Neural Network Models for Stochastic Differential Equations in Finance

Stochastic differential equations (SDEs) are a dominant modeling framework in quantitative finance, used for asset prices, interest rates, volatility, etc. papers.nips.cc
For example, the Heston model (a two-dimensional stochastic volatility model) defines the asset price (X_t) and variance (V_t) by an SDE system (illustrated below). Neural-network-based methods aim to learn or simulate such SDEs from data, matching the dynamics or the distribution of terminal prices. We survey recent approaches, including Neural SDEs, generative diffusion/score models, normalizing flows, GAN/MMD methods, and physics-informed neural networks (PINNs). We focus on techniques that train on simulated SDE paths (e.g. from Heston) to reproduce the terminal distribution or pricing surface, citing recent academic and industry sources.

Figure: Stochastic Volatility (Heston) model SDE.
The asset price (X_t) and variance (V_t) satisfy (Heston 1993):

$$dX_t = rX_t\,dt + X_t\sqrt{V_t}\,dW_t$$

$$dV_t = \kappa(\mu - V_t)\,dt + \eta\sqrt{V_t}\,dB_t$$

$$\langle W, B\rangle_t = \rho\,dt$$

Neural networks can be trained to replicate the evolution or terminal law of such SDEs. papers.nips.cc

Neural Stochastic Differential Equations (Neural SDEs)

Neural SDEs parameterize the drift and diffusion functions of an SDE with neural networks, effectively creating a continuous-time generative model for paths. papers.nips.cc
A neural SDE has the form:

$$dX_t = f_\theta(t, X_t)\,dt + g_\theta(t, X_t)\,dW_t,$$

where (f, g) are neural networks (often feedforward or recurrent), and (W_t) is Brownian motion. These models are universal: under mild conditions, any continuous SDE's law can be approximated by a neural SDE. arxiv.org
In finance, neural SDEs have been used for market modeling and derivative pricing. papers.nips.cc arxiv.org

Training Objectives

Training a neural SDE involves minimizing a discrepancy between the SDE’s path distribution and the observed (or simulated) data. Early methods used adversarial losses: for example, a Neural SDE generator in a Wasserstein GAN setup. papers.nips.cc
Li et al. (2019) introduced an SDE-GAN where the generator is an Euler simulation of a neural SDE and the discriminator learns path features. papers.nips.cc
However, GAN training can be unstable and suffer from mode collapse. papers.nips.cc
Recent work uses non-adversarial score-based objectives: Issa et al. (NeurIPS 2023) train a neural SDE by minimizing a signature-kernel MMD (a proper scoring rule on path space). papers.nips.cc
This method provides consistency guarantees and outperforms adversarial training on finance tasks. papers.nips.cc
Other objectives include maximum likelihood approximations (e.g. via latent SDE variational autoencoders) and hybrid losses combining price-matching with time-series likelihood. arxiv.org

Calibration to Option Data

In finance, neural SDEs are often calibrated to option prices rather than raw paths. Gierjatowicz et al. (2020) propose calibrating neural SDE drift/diffusion nets so that Monte Carlo option prices match targets. github.com
Cuchiero et al. (2024) present a Bayesian neural SDE calibration: they fit the network such that simulated call prices (generated under the neural SDE) match Heston-call prices, yielding implied-volatility bounds. arxiv.org
These works show that neural SDE parameters can be inferred to reproduce Heston-like pricing surfaces using only path or price data. arxiv.org

Architectures

The drift/diffusion networks (f_\theta, g_\theta) are usually multi-layer feedforward nets. Some methods also use residual nets to capture complex volatility dynamics.
In time-series contexts, recurrent or gated architectures (LSTM, GRU) have been used to handle path data. informs-sim.org arxiv.org
For example, Wang & Hong (2021) compare (a) RNNs with hidden state, versus (b) a 3-stage Euler-based neural SDE scheme, for pricing options. informs-sim.org
In the finance literature, ”Neural-SDE market models” (Cohen et al. 2022) decode latent factors from option data and simulate them with neural SDEs. github.com

Implementation

Practical implementation relies on differentiable SDE solvers.
Libraries like TorchSDE (GitHub, Google Research) provide PyTorch-integrated SDE solvers with GPU support and backpropagation. github.com
For example, one can define an SDE class with drift/diffusion nets in TorchSDE and call `torchsde.sdeint` to generate paths and compute gradients.

Score-Based Diffusion and Generative Models

Score-based diffusion models (also known as denoising diffusion probabilistic models) have emerged as powerful generative methods. They train a neural network to predict the score (gradient of log-density) of data under increasing Gaussian noise, and generate samples by simulating a learned reverse-time SDE. arxiv.org
In finance, such models can learn complex return distributions or time-series structures without requiring explicit likelihoods.

Model Setup

A diffusion model defines a forward "noising" SDE:

$$dX_t = f(t)X_t\,dt + g(t)\,dW_t$$

that gradually adds noise, and a reverse SDE (depending on the learned score) that recovers data from noise. arxiv.org
The neural network (often a U-Net or Transformer) is trained with denoising score matching, equivalently minimizing a variational upper bound on log-likelihood. arxiv.org

Architectures

While many diffusion models use image-based U-Nets, time-series applications use specialized networks. For example, TRADES (Berti et al., 2025) uses a Transformer-based denoising network for limit-order-book data. arxiv.org
In TRADES, each diffusion step conditions on past observations via concatenation and a self-attention network. Diffusion models can also use RNNs or temporal convolutions.

Loss Function

Training uses a simple (\ell_2) denoising loss (predicting noise), which is equivalent to learning the score function.
Unlike GANs, no adversarial training is needed, making training stable. papers.nips.cc arxiv.org

Applications

Diffusion models have been applied to generate synthetic market data.
For example, Stock Diffusion (Chang et al., 2024) uses score-based SDEs to simulate equity returns and achieves better covariance estimates than standard Monte Carlo. arxiv.org
TRADES (Berti et al., 2025) uses a Transformer diffusion to simulate realistic limit-order-book sequences. arxiv.org

The generated data passes statistical tests and preserves stylized facts such as autocorrelations.
These models can potentially generate terminal asset-price distributions for complex SDEs by learning from sample paths.

Score Matching & Fokker-Planck

Recent work also integrates physics into score-based methods.
For instance, Score-PINN (Yuan et al., 2024) uses score matching to estimate the score function and then solves the Fokker-Planck (Kolmogorov forward) PDE via a PINN. arxiv.org
This approach could, in principle, learn the distribution of (X_T) by solving the FP equation informed by data.
While not yet applied to Heston in literature, it demonstrates a deep link between generative scores and SDE PDEs. arxiv.org

Normalizing Flows and Other Generative Models

Normalizing flows offer an alternative: they learn an invertible transformation from a simple base (e.g. Gaussian) to the target distribution, enabling exact likelihoods.
For time-series or SDEs, one can use continuous-time flows or neural ODE/SDE flows.

Time-Changed Flows

El Bekri et al. (2023) propose Time-Changed Normalizing Flows (TCNF), which warp time in a Brownian motion to effectively model SDE dynamics.
They show TCNFs can capture SDEs like the Ornstein–Uhlenbeck (and by extension Heston-like) models that standard continuous flows struggle with. arxiv.org
The training loss is the usual flow log-likelihood on path ensembles.
This approach yields flexible SDE models with exact density, but is still nascent in finance applications.

GAN/MMD Models

Prior to diffusion models, some works used GANs or MMD for financial paths.
For example, Wan et al. (2024) train an RNN generator with a signature-kernel MMD loss, matching path distributions to historical data. arxiv.org
They view this as a "score-based" method since the kernel score acts like a discriminator.
Their experiments with Heston-generated data show that using a noise process with realistic variance (e.g. stochastic volatility noise) improves sample realism (capturing volatility clustering) compared to i.i.d. Gaussian noise. arxiv.org
Another recent idea is the Latent SDE (Li et al., 2020), a variational flow where both prior

and posterior are SDEs, trained via ELBO (combining likelihood and KL divergence). github.com

Key Training Losses

In summary, generative models typically use one of the following training objectives:

- **Maximum likelihood** (e.g. flows, latent SDEs) to fit densities.
- **Adversarial losses** (e.g. GAN/WGAN) to match distributions. papers.nips.cc
- **Kernel losses** (e.g. MMD with signature kernel). arxiv.org
- **Score matching** (e.g. diffusion, implicit flows).
- **PDE residuals** (if physics-informed).

Each has trade-offs:

- GANs can model arbitrarily complex distributions but may collapse.
- Flows ensure exact density but may require tractable transforms.
- Score/diffusion models trade off sample quality and compute time.

In finance, hybrid methods are common—for example, fitting a generator so that Monte Carlo option prices (expectations under the model distribution) match market targets. arxiv.org

Physics-Informed and PDE-Based Approaches

Physics-informed neural networks (PINNs) use known SDE/PDE structure to guide learning.
In finance, this often means solving the Fokker–Planck equation or the backward Feynman–Kac PDE for option values.

PINNs for Option Pricing

Many works use PINNs to solve the option pricing PDE from the Feynman–Kac theorem. Hainaut et al. (2024) demonstrate a PINN that directly learns the European option price surface in the Heston model. detralytics.com
The network takes inputs ((t, s, v)) and outputs (V(t, s, v)); it is trained to satisfy the Heston PDE (Fokker–Planck backward form) and boundary/terminal conditions.
Its loss is the PDE residual:

$$|\partial_t V + \mathcal{L}V|^2$$

evaluated at random sample points, plus penalties enforcing (V(T,s,v)=\max(s-K,0)), etc. They find PINNs can replicate FFT-based Heston prices to good accuracy with relatively few parameters.

Key insight: PINNs embed the physics of the SDE (the differential operator (\mathcal{L})) directly into learning, improving data efficiency.

Fokker–Planck Equation

Another approach is to learn the distribution by solving the forward Kolmogorov (Fokker–Planck) PDE.
Score-PINN (Yuan et al., 2024) first learns the score function (gradient of log-density) via score matching, then solves the Fokker–Planck–Levy PDE as a PINN using the learned score. arxiv.org
In principle, one could target (p(x,T)), the distribution of (X_T), by solving:

$$\partial_t p = \mathcal{L}^* p$$

via a neural network. These methods are still emerging in finance but offer a principled way to match distributions from SDEs.

Deep BSDE (Backward SDE) Methods

Related techniques like the Deep BSDE method (Weinan E et al.) treat PDEs via backward SDE formulations, using neural nets to approximate the PDE solution and its gradient. These have been applied to high-dimensional option pricing but are more common in risk-limit learning frameworks than in explicit SDE learning.

Architectures and Loss Functions

Network Architectures

The network architectures in these models are varied:

- **Feedforward (FNN/MLP)**:
  Common in PINNs and neural SDE drift/diffusion networks (detralytics.com).
  Residual (skip-connection) MLPs often improve training stability.

- **Recurrent (RNN/LSTM)**:
  Used for sequential data, especially in GAN/MMD generators (arxiv.org, arxiv.org).
  For example, the signature-MMD model uses an LSTM to generate price paths (arxiv.org).

- **Transformers**:
  Emerging in finance diffusion models.
  TRADES employs a Transformer with self-attention layers to model order-book time series (arxiv.org).
  Transformers can capture long-range dependencies better than RNNs.

- **Continuous/Neural CDEs**:
  Some works use neural controlled differential equations (Neural CDEs) to model

continuous-time series.
These can also serve as discriminators or generators in GANs (github.com).

Loss Functions

The loss functions depend on the modeling goal:

- **Distribution Matching**:
  Wasserstein (WGAN), Jensen–Shannon (GAN), or kernel-MMD losses are used to align generated vs. real distributions
  (papers.nips.cc, arxiv.org).
  Signature-kernel MMD loss is a "proper score" on path space, offering stable training (arxiv.org).

- **Score Matching**:
  Diffusion models use denoising score-matching loss, equivalent to minimizing
  (\mathbb{E}[|\nabla \log p - s_\theta|^2])
  (arxiv.org).
  This provides an unbiased (up to constants) estimate of the data score.

- **Maximum Likelihood**:
  Used in flows and latent SDEs to optimize (approximate) log-likelihood of observed paths.
  Often requires stochastic variational bounds.

- **Adversarial**:
  GANs use minimax adversarial losses to match distributions (papers.nips.cc).
  Neural SDE GANs train by backpropagating through the SDE simulation and an RNN discriminator (github.com).

- **PDE Residuals**:
  PINNs use physics-informed losses: mean squared error of PDE residuals plus boundary/terminal condition penalties (detralytics.com).
  For example, the Heston PINN minimizes the squared Fokker–Planck (or Black–Scholes) operator error at random collocation points (detralytics.com).

Application to Heston and Stochastic Volatility Models

Many of these techniques have been tested on or motivated by Heston-like models:

- **Neural SDE Calibration**:
  The robust neural SDE code (`msabvid/robust_nsde`) targets Heston data.
  It trains neural volatility models so that simulated call prices match Heston prices (github.com).
  This demonstrates fitting a neural network model to a known SDE's outputs.

- **Generative Heston Paths**:
  In Wan et al.'s signature-MMD study, training data is generated by the Heston model.
  They train RNN generators (with MMD loss) to reproduce the distribution of log-returns,
  finding that using a noise input with volatility dynamics yields samples exhibiting realistic volatility clustering (arxiv.org).

- **PINNs on Heston PDE**:
  As noted, PINNs have been applied to compute Heston option prices (detralytics.com).
  The trained networks reproduce FFT-based Heston prices with small errors,
  and work across strikes/maturities without re-calibration.

- **Neural SDE Market Models**:
  The `neuralSDE-marketmodel` repo (Cohen et al.) uses a Heston stochastic-local-volatility (Heston-SLV) model to synthesize option data.
  They then decode latent factors and train neural SDEs to simulate those factors,
  achieving reasonable risk estimates for option books (github.com).

- **Score-based Synthesis**:
  Though not yet applied to specific SDEs in literature, score-based diffusion models (e.g., [65]) can in principle learn the terminal distribution of any SDE by training on path endpoints.
  One could simulate many Heston terminal prices and train a diffusion model to reproduce that distribution.
  This remains an area of active research.

Open-Source Tools and Libraries

Several open-source projects and codebases implement these ideas:

- **TorchSDE (Google Research)**:
  A PyTorch library for differentiable SDE integration (github.com).
  Provides GPU-enabled SDE solvers with backpropagation support, ideal for neural SDE modeling.

- **sigker-nsdes (GitHub)**:
  Code accompanying Issa et al. (NeurIPS 2023) for training Neural SDEs using signature kernel scores.
  Implements path kernel scores and adjoint methods for efficient training.

- **msabvid/robust_nsde**:
  Implements methods from Gierjatowicz et al. (2020) to calibrate neural SDEs to option prices.
  Includes scripts (`nsde_LV.py` , `nsde_LSV.py`) that fit volatility networks to Heston-generated call prices (github.com).

- **neuralSDE-marketmodel (Cohen et al.)**:
  Python modules and notebooks for “Arbitrage-free Neural-SDE Market Models” (ICML 2022).
  Includes examples of using Heston-SLV option data to train neural SDEs for latent dynamics (github.com).
- **DeepMarket (Berti et al.)**:
  Framework for transformer-based diffusion models applied to limit order books.
  Public code for TRADES model and synthetic LOB data is available (GitHub: LeonardoBerti00/DeepMarket).
- **Score SDE Code (Song et al.)**:
  Yang Song’s official code for score-based generative modeling (Score SDEs) is available
  and can be adapted for financial applications (GitHub: yang-song/score_sde).
- **PyTorch/Numpy / General ML Tools**:
  Libraries such as DeepXDE, PyTorch, and NumPy are commonly used to implement PINNs, PDE losses, and flow-based models.

These tools provide strong foundations for researchers training neural SDEs or diffusion models on financial simulations.

Summary of Approaches

The following table summarizes neural network approaches for SDE modeling in finance. It highlights the method category, typical network architecture, training objective/loss, applications (e.g., Heston or financial time-series data), and key references.

Approach	Neural Architecture	Training Loss / Objective	Application / Example	References
Neural SDE Calibration	Feedforward nets (drift/diffusion); RNN/LSTM for path history	Distance between path distributions (Wasserstein, signature MMD) or price calibration loss	Learn SDE parameters from simulated paths or match option prices (e.g., Heston)	informs-sim.org, arxiv.org, github.com
Score-Based Diffusion	Denoising net (U-Net / Transformer) on time series	Denoising score-matching loss (equiv. to variational likelihood)	Generate realistic market time-series; e.g., stock or LOB simulators (TRADES)	arxiv.org
Normalizing Flow	Invertible / dynamic flows (e.g., TCNF)	Exact log-likelihood of observed paths	Model SDE marginals or sample paths with exact density (e.g., OU, Heston)	arxiv.org

Approach	Neural Architecture	Training Loss / Objective	Application / Example	References
GAN / Adversarial	Generator (RNN/CNN) + Discriminator; Neural CDE discriminator	Wasserstein or Jensen–Shannon loss	Time-series generation via adversarial training (e.g., Neural SDE GAN)	papers.nips.cc, github.com
MMD / Signature Kernel	RNN/LSTM generator; signature feature extractor	Kernel MMD between path distributions (signature kernel)	Match return/distribution statistics (e.g., volatility clustering)	arxiv.org
Physics-Informed (PINN)	Feedforward net (or ResNet) over (t, S_t, V_t)	PDE residual + boundary condition loss (e.g., Fokker–Planck or Feynman–Kac)	Solve option pricing PDE under Heston; compute distribution via FP	detralytics.com

Commentary

Each approach has its strengths:

- **Neural SDEs** model continuous-time dynamics and can incorporate no-arbitrage constraints.
- **Diffusion models** excel at high-fidelity time-series generation.
- **Normalizing flows** provide exact likelihood and tractable density.
- **PINNs** enforce known SDE/PDE structure via physics-based losses.

Hybrid methods (e.g., combining score matching with PINN objectives) can integrate the strengths of multiple approaches (arxiv.org).

Key Points:

- **Common architectures**: MLPs, RNNs/LSTMs, and increasingly Transformers for sequential modeling.
- **Loss functions**: Range from MSE/PDE residuals (detralytics.com) to adversarial or kernel-based divergences (papers.nips.cc, arxiv.org).
- **Applications**: Primarily in pricing and calibration — e.g., option pricing with Heston models (detralytics.com), volatility modeling (arxiv.org), and market simulation (arxiv.org).

All approaches rely on simulated data (e.g., Heston paths or market returns) to learn either the SDE itself or its implied distribution at maturity.

Sources:

We surveyed recent literature (2020–2025) on neural SDE modeling. Representative works include:

- **Neural SDE calibration**: Cuchiero et al., Gierjatowicz et al. (github.com)

- **Signature-kernel training**: Issa et al. (papers.nips.cc)
- **Signature-MMD generators**: Wan et al. (arxiv.org)
- **Diffusion models**: Berti et al., Chang et al. (arxiv.org)
- **PINNs**: Hainaut et al. (detralytics.com)

Open-source implementations include **TorchSDE** and multiple GitHub repositories (github.com) that support these models.