

# Hyper-Network-Based Neural Approximators for Lifecycle Derivative Pricing on Low-Dimensional Manifolds

Jonathan Rosen, Andrew McClelland, Serguei Issakov



# Hyper-Network-Based Neural Approximators for Lifecycle Derivative Pricing on Low-Dimensional Manifolds

Jonathan Rosen<sup>a</sup>, Andrew McClelland<sup>a</sup>, Serguei Issakov<sup>a</sup>

<sup>a</sup>*Numerix, New York, NY, USA*

---

## Abstract

We present a hyper-network-based deep learning approach for fast, accurate lifecycle pricing of complex derivatives. A supervised volatility-surface encoder compresses one or more implied-volatility surfaces into low-dimensional latent features, and contract lifecycle states. These together condition a Hyper-Network Block (HNB) that generates the parameters of a compact pricing network. This replaces repeated high-dimensional Monte Carlo (MC) simulations, adapts to evolving market regimes, and generalizes across events such as barrier activation or early redemption without retraining. It also supports low-latency online fine-tuning for new scenarios. In tests on single-name and basket autocallable derivatives under Black-Scholes, local-volatility, and hybrid models, the approach matches MC accuracy within simulation variance while delivering orders-of-magnitude inference speedups. The architecture enables high-throughput XVA calculations and real-time pricing, offering a practical, scalable alternative to conventional Monte Carlo pricing workflows.

**Keywords:** Derivative pricing, Neural networks, Hyper-networks, Autocallable derivatives, Monte Carlo simulation, Volatility surfaces, Real-time pricing, Financial machine learning, Lifecycle modeling, Deep learning

---

## 1. Introduction

Derivative pricing is a cornerstone of modern finance, providing crucial insights into the value of complex financial instruments. Accurate and efficient pricing methods are critical, especially in real-time trading environments where speed and precision are paramount. Credit valuation adjustment (CVA) and other valuation adjustments (XVA) require efficient pricing of derivatives on large numbers of forward scenarios to accurately account for counterparty risks and liquidity costs [1, 2]. XVA calculations typically require complex exposure modeling and high-dimensional state variable simulations. While traditional methods such as Monte Carlo simulations can account for these factors, they often become computationally expensive and less efficient when the number of dimensions increases [3].

The act of replacing Monte Carlo pricers with faster calculation strategies is a potential solution to the problems posed under high-throughput and real-time calculation environments. Artificial neural networks (ANN) have been suggested for this problem for a number of years, and there have been comprehensive studies on the use of neural networks for option pricing and hedging, including how ANNs can be trained to approximate the results of Monte Carlo pricers [4, 5]. The training process is highly parallelizable in general, and can be applied to a wide range of derivative products. After the training phase, these models offer extremely fast evaluation, which can significantly reduce the time needed for pricing.

Standard ANNs here refer to (possibly deep) feedforward networks that are used for learning and approximating

relationships in training data of interest. ANNs introduce non-linearity through activation functions applied to the outputs of neurons at each layer. These networks learn non-linear mappings from inputs to outputs by adjusting the weights and biases during training. ANNs are effective at approximating non-linear relationships in simple or moderately complex problems, but as the complexity of the task increases, they typically require deeper architectures and longer training times to capture the relationships in the training data, making them less efficient for highly complex problems.

ANNs are typically designed for stationary relationships, where the underlying patterns do not change over time or across inputs. Once trained, their weights are fixed, making adaptation to nonstationary or context-dependent relationships—such as evolving market regimes or lifecycle events—cumbersome. Early redemption, knock-ins, and other path-dependent lifecycle features often require retraining or architectural changes in static ANNs, limiting their practicality.

Alternatively, the concept of hyper-networks (HN) has emerged as a powerful technique in deep learning [6]. HNs are an alternative ANN architecture, which dynamically generate the weights and bias parameters for another ANN, called the target network (TN). These parameters are then used to make predictions in the TN. This allows HNs to model the weights of the TN as a function of the HN inputs. These can take the form of extrinsic control variables, such as task-type or lifecycle event indicators, or directly from the market data itself (data-conditioned HN). By learning

to generate dynamic weights for the TN based on the HN inputs, HNs can better capture complex, contextual non-linearities that vary with input conditions. Additionally, the use of a gated deep-network architecture accelerates learning by selectively activating relevant features, leading to faster convergence and improved generalization.

In application to derivative pricing, hypernetworks have been recently introduced as an efficient way to accelerate the calibration of mathematical finance models by generating proxy networks which replace standard derivative pricers and models [7]. Hypernetworks can also be applied where price non-linearity can evolve depending on market conditions or product features (e.g. knock-in options), since hyper-networks can adapt the weights dynamically to better capture the specific nature of the problem. Hyper-networks are well suited for capturing complex lifecycle events (e.g., early redemptions, knock-ins, knock-outs) without requiring model redesign. They can naturally account for such events within their architecture, as they are designed to learn patterns from past data, including how events and triggers affect pricing.

While prior works have surveyed the use of artificial neural networks in option pricing [4], these studies typically emphasize static model architectures or fixed volatility regimes. Similarly, deep hedging approaches have demonstrated strong results for path-dependent payoffs under friction [5], yet often require large network sizes and training durations. Our work differentiates itself by leveraging hyper-networks to dynamically encode market structure and product lifecycle information with compact architectures and efficient training and inference.

In this work, we introduce a novel hyper-network-block (HNB) artificial neural network architecture designed for the pricing of autocallable derivatives. The model incorporates a volatility-surface encoder and is trained using standard Monte Carlo simulations, with a training procedure robust to heteroskedasticity in the generated data distributions. Although the primary focus is on autocallables, the proposed methodology is broadly applicable to a variety of derivative instruments.

The remainder of this paper is structured as follows. In Section 2, we present a methodology detailing the HNB architecture, the volatility-surface encoding scheme, and the training framework. Section 3 reports the main numerical results, evaluating the model’s ability to reproduce price distributions under various market conditions. Section 4 highlights the real-time application of the HNB, emphasizing its inference speed and adaptability to evolving market data via online fine-tuning. Conclusions are drawn in Section 5, emphasizing the advantages of our approach in terms of scalability, speed, and accuracy. Technical details, including the extension of the method to multi-asset basket autocallables under hybrid local-volatility models, are provided in the appendix.

Throughout the paper, we illustrate the flexibility and effectiveness of our model through examples that progressively move from simple Black-Scholes dynamics to local

volatility settings and multi-underlying structures. This progression demonstrates the scalability of our approach and its capability to capture the evolution of market variables necessary for real-time, high-accuracy derivative pricing.

## 2. Methodology: HNB Architecture and Training Framework

### 2.1. Model Specification

To achieve an accurate and efficient regression model that mimics the output of a derivative pricer with high-dimensional inputs, we use a simple yet powerful ANN regression model. We explicitly consider the case of equity autocallables, however our network design is generally applicable to any Monte Carlo-based pricer for nearly any type of derivative in any asset class. In the case of equity autocallables, market-driven inputs include spot prices, interest rate curves, volatility surfaces, and lifecycle indicators like barrier events. Additionally, there are issue-specific inputs, such as coupon schedules, coupon rates, and barrier levels, which do not depend directly on financial market data.

Figure 1a presents the model architecture, highlighting the hyper network block (HNB) and volatility-surface encoder components. It is based on the HNB architecture, which enhances standard feed-forward models by introducing a HN that generates the weights and biases of the TN. ReLU activation functions are used throughout the HNB [8]. As described in Appendix Appendix A, this architecture achieves a faster learning rate and improved regression performance for the complex non-linear behaviors of the price targets when compared to standard feed-forward ANNs. The HNB is also able to adapt the regression to different regimes of the pricer, such as before and after barrier events are triggered.

The TN is kept small in our HNB, using only a single layer to minimize the number of parameters. This decision is made to limit the size of the target network due to quadratic scaling of the number of required HN parameters. The target network expands the input space by 2-3x, followed by a final single-layer feed-forward ANN (reducer) applied to HNB latent space to bring it back to the original size. This allows multiple HNBs to be stacked for deep learning. The reduction step within the HNB uses static parameters, in contrast to the prior TN stage, where the HN generates the TN parameters dynamically. In practice this adds sufficient dynamics to the generation of the HNB latent space, which can then be statically reduced to its original size.

The HNB component is central to our model, and multiple HNBs can be stacked for deep learning applications. While we use a 4-layer HNB throughout, we find that varying the number of layers often produces similar results due to the self-selecting depth achieved through gating. Each layer in the HNB may include a learnable scalar gate that modulates the contribution of the input versus the transformed output. Specifically, for hidden layer input  $h_{in}$  and

output  $h_{\text{layer}}$ , the final activation  $h_{\text{out}}$  is computed as in Equation 1 where  $\lambda$  is a trainable parameter initialized at 0.5.

$$h_{\text{out}} = \lambda \cdot h_{\text{in}} + (1 - \lambda) \cdot h_{\text{layer}}, \quad \lambda \in [0, 1] \quad (1)$$

This architecture facilitates smoother convergence by dynamically adjusting the effective depth of the network during training. This gating mechanism accelerates training and improves model accuracy by allowing the model to self-optimize the depth of the HNB layers. Figure 1b demonstrates how gating accelerates training by producing stronger gradients and improving final model accuracy.

To capture the effects of entire volatility surface dynamics for one or more volatility surfaces, an encoder ANN is directly connected to the first HNB layer. The encoder takes the full set of implied-volatilities as input, and reduces them to a 2-dimensional latent space. Volatility surfaces, though high-dimensional in raw form, effectively constrain admissible volatility surface shapes to a lower-dimensional manifold due to arbitrage constraints and structural smoothness [9]. This motivates the use of a learned low-dimensional encoding.

## 2.2. Model Training

We train the model with the market-driven inputs used in traditional calibrated models used to perform Monte Carlo simulations of the equity price processes to generate autocallable prices. For simplicity, interest rates are treated as deterministic, though this assumption can be relaxed without loss of generality. Market data, including spot prices, interest rates, and volatilities, are sourced from historical time series and augmented with random noise to expand the range of inputs representing possible market scenarios. Volatilities that corresponded with vertical or calendar arbitrage are omitted. More than 600k training data points are generated, covering a wide range market conditions.

The mean-square-error (MSE) loss function is used throughout for the training objective, by comparing Monte Carlo prices to ANN model predictions. Each Monte Carlo simulation is divided into 10 smaller simulations, each using 10k paths, enabling us to estimate the mean and variance of each data point and compare it with the model’s output to ensure accuracy. Training is conducted on a small NVIDIA RTX A1000 Laptop GPU with 6GB of VRAM at a negligible cost. Based on total training time estimates, as well as currently available cloud resources, we estimate the total cost of model training at under USD\$100. This cost refers only to the model training phase and excludes the offline data generation using Monte Carlo simulations. For the latter, computational cost is typically amortized across many pricing scenarios and can be parallelized extensively.

Our model is designed to capture the effects of market-driven inputs and is tailored to a specific term sheet valid throughout the life of the instrument. The model utilizes a small number of parameters, making it fast to

train—approximately one hour on low-cost hardware to cover all potential future market scenarios. This design allows for multiple HNB models, each capturing different inputs, to be trained separately, reducing the overall data and compute time required. Consequently, the approach is cost-effective, achieving low training costs without sacrificing accuracy.

Empirical experiments shows that increasing the latent dimension to 3D introduced minor overfitting and reduced generalization performance (see Fig. B.7 and Appendix Appendix B). This suggests that the volatility surfaces possess a compact underlying structure, which aligns with the view from manifold learning methods [10]. The encoder is trained in parallel with the HNB, which is analogous to partial-least-squares (PLS) [10, 11], a linear technique for dimensionality reduction. By using a non-linear encoder with activation functions, we generalize PLS for the input feature space generation which simultaneously optimizes the HNB’s explanatory power of the regression target.

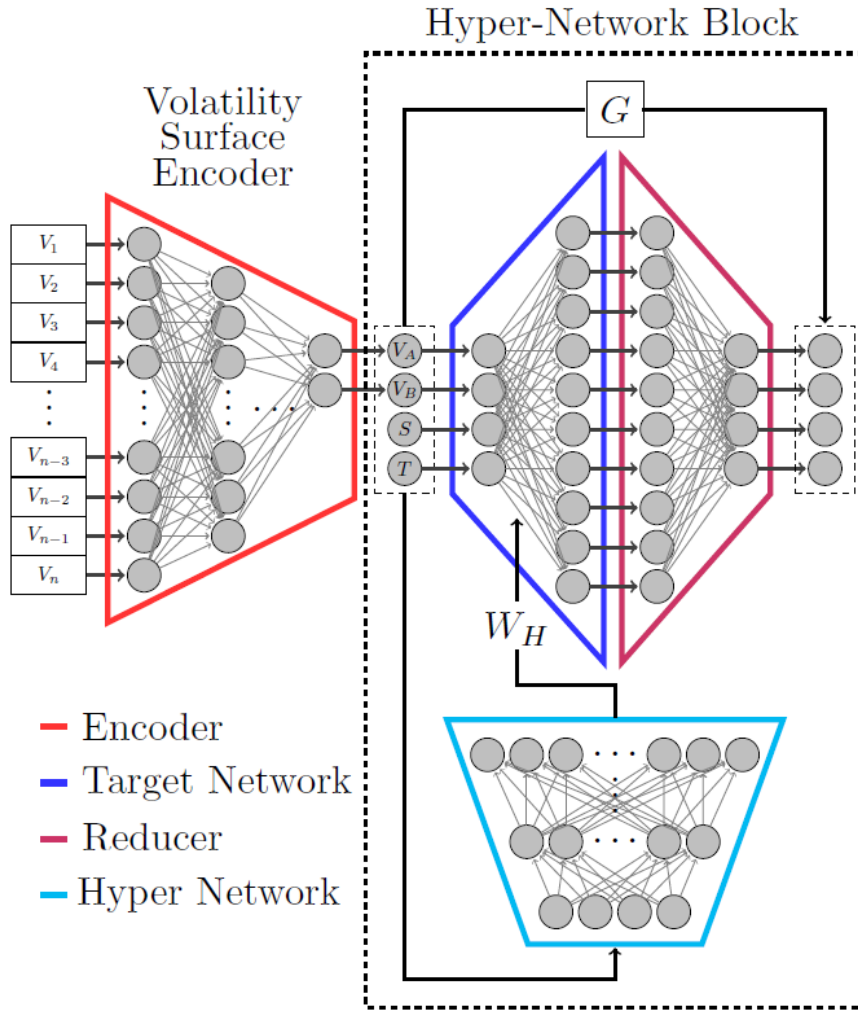
We also evaluate autoencoder techniques, a common unsupervised alternative to encoder–regression for dimensionality reduction, often used to simplify regression models, improve generalization, and reduce training costs [12]. However, autoencoders prioritize input reconstruction over prediction, which can introduce latent dimensions irrelevant to the pricing task. In contrast, encoder-regression directly optimizes explanatory power of latent features toward pricing outputs, thus offering superior bias-variance trade-off in supervised regimes [10, 11].

Figure 1c. shows the encoder-regression approach achieves faster convergence and improved pricing accuracy compared to autoencoder designs. In the end we have chosen to use a supervised encoder-regression rather than an unsupervised autoencoder, as our goal is to optimize the encoder for explanatory power with respect to pricing accuracy, not reconstruction of the input features.

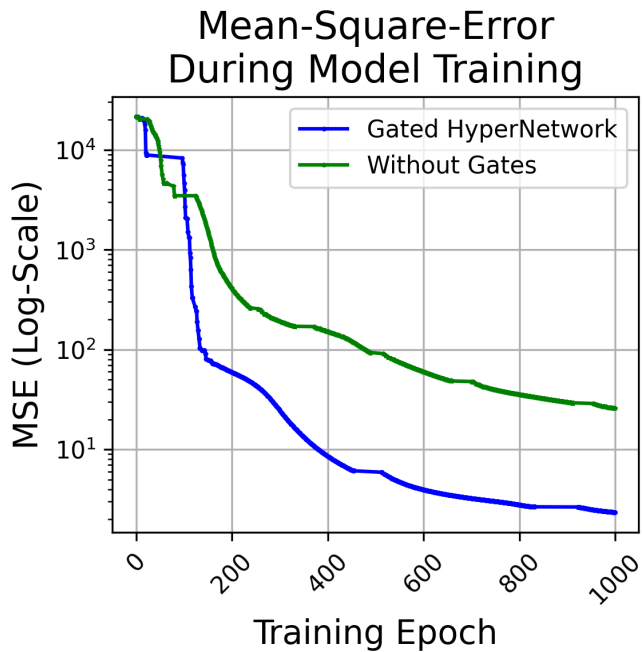
## 3. Main Results

The objective of our model is to accurately reproduce the derivative price as it evolves under changes in the market-driven inputs, and to simultaneously capture the impact of lifecycle events. We assess the model’s performance by comparing with the distribution of test data. Specifically, the test data is an independent data set generated by Monte Carlo simulation, taking the form of a series of ten sub-simulations with identical inputs, each with 10k paths but different seeds for pseudo-random number generation. The range of prices obtained from the sub-simulations is used to determine the precision of the generated price data in order to facilitate comparison to the HNB model predictions.

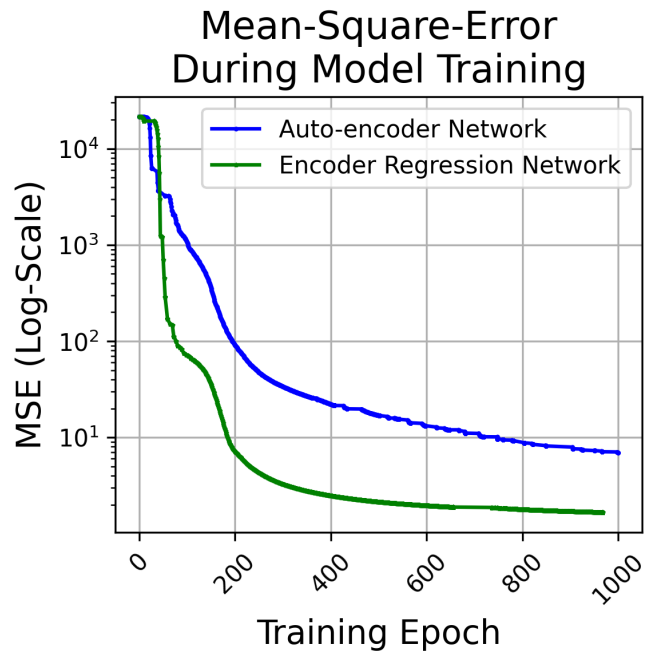
In the first instance, our model is trained on single-equity autocallable prices generated under the Black-Scholes model. The autocallable pricer makes quarterly looks at the underlying equity price, which, if above the initial spot price on these dates, results in a fixed cash payout. In



(a) Proposed model architecture incorporating a hyper network block (HNB) and volatility-surface encoder.



(b) Comparison of training dynamics for HNBs with and without gating. Gating improves learning speed and final model accuracy.



(c) Comparison between encoder-regression and autoencoder approaches for volatility surface compression. Encoder-regression achieves faster convergence and superior pricing accuracy.

Figure 1: Overview of the proposed model architecture and learning dynamics.

addition a continuously monitored barrier at 92% of the initial spot price, if triggered, results in conversion to a short put struck at the initial spot price.

As the Black-Scholes pricer only uses a single volatility as input, a volatility-surface encoder is not required, and the spot price, interest rates, and volatility are directly used as inputs to the HNB. The model captures all prices to high accuracy, and consistently within the variance of the training simulation points. Fig. 2a shows predicted and simulated prices of an autocallable derivative under a Black-Scholes model with 20% implied volatility, illustrating both pre- and post-barrier trigger states. The autocallable prices are shown as a function of the equity spot price. Two branches are depicted, consistent with both states of the indicator input for the barrier trigger state of the instrument. After the barrier is triggered, the lifecycle event converts the autocallable to a short equity put, and this transition is seamlessly captured by the HNB without needing to replace or re-train the model.

Inspection of Fig. 2a shows good agreement between the HNB and the Monte Carlo prices, though Fig. 2b reveals minor deviations between the HNB predictions and Monte Carlo prices after the barrier trigger event. In this region the HNB predictions oscillate around the Monte Carlo values, which are tightly bounded and essentially deterministic. Despite the total deviation size being small for most practical purposes, it is still straightforward to improve upon this deviation using data augmentation. Fig. 2c shows how targeted data augmentation improves prediction accuracy near critical spot price regions, reducing local oscillations. The result is to reduce deviations without reducing accuracy elsewhere. In theory, some balance should be struck between local accuracy limits and global accuracy when augmenting the training data, since it can affect the balance of training effectiveness across different regions of input space, though in practice it is straightforward to gradually add more training points to important regions when deviations are observed.

Going beyond the Black-Scholes model, we examine the input-scaling for the HNB when applied to entire volatility surface pricers using local-volatility dynamics, in the case of the single-underlying equity autocallable under the local-volatility model. In this application the encoder processes 96 implied-volatilities across a fixed range of strikes and expiries. The encoder generates a two-dimensional latent space, which is input into the HNB in addition to the spot price and interest rate market data. Appendix Appendix B and Fig. B.6 present heatmaps of the HNB’s pricing predictions across the latent space under varying spot prices and barrier states. The simultaneous training of the encoder with the HNB generates results as shown in Fig. 3a. This demonstrates excellent agreement for a simulated time-series of the evolution of market volatilities, interest-rates, and spot prices, for a simulated time-series of market inputs that is independent of the training data.

In addition, a total of 200 time-series are generated, using the same data generation technique as the training

data (with different random seed) and for each time-point the HNB predictions are compared with the generated test data. Fig. 3b presents a scatter plot comparing HNB predictions with Monte Carlo prices across 200 independent simulated time series, showing strong correlation and generalization.

Midpoint-shifted deviations are shown in Fig. 3c (before barrier trigger) and 3d (after barrier trigger) for the time-series corresponding to Fig. 3a, with all predictions staying within Monte Carlo variance bounds. This demonstrates near perfect agreement in the statistical sense that the predicted price is bounded by the observed prices coming from random sampling of different sub-simulations of the Monte Carlo pricer.

In a final test of the HNB performance, the same procedure is applied to a multi-asset basket autocallable using a hybrid local-volatility model. In this example the hybrid model is used to generate Monte Carlo prices for five underlying equities with fixed long-term correlations. This situation requires a 5-fold increase in the number of inputs to the HNB as compared to the single-asset autocallable, including 480 implied-volatilities corresponding to five distinct volatility surfaces. The results shown in Appendix Appendix C demonstrate accuracy similar to that shown for the single-asset autocallable. Overall this demonstrates that the HNB is capable of producing good agreement with Monte Carlo pricers, even in cases where pricers require a large number of inputs and depend on complicated multi-asset dynamics.

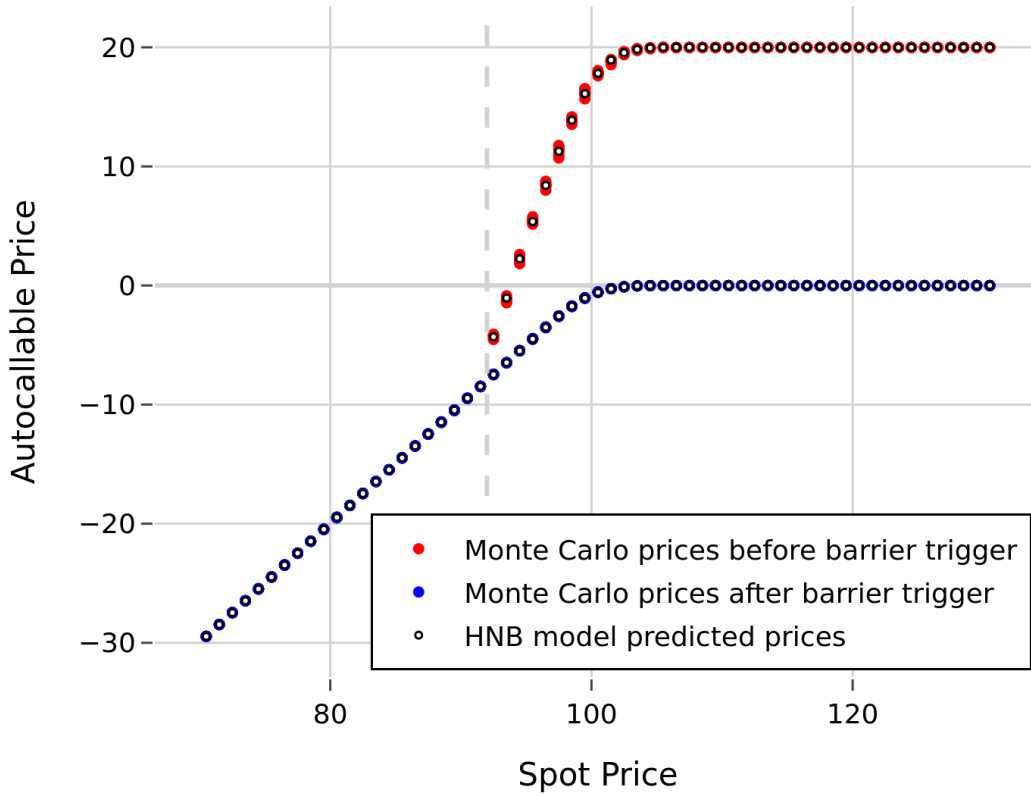
This case involves 96 input volatilities, and in Fig. C.9b we see evidence that the model scales to higher-dimensional inputs, in particular the 5-underlying equity autocallable worst-of basket with 480 input volatilities and 5 spot prices. In both cases the time-series shown is independent of the training data and demonstrates good generalization of the model to out-of-sample data, indicating the model is not overfitting the training data and accurately approximating the standard Monte Carlo pricers.

#### 4. Real-Time Application: Inference Speed and Fine-Tuning Adaptability

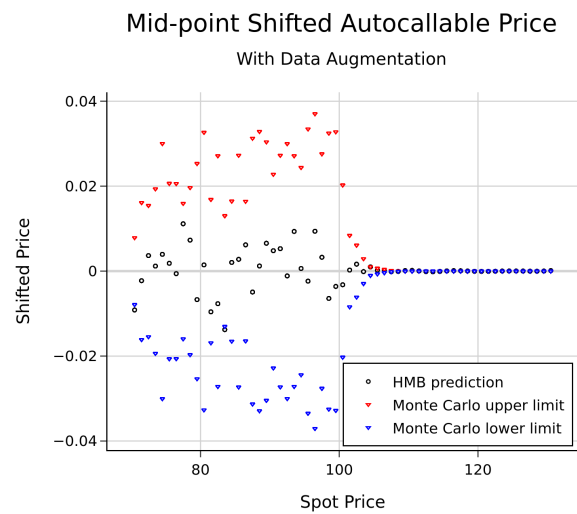
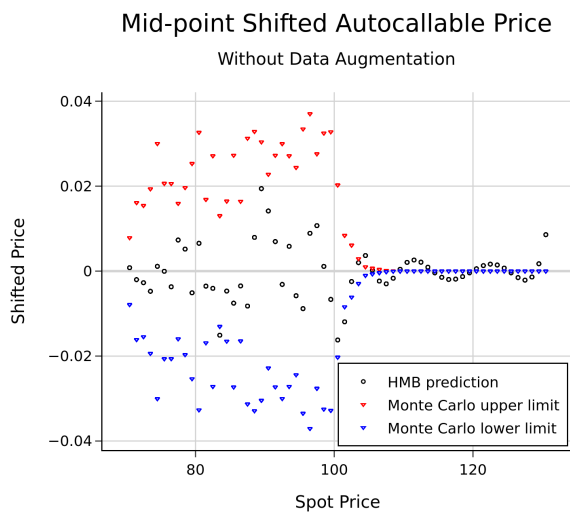
While training-time performance is important, real-time inference and adaptability to market shifts are critical for practical deployment. This section evaluates HNB performance under such constraints. The trained Hyper-Network Block (HNB) is useful in real-time pricing, because the inference speed does not depend on the complexity of the original Monte Carlo pricer. For typical configurations, HNB evaluation is 1,000–10,000× faster than full-path Monte Carlo simulation depending on product complexity and modeling assumptions, with latency on GPU hardware consistently under 2ms. This is true even when running on inexpensive GPUs and for arbitrarily complicated derivatives. This is extremely efficient considering the underlying pricer may require tens or hundreds of thousands of simulation paths under complex stochastic modeling assumptions.

# Autocallable Price (Black-Scholes)

## Multi-Branch for Barrier Trigger



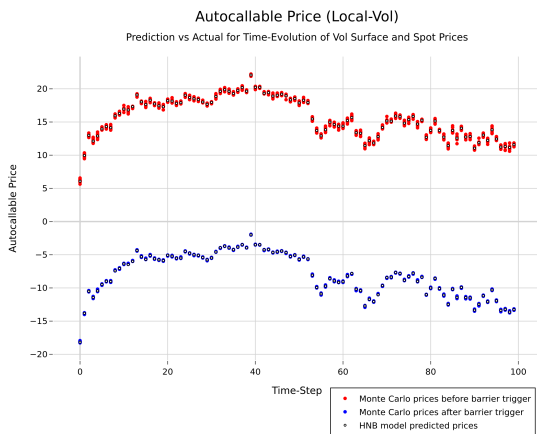
(a) Predicted and simulated prices of an autocallable derivative under the Black-Scholes model with 20% implied volatility. Two branches correspond to pre- and post-barrier trigger states.



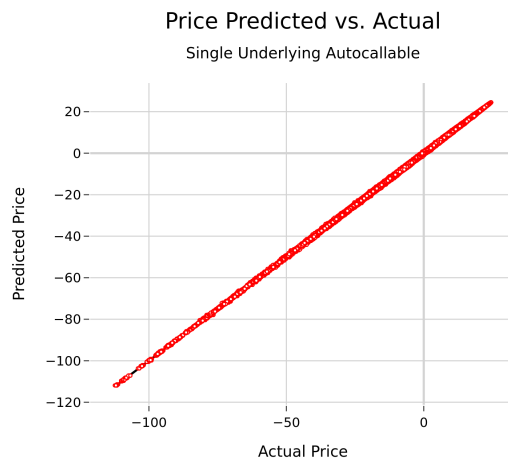
(b) Deviation of HNB predictions from Monte Carlo bounds after the barrier trigger. Midpoint shifting reveals small oscillations relative to simulation variance.

(c) Impact of targeted data augmentation on prediction accuracy near critical spot price regions. Additional samples reduce local prediction oscillations and improve global fit

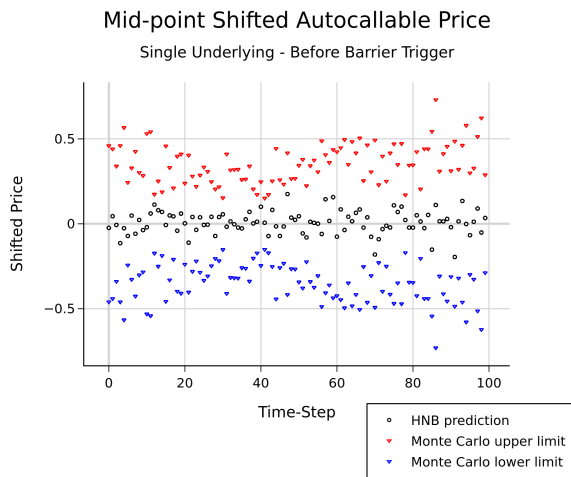
Figure 2: Visualization of autocallable price behavior under Black-Scholes simulation with data augmentation scenarios.



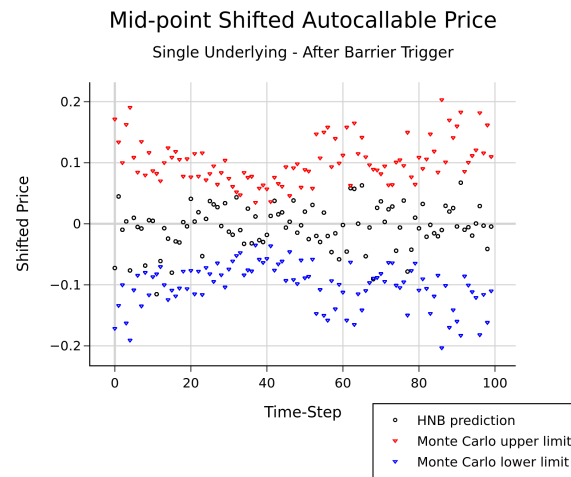
(a) Autocallable derivative pricing across a simulated market time series under a local volatility model. The HNB tracks Monte Carlo prices through changes in spot, volatility, and interest rates.



(b) Scatter plot comparing HNB predictions to Monte Carlo prices for 200 independent simulated time series. Strong correlation confirms accurate generalization to unseen data.



(c) Midpoint-shifted pricing deviations before the barrier trigger. HNB predictions remain within Monte Carlo simulation variance bounds.



(d) Same as (c), but after the barrier trigger event. The HNB maintains accuracy across lifecycle transitions.

Figure 3: Autocallable pricing under local-volatility models across multiple scenarios, including pre- and post-barrier event behavior and model-vs-simulation comparisons.

The main limitation in real-time applications is rather the ability to capture the accurate price evolution under all relevant market conditions. It is impossible to foresee all market conditions that will be encountered, though to mitigate this we can produce large training datasets in an offline fashion that cover many realistic market scenarios. This forms the base training set (BTS), designed to encompass a broad range of possible input conditions and capture heteroskedasticity via multiple smaller simulations per training point.

As noted previously, the mean square error (MSE) loss function, defined in Equation (2), is used as the training objective of the HNB. The training process is based on numerical calculation of the gradient of the loss function with respect to the HNB weights, which involves projection through each point in the base training set (BTS), as shown in Equation (3). Importantly, the gradient computation incurs a time cost that scales linearly with the size of the BTS. This implies that by averaging a set of 10 sub-simulations prior to training, the HNB training can be achieved 10x faster.

However, it is also expected that the use of the average as the training target objective, the model does not obtain information relative to the heteroskedasticity in the BTS. This should be evident in the number of HNB predictions that exceed the range of values observed over the sub-simulations, both in the positive and negative directions, since the relative distance of the HNB prediction from the training target affects the loss equally for all coordinates in the input space. This leads to an increase in what are considered outliers, presenting as model-predictions that lie outside the range of natural variance in the training data produced by the Monte Carlo simulations. Fig. 4a shows that using averaged targets results in more prediction outliers compared to multi-target training. Training on unaveraged sub-simulations captures conditional variance (heteroskedasticity) more effectively, and avoids misleading the model into overfitting to a single noisy estimate while helping to maintain robustness in regions with high model variance.

$$\mathcal{L} = \sum_{i=1}^N (y_i - \hat{y}_i)^2, \quad (2)$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^N \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_j}. \quad (3)$$

Another point of note is that the BTS likely contains a significant amount of data that is not directly applicable to the price evolution observed on short time-scales on the orders of days or even weeks. In practice only a subset of the BTS is directly applicable to the short-term evolution of prices. Training the HNB on the BTS results in globally reduced MSE loss, but the fact that a large number of irrelevant points may be considered can reduce the accuracy in specific regions of the BTS that are relevant in the near

future. To alleviate this, we have developed an approach to bridge the criteria of global optimization of the BTS with the local-region optimization for near-term evolution of the market dynamics. This approach uses a fine-tuning of a base HNB, which is first calibrated to the BTS. The fine-tuning stage subsequently takes the base model and performs rapid additional training using only a subset of the BTS that is selected for its proximity to both current and expected relevant market conditions.

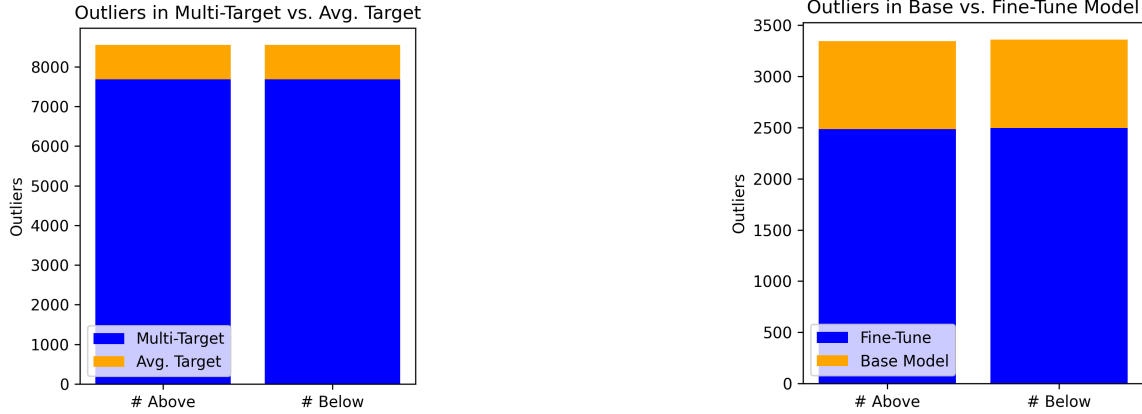
Furthermore, while the base model is trained offline using the full training data, the BTS subset that is selected for fine-tuning is instead averaged over the sub-simulations. This is done so that rapid fine-tuning is achieved. Fig. 4b demonstrates that fine-tuning the HNB using relevant market data subsets significantly improves local prediction accuracy and reduces outlier rates. Using the rapid fine-tuning procedure described here, it is possible to refine the model continuously over the course of a single trading session, often taking only a few minutes to fine-tune the model as market conditions drift, and switching to the newly fine-tuned version of the base model at regular intervals.

In practice, this fine-tuning procedure can be automated to run hourly or during low-liquidity windows, enabling continuously refreshed models in active trading environments. Under this strategy the HNB takes the form of a continuously on-line, fine-tuned version of a base HNB that is pre-trained off-line to the BTS. This procedure balances the need for global accuracy against the need for increased accuracy specific to current market conditions, and does so in a way that is extremely efficient and suitable to real-time applications.

## 5. Conclusions

In this paper, we introduce a fast and scalable approach for full lifecycle derivative pricing using hyper networks, delivering high speed price approximations without compromising accuracy. To our knowledge, this is the first architecture to combine supervised volatility surface compression, lifecycle conditioning, and real-time online fine-tuning within a unified Hyper-Network Block framework. This makes the method particularly well suited for real time financial applications and for the scenarios used in XVA, where the computational complexity of standard pricers becomes a significant barrier to implementation.

A key advantage of our HNB architecture is its ability to accurately capture complex nonlinear price behaviors despite its small size and parsimonious architecture. The HNB models external inputs such as lifecycle events including autocall and barrier triggers through dynamic weight generation, leading to excellent accuracy and precision in derivative pricing problems. The HNB also scales to high dimensional inputs, such as multi asset scenarios involving multiple underlying assets and volatility surfaces through unsupervised training of the latent space. To address the challenge of volatility surface dimensionality reduction, we



(a) Impact of multi target versus average target training strategies on prediction outliers. Multi target training better captures heteroskedasticity and reduces extreme deviations.

(b) Comparison of prediction outlier rates between the base model and fine tuned model. Fine tuning using relevant data subsets improves local accuracy and reduces error rates.

Figure 4: Analysis of model outliers compared to Monte Carlo variance bounds under different training regimes.

utilize an encoder latent space that compresses the volatility surface into a lower dimensional representation while preserving critical pricing information, thereby reducing model complexity without sacrificing accuracy.

The HNB is a notably compact ANN tailored to a single term sheet, yet remains highly efficient for both training and inference. This contrasts with larger models featuring deeper architectures and more parameters, which can support broader use cases beyond single instrument pricing, but at substantially higher training and operational costs. By comparison, the HNB promotes a strategy based on many small and lightweight models that are quick to train, fast to run, and easy to store for deployment.

This work demonstrates key results on how hyper networks combine speed, scalability, and accuracy, offering a practical and powerful tool for pricing both simple and highly complex derivatives. The combination of offline-trained base models with low-cost online fine-tuning offers a compelling solution for maintaining local pricing accuracy under drifting market regimes. These results illustrate the capabilities of our approach and highlight its potential for real-time financial market applications. A key limitation is reliance on representative Monte Carlo training data—extreme regimes not well-covered in the BTS could degrade accuracy, motivating adaptive training set expansion.

From the present work, we identify at least two promising directions for future research. First, while model calibration traditionally relies on vanilla instruments, there is a growing interest in calibrating directly to exotic derivatives. Incorporating fast ANN based pricers into traditional model calibration could yield substantial efficiency gains. Second, although this work focuses on pricing, extending the HNB framework to also compute sensitivities offers further utility for risk management applications. Given that all components of the HNB are differentiable, Greek sensitivities can be obtained via automatic differentiation.

Future work may explore multi-output architectures trained to co-optimize pricing accuracy and delta/vega precision.

## References

- [1] A. Savine and J. Andreasen, *Modern Computational Finance: Scripting for Derivatives and xVA*. Wiley, 2021. ISBN: 9781119540786. Available at: <https://www.wiley.com/en-us/9781119540786>.
- [2] S. Crépey, T. R. Bielecki, and D. Brigo, *Counterparty Risk and Funding: A Tale of Two Puzzles*, 1st ed. Chapman and Hall/CRC, 2014. Available at: <https://doi.org/10.1201/9781315373621>.
- [3] P. Glasserman, *Monte Carlo Methods in Financial Engineering*. Springer, 2004. Available at: <https://link.springer.com/book/10.1007/978-0-387-21617-9>.
- [4] J. Ruf and W. Wang, “Neural networks for option pricing and hedging: A literature review,” *Journal of Computational Finance*, vol. 24, no. 1, pp. 1–46, 2020. Available at: <https://doi.org/10.21314/JCF.2020.390>.
- [5] H. Buehler, L. Gonon, J. Teichmann, and B. Wood, “Deep Hedging,” *Quantitative Finance*, vol. 21, no. 1, pp. 1–20, 2021. Available at: <https://doi.org/10.1080/14697688.2019.1571683>.
- [6] V. K. Chauhan, J. Zhou, P. Lu, S. Molaei, and D. A. Clifton, “A brief review of hypernetworks in deep learning,” *Artificial Intelligence Review*, vol. 57, no. 9, article 250, 2024. Available at: <https://doi.org/10.1007/s10462-024-10862-8>.
- [7] Y. Yang and T. M. Hospedales, “On calibration of mathematical finance models by hypernetworks,” in *Machine Learning and Knowledge Discovery in Databases: Applied Data Science and Demo Track (ECML PKDD*

- 2023), G. De Francisci Morales *et al.*, Eds., Cham: Springer, 2023, pp. 227–242. Available at: [https://doi.org/10.1007/978-3-031-43427-3\\_14](https://doi.org/10.1007/978-3-031-43427-3_14).
- [8] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proc. 14th Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, vol. 15, pp. 315–323, 2011.
- [9] P. Carr and L. Wu, “Analyzing volatility risk and risk premium in option contracts: A new theory,” *Journal of Financial Economics*, vol. 120, no. 1, pp. 1–20, 2016. Available at: <https://doi.org/10.1016/j.jfineco.2016.01.004>.
- [10] R. Rosipal and N. Krämer, “Overview and recent advances in partial least squares,” in *Subspace, Latent Structure and Feature Selection*, C. Saunders, M. Grobelnik, S. Gunn, and J. Shawe-Taylor, Eds., Springer, Berlin, Heidelberg, 2006, pp. 34–51. Available at: [https://doi.org/10.1007/11752790\\_2](https://doi.org/10.1007/11752790_2).
- [11] V. E. Vinzi, W. W. Chin, J. Henseler, and H. Wang, Eds., *Handbook of Partial Least Squares: Concepts, Methods and Applications*. Berlin: Springer, 2010. Available at: <https://link.springer.com/book/10.1007/978-3-540-32827-8>.
- [12] I. D. Mienye and T. G. Swart, “Deep autoencoder neural networks: A comprehensive review and new perspectives,” *Archives of Computational Methods in Engineering*, 2025. Available at: <https://doi.org/10.1007/s11831-025-10260-5>.
- [13] Y. Zhang and Q. Yang, “A survey on multi-task learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, 2022. Available at: <https://doi.org/10.1109/TKDE.2021.3070203>.

## Appendix

### Appendix A. Hyper-Networks for Flexible Regression

Standard ANNs process inputs through linear weights and bias term, then apply non-linear activation functions. In our research we attempt to improve the learning behavior of such networks for application to regression for derivative pricers. This idea is the central motivation to consider hyper-network architecture.

$$f(\vec{x}) = g(\vec{x} \cdot W + \vec{b}) \cdot V + d \quad (\text{A.1})$$

$$f(\vec{x}) = g(\vec{x} \cdot W^H(\vec{x}) + \vec{b}^H(\vec{x})) \cdot V + d \quad (\text{A.2})$$

Equation (A.1) represents the output of a single layer of a standard ANN, while Equation (A.2) shows the equivalent formulation for a hyper network. Comparing these demonstrates the key difference between standard feed-forward

networks and hyper-networks, which is the dependence of the weights and bias terms on the inputs. The inputs to the hyper-network can be chosen independently from the inputs to the target network, leading to a number of interesting possibilities, including external static and data-dependent dynamic contexts for weight generation in the TN.

In traditional pricers, typically the calculation is divided into two major components, the model and the instrument. In the case of Monte Carlo pricers, typically the model is a simulation based on a stochastic process for the financial state variables, which requires calibration to the market inputs such as vanilla options and interest rates. The instrument is instead a numerical representation of the term sheet, containing rules for calculation of the derivative payoff. In the process of pricing under Monte Carlo simulations, generally the state variable paths are generated by the model, and as a second step the instrument is priced based on the generated paths. Thus it is possible in the HN architecture to consider the hypernetwork as a representation of the model, and the target network corresponding to the instrument.

Even if viewed as the counterpart to traditional stochastic models, the HNB is not independent of the instrument details. Given the intended application of the HNB is to represent a single term-sheet, the training data is specific to Monte Carlo simulations for the same term-sheet, and the trained HNB therefore directs the weights of the TN to produce the approximate price for the term-sheet using given market data variables. This intertwined relationship is not unprecedented even in traditional stochastic models, in particular Monte Carlo simulations can include state-dependent interpolation, special time-point selections, and other specializations to the instrument, that while not necessarily found in the mathematical specification of a stochastic process, nonetheless often appear in practical Monte Carlo implementations.

Another interpretation of the role of the hyper-network is to take in auxiliary variables that influence the nature of the pricer and perform this change in the target network. This is used in the current work to apply barrier triggers as product lifecycle events. The influence of the auxiliary variables on the target network behavior allows the model to simultaneously learn different tasks, and this is commonly referred to as multi-task learning [13].

Considering the role of the HN in controlling the weights of the TN, it is also interesting to see the additional dependence of the TN weights themselves on the inputs, as shown in Equations (A.3) and (A.4) for the weight and bias terms. This necessarily implies in (A.2) that there is increased non-linearity formed by the product of non-linear terms when compared to the standard ANN in (A.1). Furthermore, the model itself learns to include the additional non-linearity through the functional dependence of the HN outputs found in the model training. Fig. A.5 highlights a consistent observation throughout our investigations - the hyper-network shows significantly increased learning

rates, likely due to these additional capacities to model non-linearities.

$$W_{ij}^H(\vec{x}) = \left[ g \left( \vec{x} \cdot W + \vec{b} \right) \right]_{(n \cdot i + j)} \quad (\text{A.3})$$

$$b_i^H(\vec{x}) = \left[ g \left( \vec{x} \cdot W + \vec{b} \right) \right]_{(n \cdot n + i)} \quad (\text{A.4})$$

Lastly it is important to note that the number of weights in the target network grows quadratically with the size of the target network, which corresponds to growth of the hyper-network that supplies these parameters. This is one of the main drawbacks of the hyper-network architecture, since larger networks with more parameters incur longer training times and can possibly lead to overfitting. In order to mitigate this, our HNB architecture is intended to keep the target network as small as possible while retaining the beneficial features of HNs for learning non-linear behavior and multi-task learning. Despite the small target network utilized, we can stack a number of HNB for deep learning with only a linear scaling of the total parameter count to achieve shorter training times while maintaining model accuracy.

## Appendix B. Volatility Surface Latent Space

The volatility encoder in our Hyper-Network Block (HNB) architecture produces a 2D latent space provides the best trade-off between compactness and predictive performance for the datasets considered; a broader analysis of dimensionality is left for future work. Fig. B.6 shows heatmaps of the single-underlying autocallable price under the local-volatility model as a function of the 2D latent space produced by the encoder. Specifically, it illustrates how the HNB produces different prices as a function of the latent space, depending on other inputs like spot price and barrier trigger states. This highlights the flexibility offered by the HNB when used with an encoder optimized for compressing high-dimensional inputs into a compact latent space.

The choice of a 2D latent space for encoding up to five volatility surfaces may initially seem counterintuitive, especially in the context of standard Monte Carlo pricers, which often use dimensionality reduction techniques such as early-exercise regression. In contrast, the HNB predicts the average terminal price rather than simulating path-wise dependencies. Thus, the latent space is viewed as an embedding of the volatility surfaces, rather than a set of explanatory variables. In a sense, the model captures coordinated movements of equity option volatilities, leading to a lower-dimensional embedding that is more appropriate for the prediction task. Fig. B.7a shows in our datasets that using a 2D latent space results in lower prediction errors compared to a 3D latent space, likely due to its more compact and generalizable representation.

These results suggest that the 2D latent space effectively captures the main modes of variation in the volatility

surfaces relevant to the prediction task, while avoiding overfitting that may arise from a higher-dimensional representation. The added degrees of freedom in a 3D latent space may introduce unnecessary flexibility, causing the model to fit idiosyncrasies in the training data that do not generalize well to unseen market conditions. Therefore, the dimensionality of the latent space should be understood not in terms of the number of raw inputs, but rather in terms of the complexity of the target prediction function and the smooth mapping from volatility surface structure to expected terminal payout.

This observation is supported by the results shown in Fig. B.7b, where an autoencoder is trained to the input volatility surface directly, to reproduce the volatilities from the latent space, rather than the prices themselves. In this case the results are opposite, in the sense that a higher dimension of the latent space performs better due to the inherently higher-dimensional nature of the input space compared to the combined effect of the volatility inputs on the pricer output space.

To further explore the structure of the volatility surface latent space, we analyze how parallel shifts applied to each of the five volatility surfaces manifest as trajectories in the 2D latent embedding. Fig. B.8 shows these trajectories, color-coded from red (negative shift) to green (positive shift). Each path reflects the encoder’s response to a systematic perturbation of a single surface. While some regions of the trajectories are relatively smooth, others exhibit loops or sharp directional changes. These non-smooth features suggest regions where the encoder transitions between different regimes of latent representation, potentially indicating nonlinear boundaries in the manifold where pricing sensitivity to certain surface features becomes more pronounced. Such behavior illustrates that the latent space is not merely a dimensionality reduction, but a learned representation that actively reorganizes the input geometry to reflect pricing relevance. These complex trajectory shapes offer insight into how the encoder prioritizes and interprets variation across the volatility surfaces.

## Appendix C. Multi-Asset Hybrid Local-Volatility Autocallable

Fig. C.9 presents detailed results for a five-underlying basket autocallable derivative under a hybrid local volatility model, demonstrating the scalability of the Hyper-Network Block (HNB). Similar to Fig. 3, the results for a single simulated time-series of market data are shown in Fig. C.9a. Midpoint-shifted prices for this series are shown in Fig. C.9c - C.9d, and a collection of 100 test data time-series are summarized in Fig. C.9b, which compares HNB predictions with Monte Carlo time-series. The HNB in this case uses more than 500 market data inputs, including five implied-volatility surfaces, and demonstrates accuracy that is equal to the simpler case of the single-underlying autocallable shown in Fig. 3. The fact that the HNB is seen to perform with high quality as the number of inputs scales

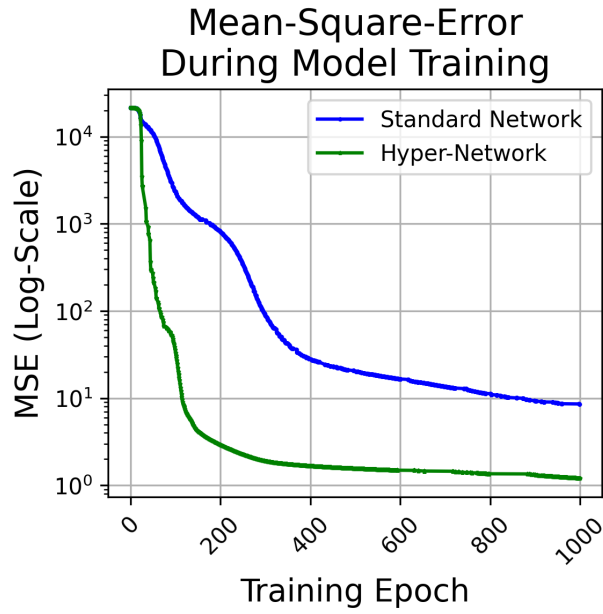


Figure A.5: Comparison of convergence speeds between standard artificial neural networks and hyper network based models. Hyper networks achieve faster training due to dynamic weight generation.

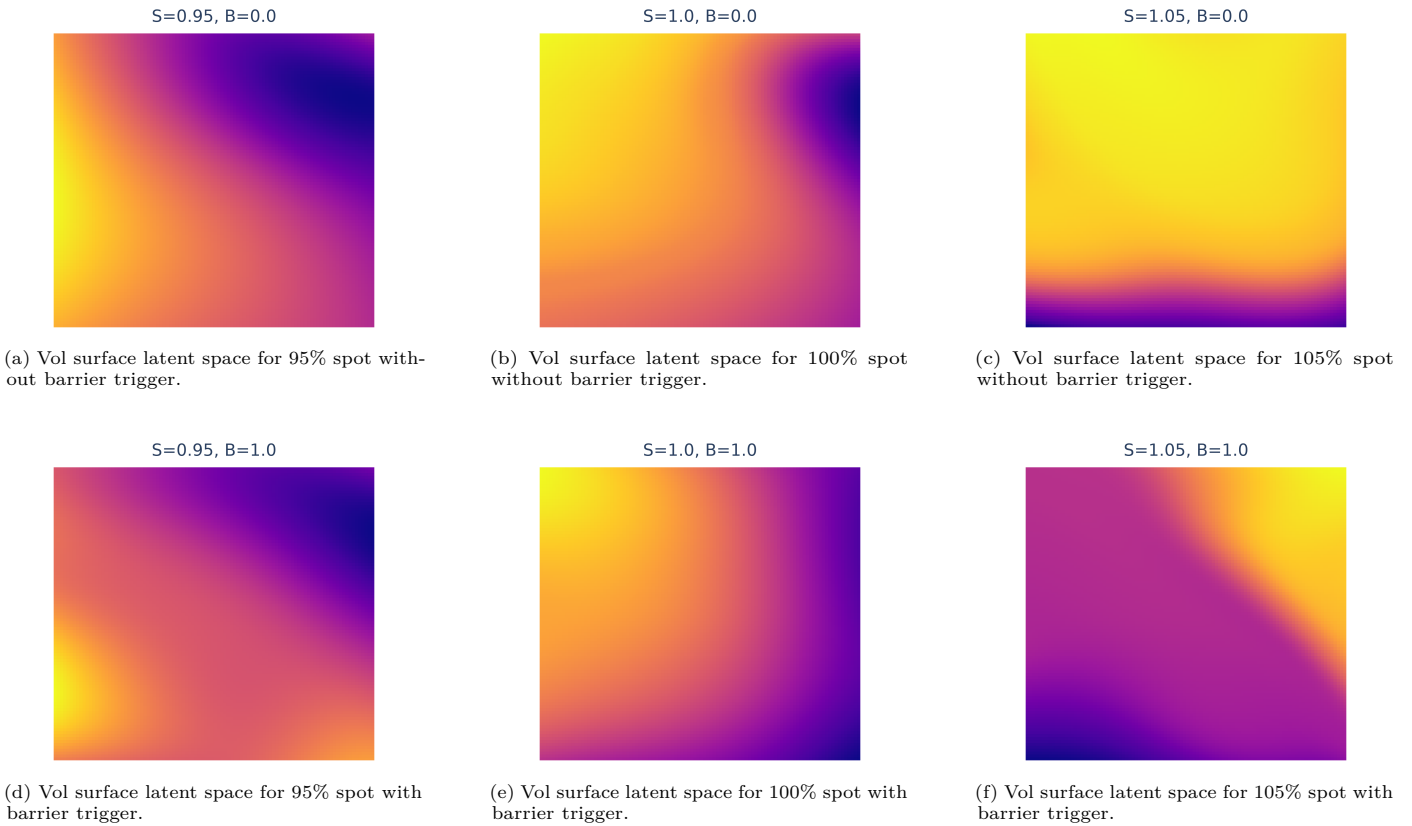
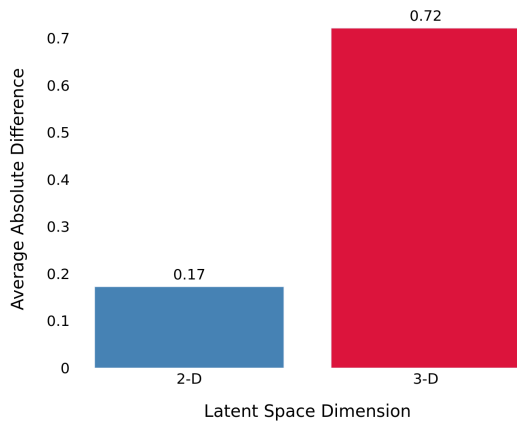
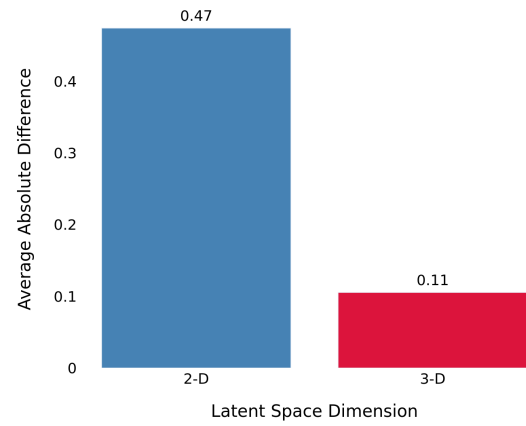


Figure B.6: Heatmaps showing HNB pricing predictions across the volatility surface latent space, under varying spot price levels (95%, 100%, 105% of initial) and barrier trigger states (before and after triggering). The vol surface encoder latent space adapts flexibly to changes in both spot prices and lifecycle events.

Difference Between HNB Predictions and Test Data Prices



Difference Between AE Reconstruction and Volatility Input



(a) Comparison of average absolute prediction errors for two-dimensional and three-dimensional latent spaces for volatility surface encoding.

(b) Comparison of average absolute prediction errors for two-dimensional and three-dimensional latent spaces for standalone autoencoder.

Figure B.7: Comparison of accuracy results for different latent space configurations.

from tens to hundreds is a good illustration of the scaling performance and applicability of the HNB architecture to high-dimensional derivative pricing.

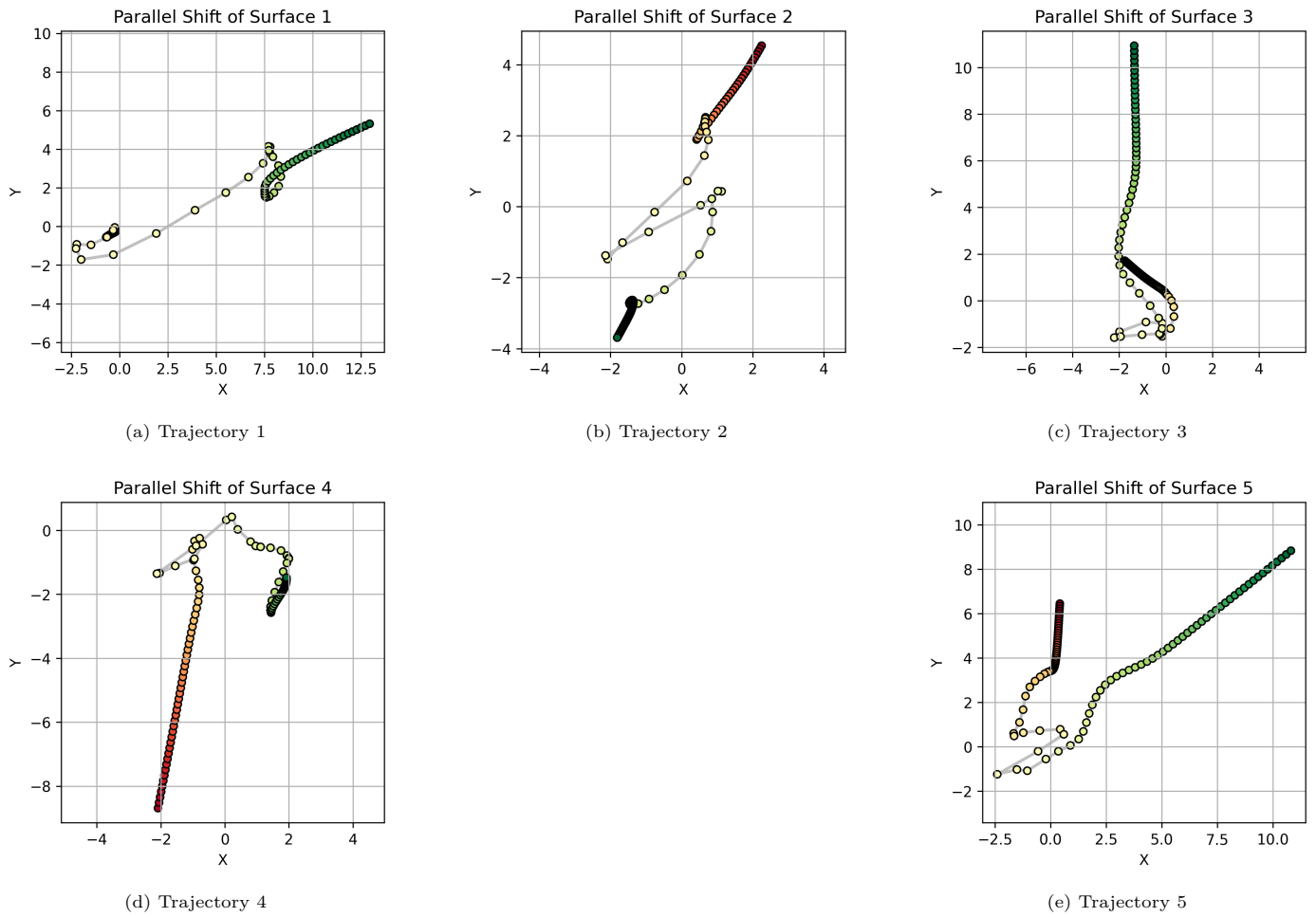
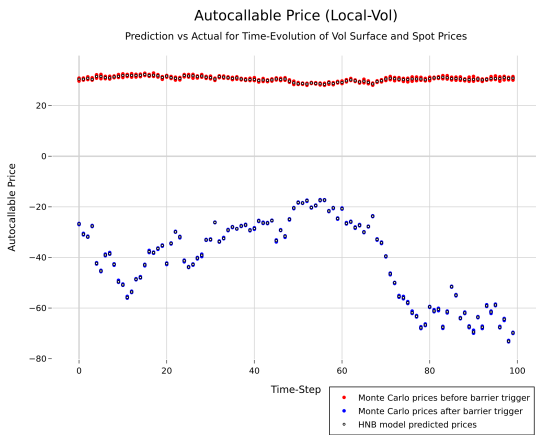
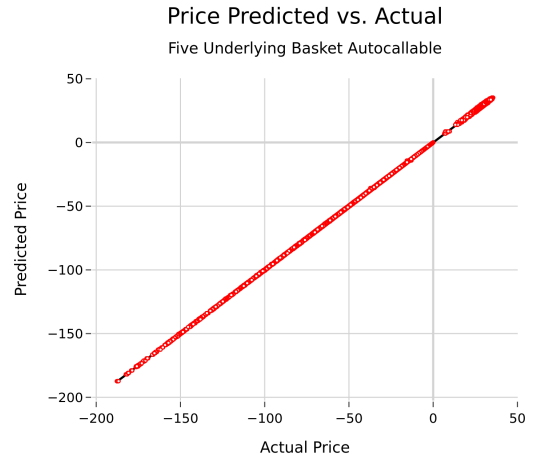


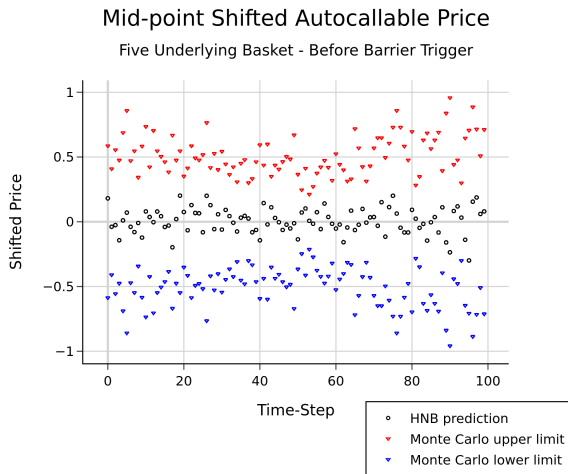
Figure B.8: Visualization of latent space trajectories corresponding to parallel shifts of five distinct volatility surfaces. Each trajectory represents a surface shift, with a color scale from red (negative shift) to green (positive shift) indicating the magnitude of the change. These figures illustrate how the encoded high-dimensional volatility surface is mapped into a 2D latent space, showcasing the nonlinear manifold approximation of the pricer function. This approach encodes complex, high-dimensional data into a lower-dimensional representation, with the resulting trajectories capturing the intricate geometry of the manifold used to efficiently approximate the pricer function.



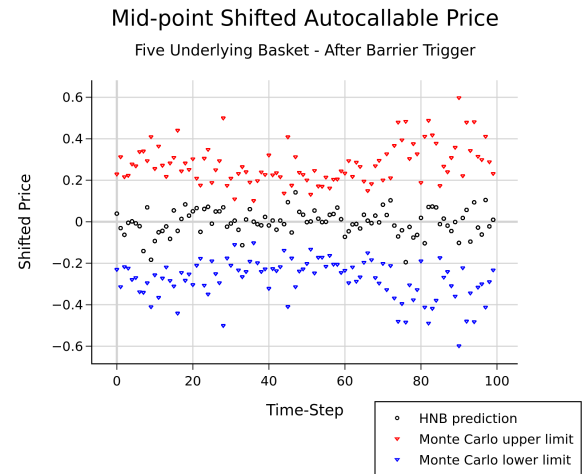
(a) Pricing evolution of a five underlying basket autocallable derivative across a simulated market time series under a hybrid local volatility model.



(b) Scatter plot comparing HNB predictions to Monte Carlo prices across 100 simulated time series for the five underlying basket product. Strong agreement demonstrates model scalability.



(c) Midpoint-shifted pricing deviations before the barrier trigger for the basket autocallable. Predictions remain within Monte Carlo simulation variance.



(d) Same as (c), but after the barrier trigger event. The HNB maintains prediction accuracy through lifecycle transitions.

Figure C.9: Basket autocallable pricing under local-volatility models: price evolution time-series, Monte Carlo comparisons, and behavior across barrier trigger events.

Numerix LLC Corporate Headquarters

100 Park Avenue

15th Floor

New York, NY 10017

Tel: +1.212.302.2220

Fax: +1.212.302.6934

[www.numerix.com](http://www.numerix.com)

