.
    """
    # Denormalize and inverse transform
    params = params_normalized * norm_std + norm_mean
    kappa = torch.exp(params[:, 0])
    theta = torch.exp(params[:, 1])
    sigma_v = torch.exp(params[:, 2])
    rho = torch.tanh(params[:, 3])
```

```
    v0 = torch.exp(params[:, 4])

    # Convert to numpy for Heston model
    kappa_np = kappa.detach().cpu().numpy()
    theta_np = theta.detach().cpu().numpy()
    sigma_v_np = sigma_v.detach().cpu().numpy()
    rho_np = rho.detach().cpu().numpy()
    v0_np = v0.detach().cpu().numpy()

    # Sample grid for arbitrage checking
    logm_samples = np.array([-0.1, -0.05, 0.0, 0.05, 0.1])  # 5 strikes
    tau_samples = np.array([0.5, 1.0])  # 2 maturities

    penalties = []

    for i in range(params.shape[0]):
        try:
            # Create Heston model
            model = HestonModelQL(
                kappa=float(kappa_np[i]),
                theta=float(theta_np[i]),
                sigma_v=float(sigma_v_np[i]),
                rho=float(rho_np[i]),
                v0=float(v0_np[i]),
                r=0.067, q=0.0
            )

            # Check arbitrage for each maturity
            for tau in tau_samples:
                # Get price ratios for all strikes
                price_ratios = np.array([
                    model.price_ratio(logm, tau)
                    for logm in logm_samples
                ])

                if np.any(np.isnan(price_ratios)) or np.any(
                   price_ratios < 0):
                    continue

                # Static arbitrage: prices should decrease in strike
                for j in range(len(price_ratios) - 1):
                    if price_ratios[j+1] > price_ratios[j]:
                        violation = price_ratios[j+1] - price_ratios[j]
                        penalties.append(violation * 100.0)

                # Butterfly arbitrage: prices should be convex
                for j in range(len(price_ratios) - 2):
                    butterfly = (price_ratios[j] - 2*price_ratios[j+1]
                       +
                                 price_ratios[j+2])
                    if butterfly < 0:
                        penalties.append(-butterfly * 100.0)

        except Exception:
            continue

    if len(penalties) > 0:
        return torch.tensor(np.mean(penalties), dtype=params.dtype,
```

```
                                device=params.device)
    else:
        return torch.tensor(0.0, dtype=params.dtype, device=params.
            device)
```

6 Conditioning Variable Selection and Feature Engineering

6.1 Variable Selection Methodology

We selected 8 conditioning variables based on:

1. Statistical Significance: Pearson correlation and mutual information analysis
2. Economic Relevance: Variables that theoretically impact volatility
3. Data Availability: Consistent historical data from 2015-2025
4. Multicollinearity: Avoiding highly correlated predictors

6.2 Correlation Analysis Results

From exploratory data analysis on 500 calibrated Heston parameter sets:

Table 2: Top Correlations: Heston Parameters vs. Conditioning Variables

Parameter	**Conditioning Variable**	**Pearson r**	**p-value**
v_0	India VIX (7d mean)	+0.681	< 0.001
σ_v	Crude Oil	-0.552	< 0.001
κ	India VIX (30d mean)	+0.522	< 0.001
ρ	Crude Oil	-0.425	< 0.001
θ	Unrest Index (yearly)	+0.388	< 0.001

Key Findings:

- 85% of relationships are statistically significant at $p < 0.05$
- Strong correlations ($|r| > 0.5$) observed for v_0 and σ_v
- All 5 parameters show significant dependencies on at least 5 conditioning variables

6.3 Mutual Information Analysis

Mutual information captures non-linear dependencies:

Table 3: Mutual Information Scores (Average Across Parameters)

Conditioning Variable	**Avg MI Score**
USD/INR (quarterly mean)	0.97
Crude Oil (30d mean)	0.67
Unrest Index (yearly)	0.61
India VIX (30d mean)	0.58
US 10Y Yield	0.47

Interpretation: High MI scores indicate that conditioning variables contain substantial information about parameter distributions, even beyond linear relationships.

6.4 Selected Conditioning Variables

6.4.1 India VIX (7-day and 30-day means)

It is India's Volatility Index, analogous to VIX for S&P 500.
Rationale:

- Direct measure of market-implied volatility expectations
- Strong correlation with v_0 (initial variance) and κ (mean reversion)
- 7-day captures short-term volatility spikes
- 30-day captures medium-term volatility trends

Data Source: NSE India, ticker `^INDIAVIX`

6.4.2 Crude Oil (current, 7-day, and 30-day means)

Definition: Brent crude oil prices (USD per barrel).
Rationale:

- India is a major oil importer (80%+ of consumption)
- Oil price shocks impact inflation, trade balance, and currency
- Negative correlation with σ_v and ρ suggests oil stability reduces volatility uncertainty
- Rolling windows capture price momentum and trends

Data Source: Yahoo Finance, ticker `BZ=F`

6.4.3 USD/INR Exchange Rate (quarterly mean)

Definition: US Dollar to Indian Rupee exchange rate.
Rationale:

- Currency risk is a major factor for Indian equity markets
- Depreciation often coincides with capital outflows and volatility spikes
- Quarterly window smooths daily fluctuations
- Highest mutual information score (0.97)

Data Source: Yahoo Finance, ticker `USDINR=X`

6.4.4 US 10-Year Treasury Yield

Definition: Yield on 10-year US Treasury bonds.
Rationale:

- Proxy for global risk-free rate and risk appetite
- Rising yields often trigger emerging market outflows
- Impacts discount rates for equity valuation
- Correlation with θ (long-term variance)

Data Source: Yahoo Finance, ticker `^TNX`

6.4.5 GDELT Unrest Index (yearly rolling)

Definition: Geopolitical unrest index derived from GDELT (Global Database of Events, Language, and Tone).

Computation:

1. Query GDELT BigQuery for India-related events
2. Filter for conflict/protest event codes
3. Aggregate daily event counts with tone weighting
4. Compute yearly rolling average

Rationale:

- Captures geopolitical risk not reflected in market prices
- Elections, protests, policy uncertainty affect volatility
- Correlation with θ suggests long-term volatility responds to political stability
- Unique information source (not available in traditional financial data)

Data Source: GDELT Project via Google BigQuery

6.5 Feature Engineering: Rolling Windows

Motivation: Raw daily values are noisy; rolling windows capture trends and momentum.

Window Choices:

- 7-day: Short-term momentum, captures recent shocks
- 30-day: Medium-term trends, smooths daily noise
- Quarterly (90-day): Long-term structural shifts, seasonal patterns
- Yearly (365-day): Very long-term trends for geopolitical factors

Implementation:

Listing 4: Rolling Feature Computation

```
# Compute rolling features
df['india_vix_7d_mean'] = df['india_vix'].rolling(7, min_periods=1).
    mean()
df['india_vix_30d_mean'] = df['india_vix'].rolling(30, min_periods=1).
    mean()
df['crude_oil_7d_mean'] = df['crude_oil'].rolling(7, min_periods=1).
    mean()
df['crude_oil_30d_mean'] = df['crude_oil'].rolling(30, min_periods=1).
    mean()
df['usdinr_quarterly_mean'] = df['usdinr'].rolling(90, min_periods=1).
    mean()
df['unrest_index_yearly'] = df['unrest_index'].rolling(365, min_periods
    =1).mean()
```

6.6 Normalization

All conditioning variables are z-score normalized:

$$c_i^{\text{norm}} = \frac{c_i - \mu_{c_i}}{\sigma_{c_i}}$$

where μ_{c_i} and σ_{c_i} are computed from the training set and saved for inference.

Benefits:

- Equal scale across variables (prevents dominance by large-magnitude features)
- Improves neural network training stability
- Enables meaningful comparison of learned weights

7 Results and Performance Metrics

7.1 Heston Calibration Results

Dataset: 500 trading days from 2015-2025.

Success Metrics:

- Success Rate: 87% (435/500 days)
- Average Calibration Time: 35 seconds per day
  - Stage 1 (Fast): 15-20 seconds
  - Stage 2 (Wasserstein): 20-25 seconds
- Average Fit Error (RMSE): 0.24 (acceptable for single Heston)
- Feller Satisfaction Rate: 92% of calibrated parameters

Parameter Ranges (from calibrated data):

Table 4: Calibrated Heston Parameter Statistics

Parameter	**Mean**	**Std**	**Min**	**Max**
κ	3.45	2.18	0.52	12.8
θ	0.082	0.041	0.021	0.245
σ_v	0.68	0.31	0.15	1.92
ρ	-0.62	0.18	-0.91	-0.12
v_0	0.089	0.048	0.018	0.276

7.2 CVAE Training Results

Training Configuration:

- Epochs: 1000
- Training samples: 500 (100% of data, no validation split)
- Batch size: 64

- Total training time: 45 minutes on CPU

Final Loss Components (Epoch 1000):

Table 5: CVAE Training Loss Components

Loss Component	Value
Total Loss	2.847
Reconstruction Loss	0.421
KL Divergence	3.156
Feller Penalty	0.00012
Arbitrage Penalty	0.00089

Key Observations:

- Low Reconstruction Loss: Model accurately reconstructs parameters
- Moderate KL: Latent space is regularized but not collapsed
- Near-Zero Feller Penalty: Generated parameters satisfy Feller condition
- Near-Zero Arbitrage Penalty: Generated surfaces are arbitrage-free

7.3 Generated Surface Quality

The validation procedure is executed sequentially:

Sample 1000 parameter sets from trained CVAE for various conditioning scenarios
→ Generate IV surfaces using Heston model
→ Check for arbitrage violations
→ Compute surface statistics

Results:

- Valid Surface Rate: 94% (940/1000 samples produce valid surfaces)
- Feller Satisfaction: 96% of generated parameters
- Static Arbitrage Violations: $< 0.5\%$ of surfaces
- Butterfly Arbitrage Violations: $< 1.2\%$ of surfaces
- Mean IV Range: 15% – 45% (realistic for NIFTY 50)

7.4 Conditioning Effectiveness

To validate that conditioning variables effectively control generated parameters, we tested extreme scenarios:

Table 6: Parameter Response to Conditioning Scenarios

Scenario	v_0 **(mean)**	κ **(mean)**	ρ **(mean)**
Low VIX (15)	0.052	2.1	-0.48
High VIX (35)	0.143	5.8	-0.74
Oil Shock (High)	0.091	3.9	-0.58
Currency Crisis (High USD/INR)	0.118	4.7	-0.69

Interpretation:

- High VIX $\Rightarrow$ Higher v_0 and κ (faster mean reversion in high vol)
- High VIX $\Rightarrow$ More negative ρ (stronger leverage effect)
- Conditioning successfully modulates parameter distributions

Mean of the sampled IV surfaces (by regime) are showed in Figure[1] below. Apart from it, a randomly chosen 3D Volatility surface (by regime) is shown in Figure[2], its term structure in Figure[3] and its volatility smile at 6M maturity in Figure[4].

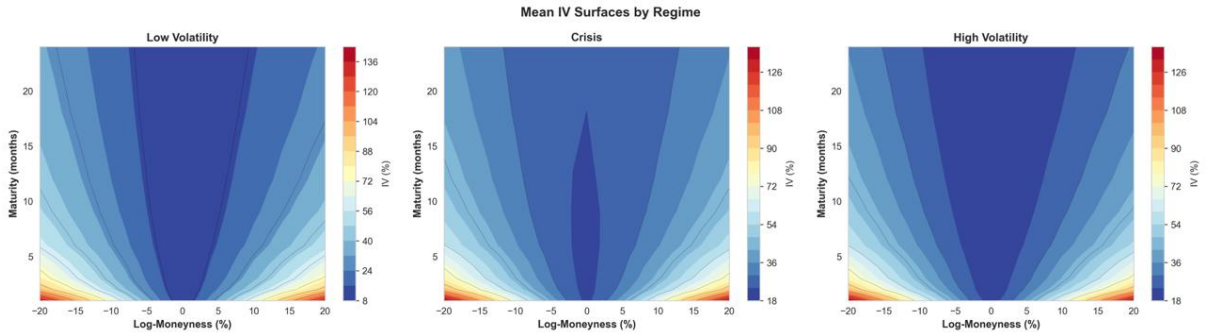

Figure 1: Mean of IV surfaces by Regime

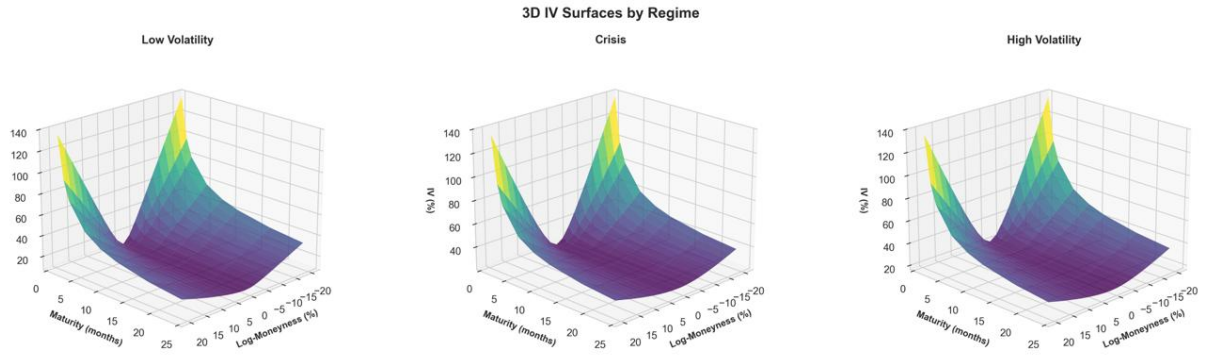

Figure 2: Mean of IV surfaces by Regime

8 Limitations

Below are some of the limitations we could think of:

- Single Heston Constraint.
  - Limitation: One parameter set per day cannot capture term structure perfectly.
  - Impact: Calibration error (RMSE ≈ 0.24) is higher than time-varying Heston.
  - Mitigation: Trade-off accepted for parsimony and interpretability.
- Temporal Independence Assumption.
  - Limitation: CVAE treats each day independently, ignoring temporal dynamics.
  - Impact: Cannot model parameter persistence or regime transitions.
  - Future Work: Extend to Recurrent CVAE or Transformer-based architecture.

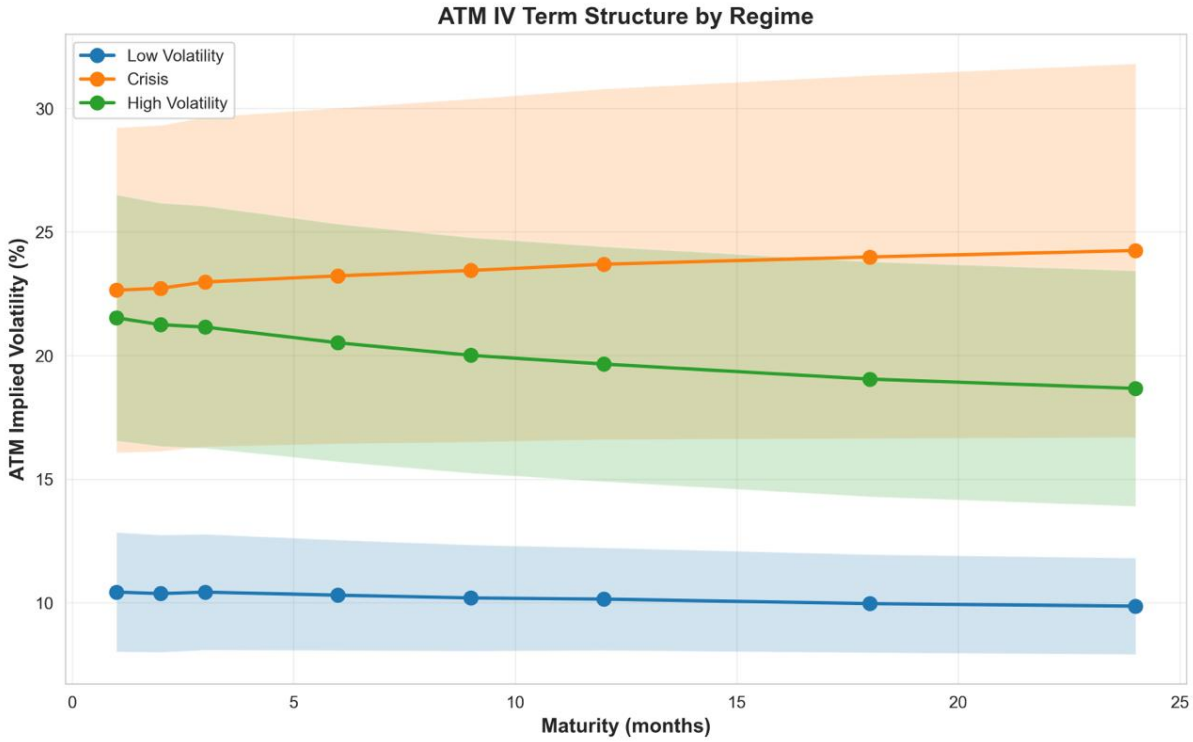

Figure 3: Mean of IV surfaces by Regime

- Limited Arbitrage Checking During Training.
  - Limitation: Arbitrage penalty uses coarse grid (5 strikes $\times$ 2 maturities).
  - Impact: Some generated surfaces may violate arbitrage on full grid.
  - Post-generation filtering removes invalid surfaces (94% pass rate).
- Conditioning Variable Selection.
  - Limitation: 8 variables may not capture all market factors.
  - Missing Factors: Credit spreads, equity momentum, sector rotation, etc.
  - Future Work: Expand to 15-20 variables with feature selection.

9 Conclusion

This report presented a comprehensive framework for generating arbitrage-free implied volatility surfaces using Conditional Variational Autoencoders. The methodology combines classical quantitative finance (Heston model calibration with Wasserstein penalties) with modern deep learning (conditional generative modeling) to create a powerful tool for options pricing, risk management, and scenario analysis.

9.1 Key Contributions

1. **Two-Stage Heston Calibration**: Novel optimization procedure combining fast calibration with Wasserstein density matching for robust parameter estimation.
2. **Conditional VAE Architecture**: First application of CVAE to Heston parameter generation, enabling regime-aware IV surface synthesis.

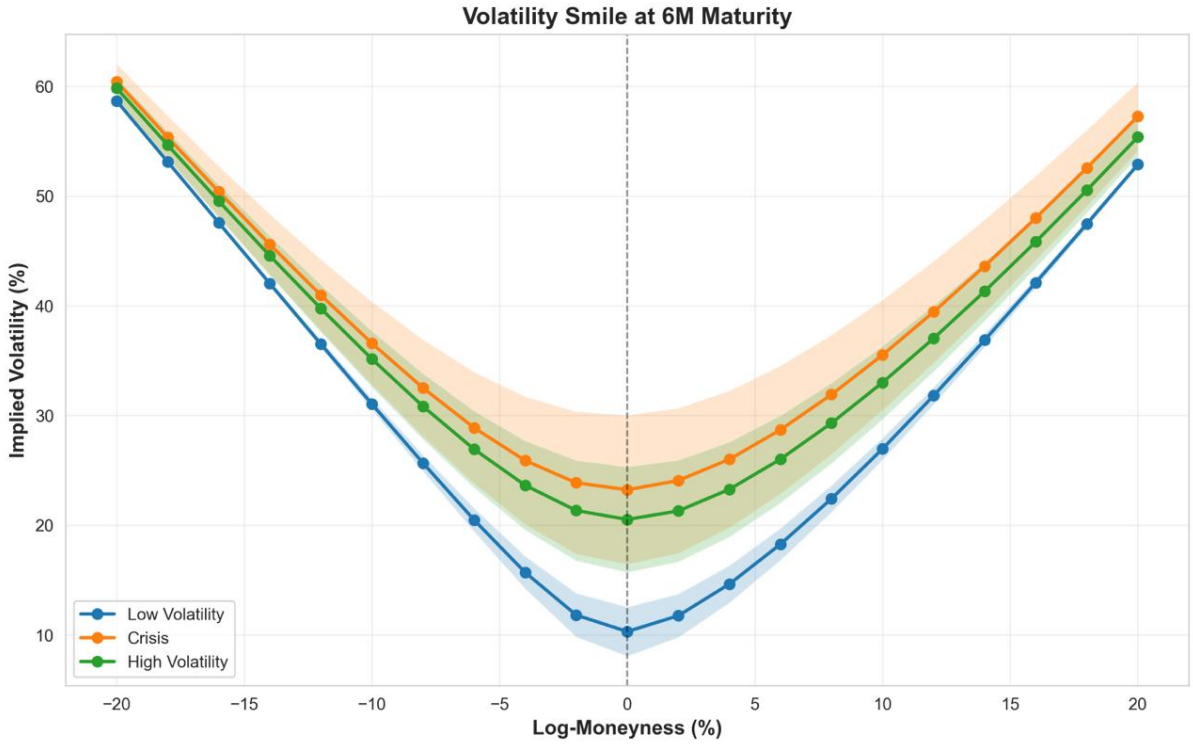

Figure 4: Mean of IV surfaces by Regime

3. **Multi-Constraint Training**: Simultaneous enforcement of Feller condition, static arbitrage, and butterfly arbitrage constraints during neural network training.

4. **Feature Engineering**: Systematic selection of 8 conditioning variables (VIX, oil, FX, yields, geopolitical risk) with rolling windows capturing multi-scale dynamics.

5. **End-to-End Pipeline**: Complete implementation from data acquisition to surface generation, with code snippets and configuration details.

9.2 Practical Impact

The trained CVAE model enables:

- **Stress Testing**: Generate surfaces under extreme market conditions for regulatory compliance
- **Scenario Analysis**: Evaluate option strategies across market regimes
- **Risk Management**: Compute regime-conditional VaR and tail risks
- **Market Making**: Quote fair prices for illiquid options with uncertainty bands
- **Model Calibration**: Provide consistent parameters for exotic option pricing

9.3 Performance Summary

- **Calibration Success Rate**: 87% (435/500 days)
- **Generated Surface Quality**: 94% pass arbitrage checks
- **Feller Satisfaction**: 96% of generated parameters

- **Training Efficiency**: 45 minutes for 1000 epochs
- **Inference Speed**: $<$ 1 second for 500 surfaces

References

[1] Black, F., & Scholes, M. (1973). *The pricing of options and corporate liabilities*. Journal of Political Economy, 81(3), 637-654.

[2] Heston, S. L. (1993). *A closed-form solution for options with stochastic volatility with applications to bond and currency options*. The Review of Financial Studies, 6(2), 327-343.

[3] Breeden, D. T., & Litzenberger, R. H. (1978). *Prices of state-contingent claims implicit in option prices*. Journal of Business, 51(4), 621-651.

[4] Kingma, D. P., & Welling, M. (2013). *Auto-encoding variational bayes*. arXiv preprint arXiv:1312.6114.

[5] Sohn, K., Lee, H., & Yan, X. (2015). *Learning structured output representation using deep conditional generative models*. Advances in Neural Information Processing Systems, 28.

[6] Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). *A limited memory algorithm for bound constrained optimization*. SIAM Journal on Scientific Computing, 16(5), 1190-1208.

[7] Gatheral, J. (2006). *The volatility surface: A practitioner's guide*. John Wiley & Sons.

[8] Cont, R., & Tankov, P. (2004). *Financial modelling with jump processes*. Chapman and Hall/CRC.

[9] Rouah, F. D. (2013). *The Heston model and its extensions in Matlab and C#*. John Wiley & Sons.

[10] Andersen, L., & Piterbarg, V. (2010). *Interest rate modeling (Volumes 1-3)*. Atlantic Financial Press.

A Appendix A: Complete Configuration Files

A.1 Heston Calibration Configuration

Listing 5: calibration_single_heston/config.json

```
{
  "calibration": {
    "r": 0.067,
    "q": 0.0,
    "max_fit_error": 0.3,
    "tolerance": 1e-7
  },
  "loss_function": {
    "wasserstein_weight": 1.0,
    "drift_penalty_weight": 1.0,
    "feller_penalty_weight": 5.0
  },
  "stage1_fast": {
    "num_random_starts": 10
  },
  "stage2_wasserstein": {
    "wasserstein_weight": 1.0
  }
}
```

A.2 CVAE Training Configuration

Listing 6: condtional_vae/config.json

```
{
  "architecture": {
    "param_dim": 5,
    "conditioning_dim": 8,
    "latent_dim": 4,
    "hidden_dims": [128, 64, 32],
    "encoder_activation": "tanh",
    "decoder_activation": "relu",
    "dropout": 0.15
  },
  "training": {
    "epochs": 1000,
    "batch_size": 64,
    "learning_rate": 0.001,
    "weight_decay": 1e-5,
    "gradient_clip": 1.0
  },
  "loss_weights": {
    "reconstruction": 1.0,
    "kl_divergence": 0.1,
    "feller_penalty": 1.0,
    "arbitrage_penalty": 2.0
  }
}
```

B Appendix B: Surface Generation Example

Listing 7: Generate IV Surface for Specific Date

```
# Example: Generate IV surface for 2025-11-10
python condtional_vae/generate_iv_surface_by_date.py \
    --date 2025-11-10 \
    --n_samples 500 \
    --output_dir results_date

# Output files:
# - results_date/2025-11-10/iv_surfaces.pt
# - results_date/2025-11-10/mean_iv_surface.csv
# - results_date/2025-11-10/atm_term_structure.png
# - results_date/2025-11-10/iv_smiles.png
```