**Quant\_blog draft**

Title: Time-Series analysis using Stock market data with LSTM, VAE and Diffusion based models and comparing them.

**The Challenge of Forecasting Stocks**

Stock price forecasting is very much similar to weather prediction. They depend on a large number of external variables introducing noise, uncertainty and complexity in their prediction process.

Classical time-series models like ARIMA, SARIMA, GARCH, etc, have been the backbone of financial forecasting for decades. They are powerful, interpretable, and statistically rigorous. But their **stationarity assumptions, Linearity** and **deterministic** modelling, they often fall short when applied to stock market data.

With lots of applications of transformer models coming to surface, **LSTM** models have also been used for capturing nonlinear patterns and long-term dependencies. However, they may end up **memorizing random fluctuations** (noise). Also, standard LSTM architectures output a **single forecast** for the future.

For modelling highly unpredictable stock data, it is important to deal with noise and uncertainty differently by shifting from point forecast to probabilistic forecast.

**GenAi models** have made it possible. Generative models like VAEs and Diffusion models can be modified to tackle this challenge. They can be modelled to remove noise and give probabilistic forecasts.

In this blog we have discussed the variants of VAE and Diffusion model for stock predictions and used suitable metrics to compare between LSTMs and GenAi models. We adopt simpler versions of the variants discussed in the recent research papers. We focused on explaining the core concepts in clear and accessible language rather than reproducing the technical complexity so that readers from diverse backgrounds can follow along.

**The Great Berry Hunt**

In Sunny Meadow, three friends wanted to find where the juiciest berry would grow next.

First was Leo the Lizard, who represented the LSTM. Leo had a super memory. He carefully walked along a branch, remembering every single twist and turn it took. "Aha!" he said. "After following this exact path, I predict the berry will grow right at this tip!" He gave one, single answer based on the path he followed.

Next was Vicky the Bear, who was like a VAE. Vicky didn't care about the path. She just wanted to know how healthy the whole bush was. She checked the soil and sniffed the leaves. "The bush feels very strong today!" she declared. "So, a berry will probably grow somewhere in this big, sunny area." She gave a general zone, not just one spot.

Last came the Firefly Squad, who acted like a Diffusion model. They started as a messy, blinking cloud of random light. But they had a special trick. They could turn their messy cloud into a clear picture. By working together, their random blinking suddenly formed ten different, glowing images of berries, showing all the places a new one might pop up!

**Forecasting with LSTM**

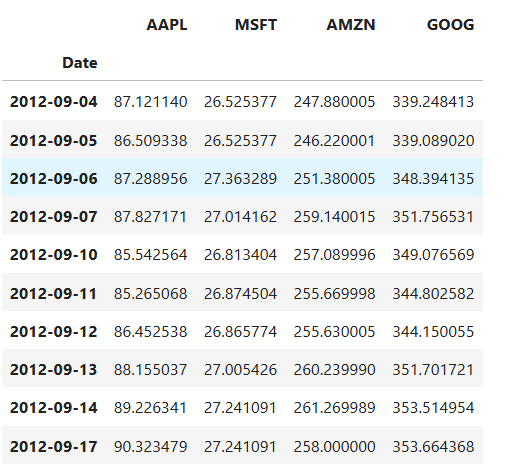
Unlike statistical approaches, LSTMs can learn complex, non-linear relations directly from data. They are designed to handle sequential information, making them well-suited for financial time series. In this module we will not describe the working of LSTM. As the topic is different, we will go through this section learning how to use LTSM for time-series data directly.

**Code work**

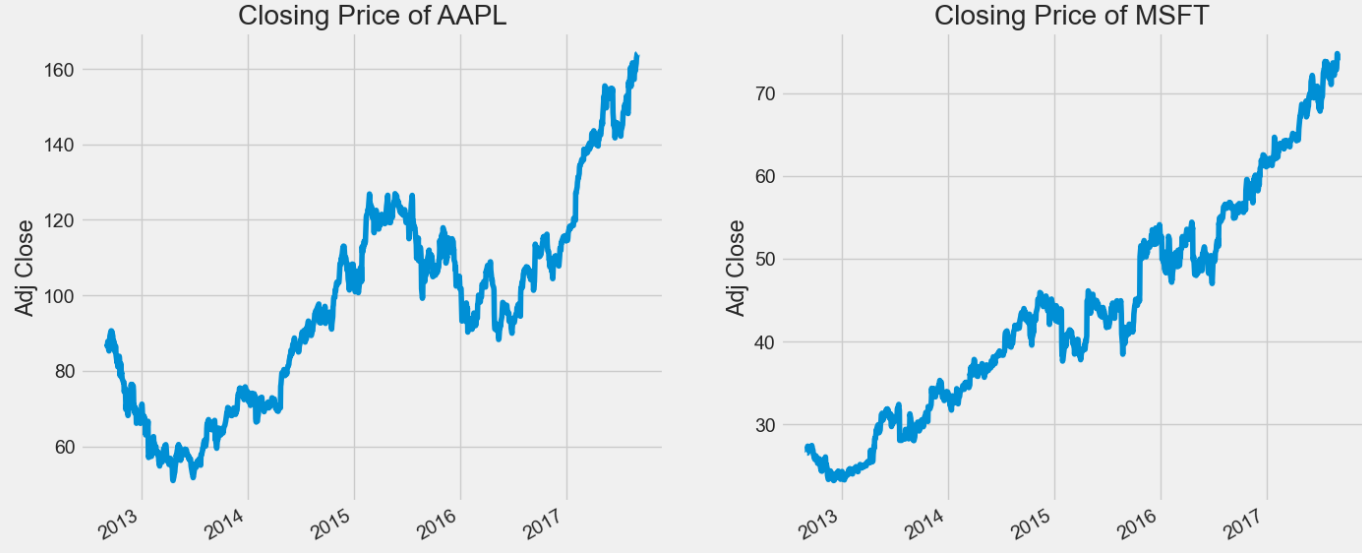
The first step is getting the data. We have used StockNet dataset for companies Apple (ticker: AAPL), Microsoft (ticker: MSFT), Amazon (ticker: AMZN), Google (ticker: GOOG).



The below is the output. We have used ‘Adj Close’ data from the csv files. For prediction, the adjusted close price is used instead of raw close price because it provides a more accurate and consistent representation of a stock’s historical value.

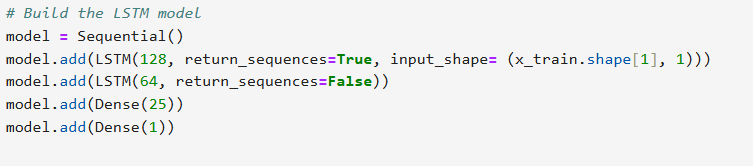


To get the idea of behaviour of stock closing prices we have plotted their closed prices. As we are not using classical models, the stationarity, dependencies and patterns do not matter while working with LSTMs.





Now we will write the LSTM model architecture.



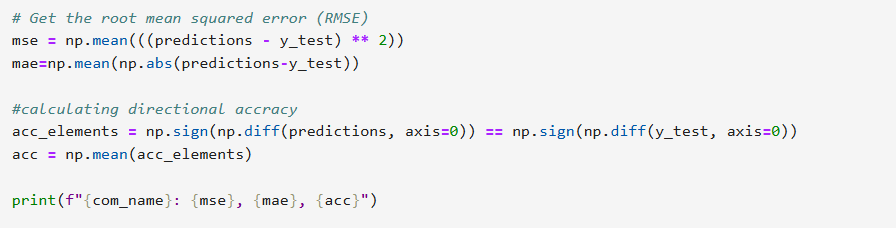
In order to work on four companies directly, we have applied iteration method to evaluate the performance of our model on each company.

We have used MSE (Mean-Squared Error), MAE (Mean Absolute Error) and Directional Accuracy as our performance metrics.

Quick Recap:

**Directional Accuracy** measures the percentage of correctly predicted upward or downward movements of a stock’s price over a specific period.

The below is the working code for calculating above performance metrics:



Results

| Metrics | AAPL | MSFT | AMZN | GOOG |
| --- | --- | --- | --- | --- |
| MSE | 0.00102 | 0.00273 | 0.00076 | 0.00264 |
| MAE | 0.02612 | 0.04742 | 0.02249 | 0.04491 |
| Directional Accuracy | 0.54098 | 0.4918 | 0.54098 | 0.5246 |

The above MSE and MAE are scaled values i.e. calculated on the scaled data. Hence their value will always be in the interval [0,1]. The directional accuracy is calculated by counting the number of times the model correctly predicted the market movement (positive sign for up and negative for down movement) and then taking the average.

Why poor directional accuracy in spite of low MAE/MSE?

MSE/MAE are low because LSTM is minimizing the distance between values by virtue of the loss function on which the model is trained. Scaling further reduces the magnitude. On the other hand, directional accuracy is poor because the model misses whether the next move is up or down.

The directional accuracy can be improved by the following ways:

1. Training the model to predict returns rather than raw price.
2. Modifying the loss function.

Below we have plotted the predicted returns with actual returns for AAPL for visualisation.



**Generative AI for Time-Series**

**Why Generative models?**

Instead of predicting a single number, they generate multiple plausible futures giving a probabilistic forecast. Though we will not cover this topic in this blog, but they are great for **risk management scenarios** and **stress testing**.

**Forecasting with Diffusion model**

Before we dive into the model building and stock prediction code work, it is important to go through some concepts integral to the core of diffusion model.

**How Diffusion model works:**

The goal of Diffusion model is to learn data distribution in order to predict generate new samples that look like the training data (images, text, time series, etc.). The working can be summarised in the following steps:

Step-1: Forward process (adding noise)

We start with real data and we gradually **add Gaussian noise** step by step until it becomes pure noise. This is like slowly corrupting your data until all structure is lost.

Step-2: Reverse process (denoising)

Now, we want to **learn** the *reverse*: how to go from noise to clean data. We don’t know the true reverse process. So, we **train a neural network** to learn how the noise distorts the data and then use it for prediction.

Step-3: Sampling (the generating step)

To generate new data:

Start with pure Gaussian noise. Iteratively apply the learned reverse step along with a parallel analysis of user prompt. After large number of steps, you get a clean sample x0​ that looks like the training data, which is indeed the final image/text/any sample presented to the user.

Now that we have built an understanding of how diffusion models work, we will see how researchers have combined them with transformers to predict stock data. The research article we are following is quite detailed and mathematical. Instead of diving into every equation, we will keep things simple by focusing on intuition behind their approach.

For our own experiment, we have adopted a simplified version of their model. Instead of using large-scaled datasets, we worked with a smaller set so that core ideas are easier to follow.

**How DiffStock uses Diffusion for Stock Prediction**

DiffStock is the model suggested by the paper. They have **combined diffusion modelling** (for uncertainty / probabilistic forecasting) with a **relational deterministic model (MaTCHS)** that captures inter-stock relations, masked relational attention and is also the **denoising network**. The Diffusion part is the probabilistic backbone in which their relation transformer is embedded.

**What is a Masked Relational Transformer?**

Stocks are not independent. They influence each other. To model this, researchers use graph-based models or relational models where each stock is a node and edges represent the relation.

**MaTCHS** (Masked Transformer and Convolutional network for Hypergraph relation-based Stock time-series generation) is the denoising network and also the relational model. It captures relation between stocks via **attention mechanism** and learns the denoising process to predict the noise (deterministic prediction).

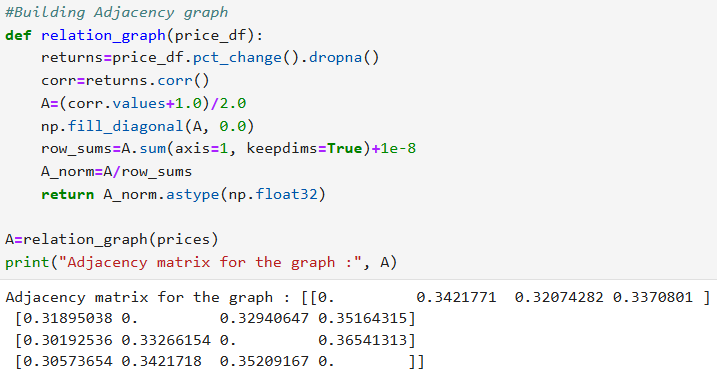
Workflow of our code:





**Code work:**

We have used the same stock data of four companies, AAPL, MSFT, AMZN, GOOG. We have just considered the correlation matrix and the corresponding relation graph rather than hypergraphs. Below is the code for adjacency matrix:



This is followed by the Relational Self-Attention layer and the SimpleMaTCHS architecture.

The **Relational Self-Attention layer** is inspired by how the **self-attention mechanism** in Transformers allows each token to “attend” to other tokens — but here, we’re adapting that idea to **graph-structured data**, where relationships between nodes are encoded in an **adjacency matrix**. The **adjacency matrix** tells us which nodes are connected — and we can use that structure to bias the attention mechanism, making the model focus more on **neighbouring nodes** instead of treating all nodes equally.

Below is the code for SimpleMaTCHS architecture:



Let’s understand the model architecture a bit more in detail:

**Input Projection:**

self.input\_proj = nn.Linear(feature\_dim, model\_dim)

This converts the raw input features for each node into a higher-dimensional **model space**.  
Think of it as embedding your input into a form the network can reason with — just like word embeddings in NLP.

**Diffusion step embedding:**

self.temb = nn.Embedding(T\_DIFFUSION + 1, model\_dim)

In diffusion models, each sample is trained with a “noise level” or **diffusion timestep t**.  
This line learns an **embedding vector** for every possible timestep — encoding *how much noise* was added.  
Later, this gets added to the node features so the model knows which diffusion step it’s denoising.

**Self-Attention Layer:**

self.attn = RelationalSelfAttention(model\_dim, num\_heads, N)

This layer lets every node **attend** to its graph neighbors, weighted by the adjacency matrix.  
It’s similar to the Transformer’s self-attention but *graph-aware*.  
So nodes exchange information — but only along the graph’s valid edges.

**Feedforward Network:**

self.ff = nn.Sequential(

nn.Linear(model\_dim, model\_dim),

nn.ReLU(),

nn.Linear(model\_dim, model\_dim)

)

This defines a **two-layer feedforward neural network (FFN)** — also known as a **position-wise feedforward network** in Transformer architectures.

It processes each feature vector independently (no recurrence or convolution), applying a nonlinear transformation that helps the model **refine its internal representations**.

**Output Head:**

self.out\_head = nn.Linear(model\_dim, 1)

After all the processing, each node has a final vector of size model\_dim.  
This layer projects that vector down to a **single scalar prediction** — e.g., the reconstructed signal at that node.

**Sequence Pooling:**

self.seq\_pool = nn.AdaptiveAvgPool1d(1)

Since the input can be a sequence (multiple timesteps), this layer performs **temporal pooling**, averaging across the sequence dimension to get one summary vector per node.

Explanation of model’s working:

* Each feature vector is linearly projected and combined with the diffusion-step embedding. This conditions the model on the current noise level.
* Each node looks at its graph neighbours to update its representation using **Relational Self-Attention**.  
  This captures spatial dependencies across the graph.
* Then we do Temporal Pooling where we reshape the tensor so that each node’s temporal sequence can be averaged over time (seq\_pool). After pooling, each node has a single feature vector summarizing its entire time-sequence.
* Finally, we produce one scalar output per node — e.g. the denoised signal at that diffusion step.

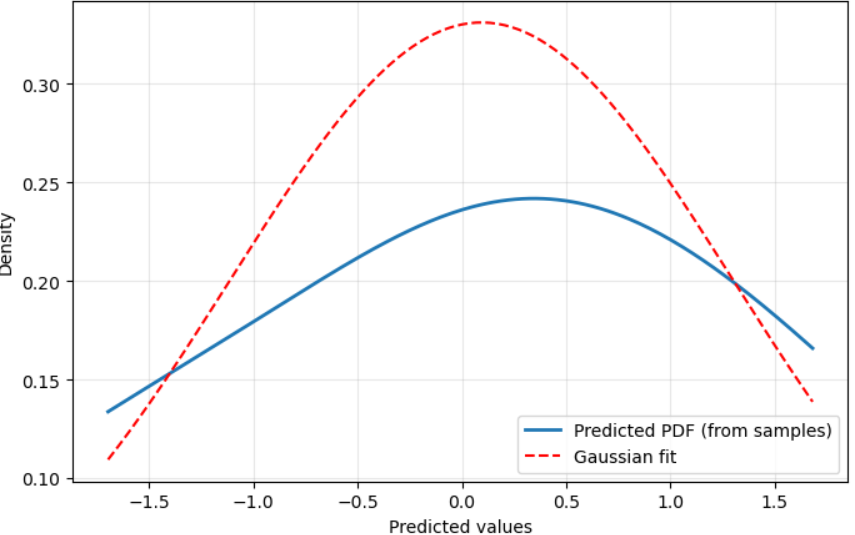
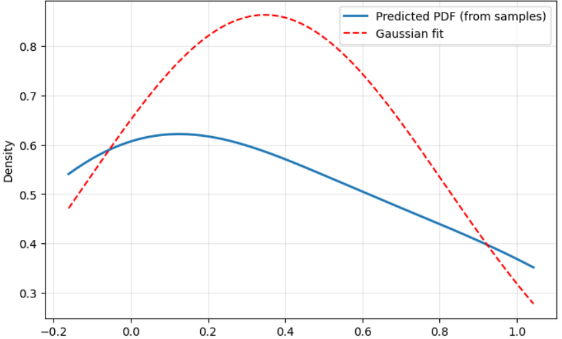
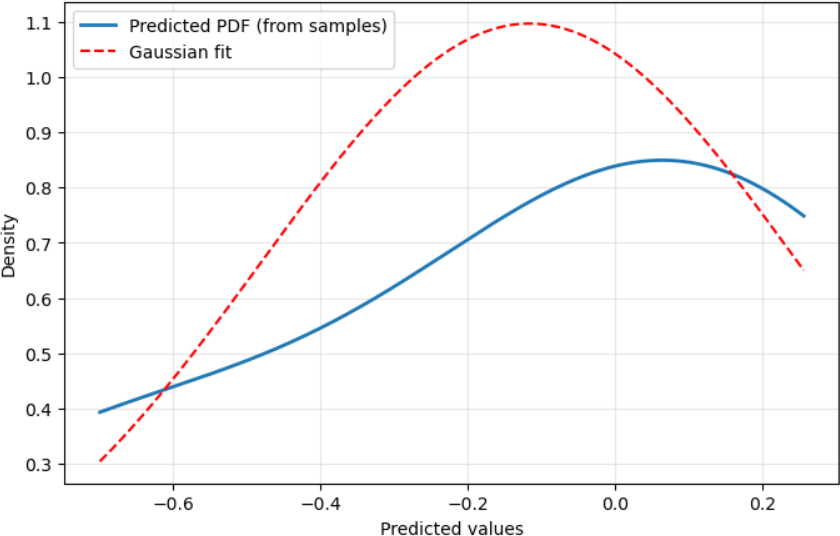
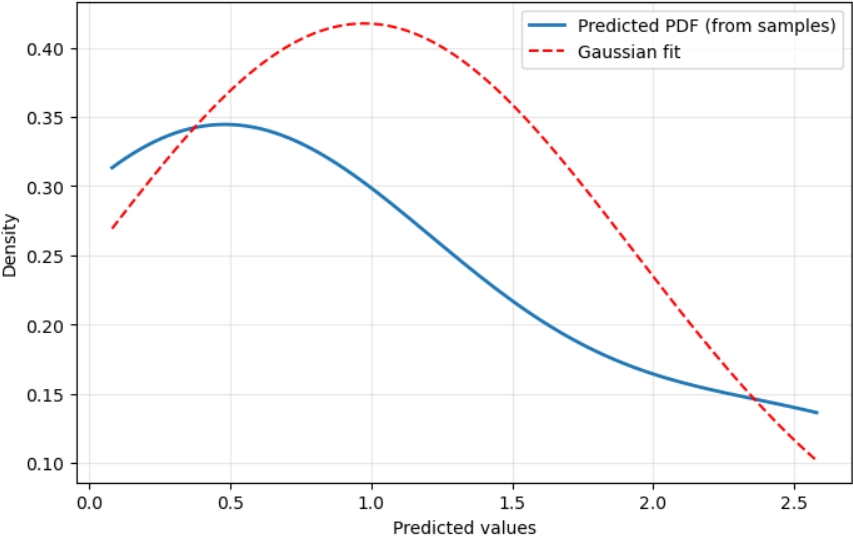
This model is then combined with the diffusion forward and reverse process which is followed by sampling and evaluation. As the output of this DiffStock model is probabilistic, we have taken the **mean of all the possible future outcomes** (summary metrics) for computing deterministic model performance metrics like MSE, MAE and Directional Accuracy.

| Metrics | AAPL | MSFT | AMZN | GOOG |
| --- | --- | --- | --- | --- |
| MSE | 0.92741 | 1.00392 | 0.82797 | 0.9471 |
| MAE | 0.77776 | 0.80436 | 0.71733 | 0.7731 |
| Directional Accuracy | 0.4609375 | 0.45667614 | 0.56392045 | 0.48899148 |

Why is the MSE greater than 1 even when calculated on scaled data? It is because we used standard scaling for this model which doesn’t necessitates that MSE and MAE are less than 1.

There are also some metrics for evaluating the quality of a predicted interval given the true value. Eg **Negative Log-Likelihood (NLL)**, **Prediction Interval Coverage Probability (PICP)** and **Sharpness.** Here we have discussed the Negative Log-Likelihood. You are encouraged to test explore the other metrices also.

NLL tells how likely the true value is under your predicted distribution. Lower is better. For calculating NLL, we have to assume the distribution of true samples (i.e. true price value if they were assumed probabilistic) as gaussian distribution (standard assumption). Though we have assumed Gaussian distribution of our samples (for simplicity), we are providing the KDE plot of some predicted samples of stock **AAPL** to give a better idea of the distribution.

The NLL for the stock AAPL for the given time-steps is about 1.7. You can try finding the NLL for other stocks as well.

**Forecasting with VAE**

### **Forecasting Stocks with a Diffusion Variational Autoencoder (D-Va)**

We will repeat the same procedure and first go through some concepts integral to the core of our model. The analysis is based on the research paper **"Diffusion Variational Autoencoder for Tackling Stochasticity in Multi-Step Regression Stock Price Prediction"** by Koa et al.

For our own experiment, we have adopted the paper's "specialist" strategy, training a separate model for each stock. We worked with a smaller set of four major tech stocks—AAPL, AMZN, MSFT, and GOOG—to make the core ideas easier to follow.

### **How the D-Va Model Uses a VAE for Stock Prediction**

The D-Va model uses a Variational Autoencoder (VAE) as its main prediction engine, trained with the diffusion technique.

**What is a Variational Autoencoder (VAE)?**

A VAE is a neural network that learns to compress data into its most essential, underlying essence.

* + **The Encoder:** This part of the network takes a high-dimensional input (e.g., 16 days of 5 features) and compresses it down into a small, low-dimensional summary called the latent vector (z). This forces the model to learn the most important patterns and ignore noise.
  + **The Decoder:** This part takes the compressed summary z from the encoder and attempts to reconstruct the original data. In our model, it's a **conditional generator**: it takes the essence of the *past* and generates a prediction for the future.

The D-Va model cleverly combines these ideas.

* + The **VAE acts as the main prediction engine**. The encoder learns the "essence" of the past 16 days, and the decoder generates a prediction for the next 16 days based on that essence.
  + The **Diffusion process is used as a training technique**. By training the VAE on noisy versions of the past and future data, we force it to become incredibly robust at ignoring random market fluctuations.
  + A **second, separate VAE** is trained to act as the "denoising network" from the diffusion model, allowing for a final cleanup step to make predictions even more stable.

### **Workflow of Our Code**

Our final, corrected Python notebook follows the "specialist" strategy from the research paper. The main script in the final cell orchestrates the entire workflow:

1. **Main Loop:** It starts a for loop that iterates through each stock file in our DATA\_FILES list ('AAPL.csv', 'AMZN.csv', etc.).
2. **Data Loading:** Inside the loop, it loads and preprocesses the data for **only one stock**. A critical step here is converting prices to **percentage returns**, which creates a more stable ("stationary") target for the model to predict.
3. **Model Creation:** It builds a **brand new, untrained D-Va model** instance.
4. **Training:** It calls dva\_model.fit() to train this specialist model **only on the current stock's data**.
5. **Evaluation:** After training is complete, it uses the trained specialist to make predictions on its own held-out test set.
6. **Visualization:** It generates a separate, clearly labeled plot for the current stock, comparing the predicted returns to the actual returns, and calculates performance metrics.
7. The loop then repeats this entire process for the next stock.

### **Code Work: The D-Va Model Architecture**

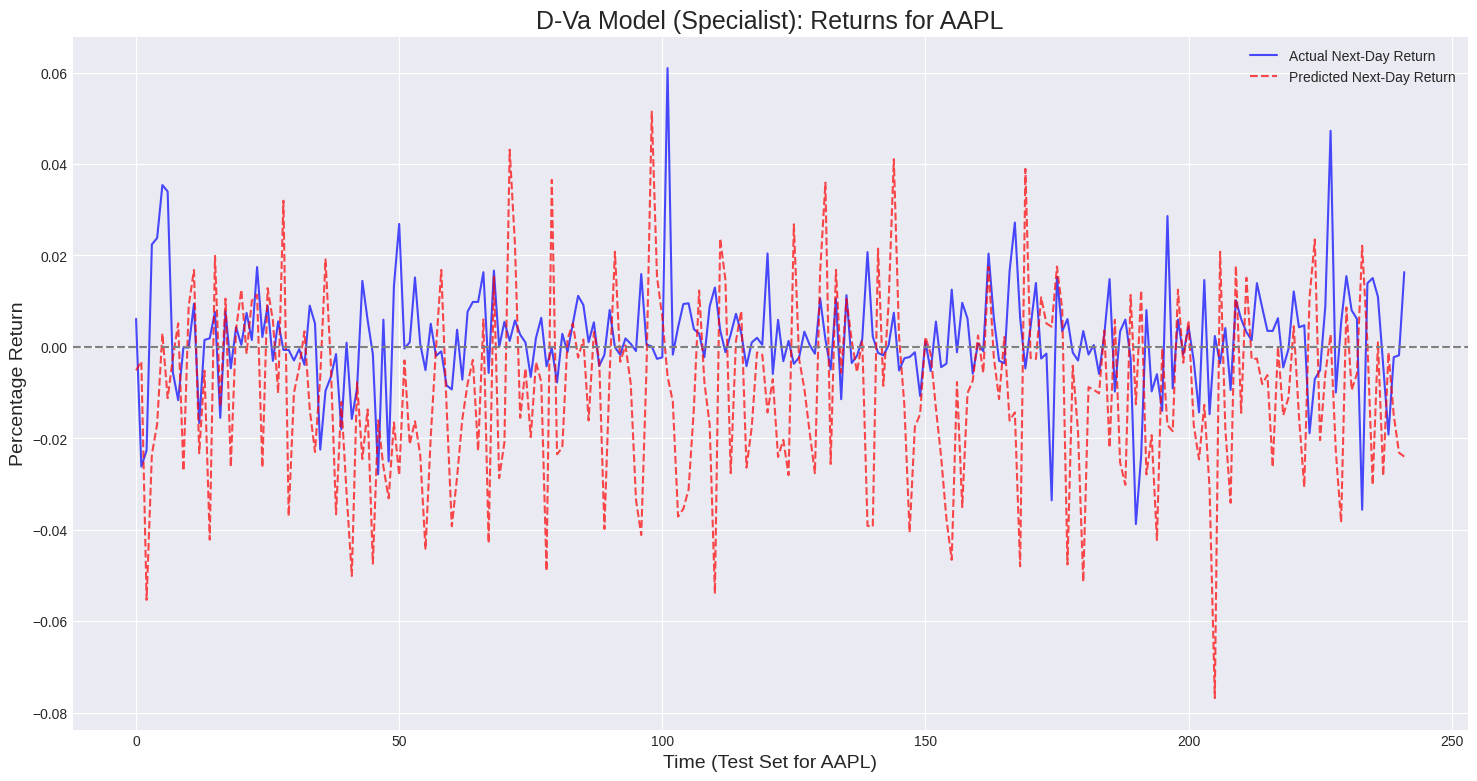
Let’s understand the model architecture from our TensorFlow code in more detail. The core is the build\_nvae function, which assembles a series of custom layers.

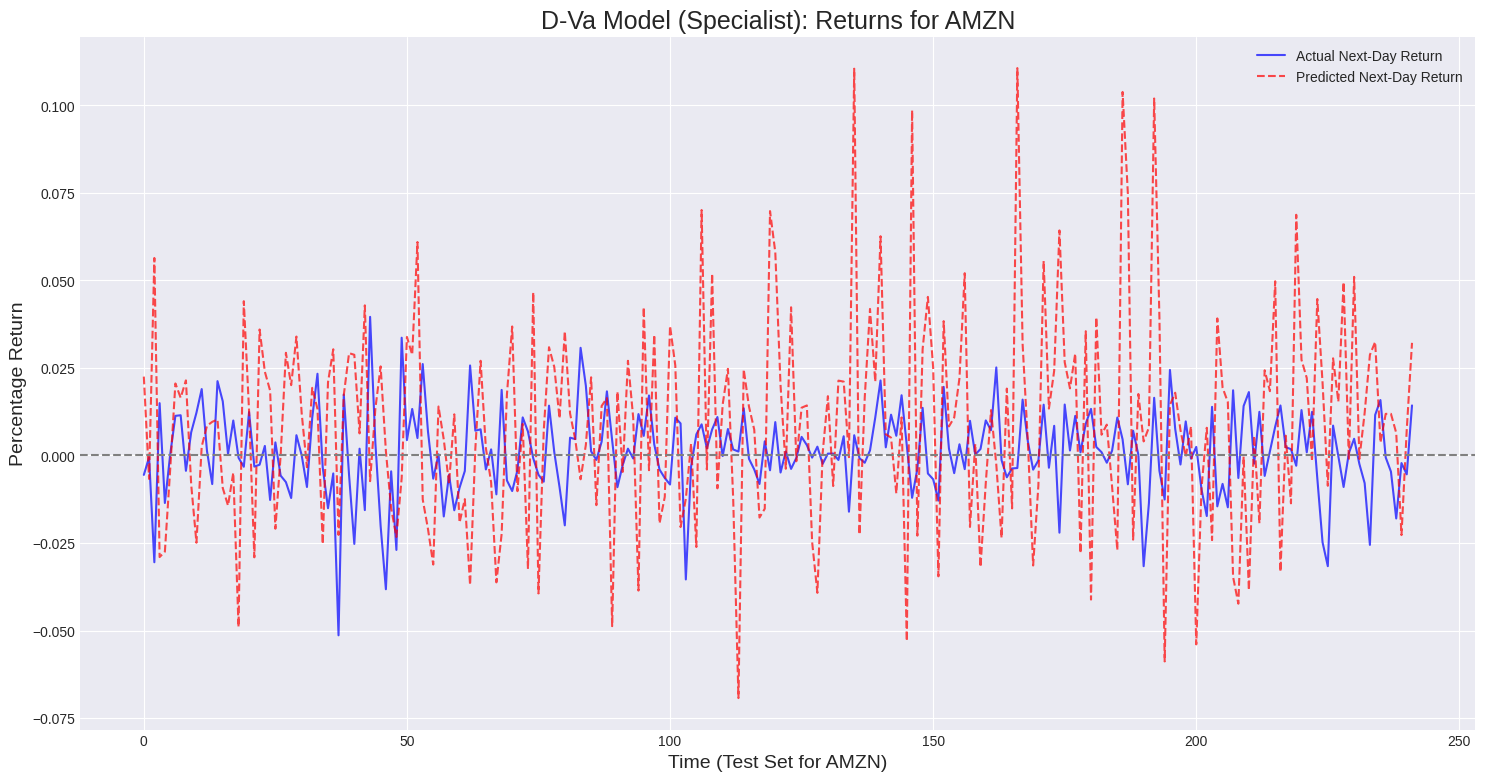
* **Input Data:** The model takes an input sequence of shape (batch\_size, 16, 5), representing a 16-day history with 5 features each day.
* **Encoder Residual Cell** (The Information Compressor): This is the workhorse of the encoder. It uses 1D Convolutional layers (Conv1D) as pattern detectors that slide across the 16-day timeline to find trends and shapes. Some of these cells use a stride=2, which halves the length of the sequence, effectively compressing the information. The encoder stacks these cells to progressively summarize the input data.
* **Squeeze-and-Excitation (SE) Block** (The "Attention" Mechanism): Inside each residual cell is an SE block. After the Conv1D layers extract features, the SE block analyzes them and decides which features are most important for the current sequence. It then amplifies the important features and suppresses the less important ones, acting as a dynamic attention mechanism.
* **Sampling Layer (The VAE Core)**: After the data is fully compressed by the encoder cells, it is passed to two Dense layers to produce z\_mean and z\_log\_var. The Sampling layer then uses the reparameterization trick (z=μ+σ⋅ϵ) to pick a single point z from the distribution defined by these parameters. This is the final, compressed "essence" of the input.
* **Decoder Residual Cell** (The Information Expander): This is the mirror image of the encoder cell. It takes the compressed data and expands it. Its key component is an UpSampling1D layer, which lengthens the sequence (e.g., from a length of 4 to 8). The decoder stacks these cells with Conv1D layers to reconstruct a full 16-day sequence from the abstract latent vector z.
* A final Conv1D layer with a single filter collapses the internal features of the decoder down to a single feature per timestep: our predicted percentage return.

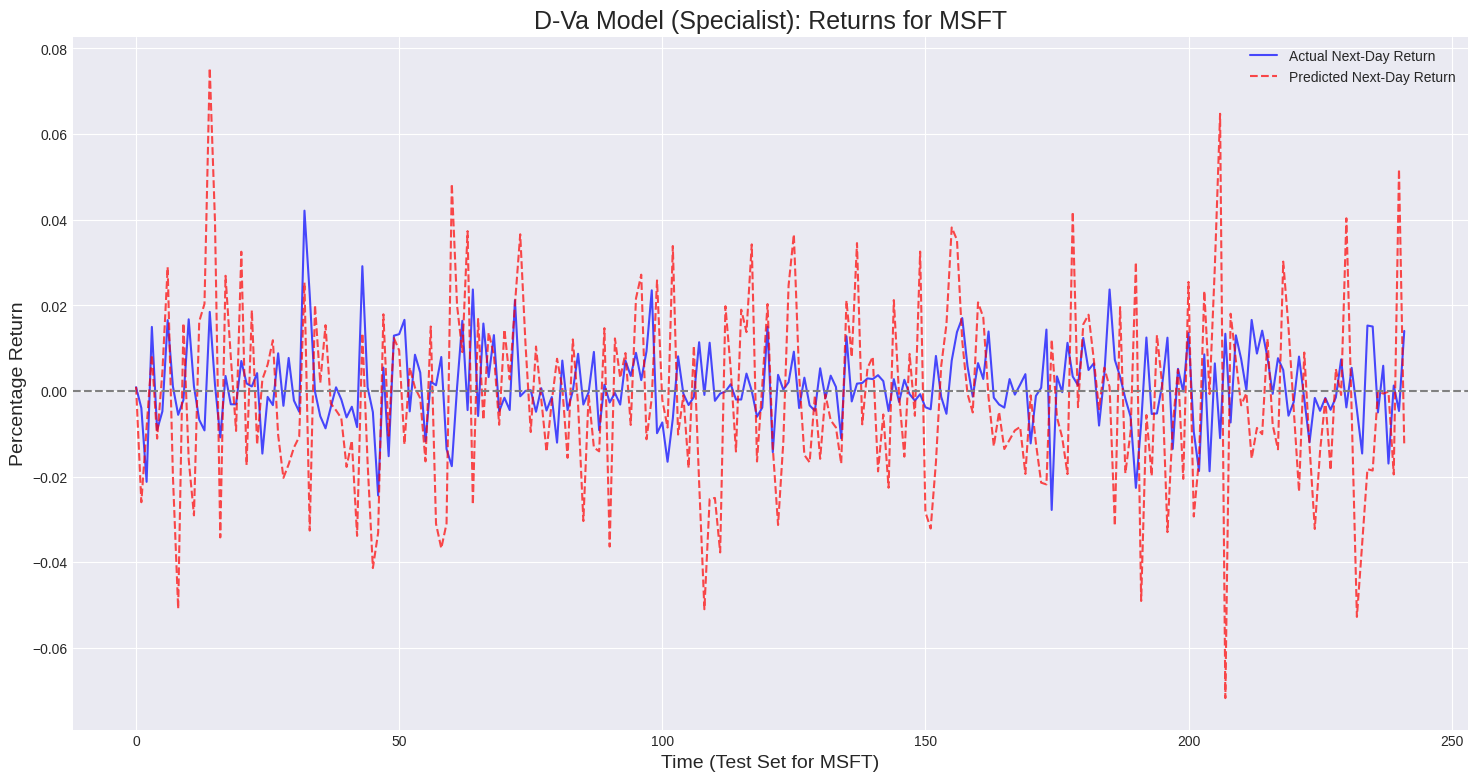
This architecture allows the model to learn a rich, hierarchical representation of the time series data, which is essential for making robust predictions.

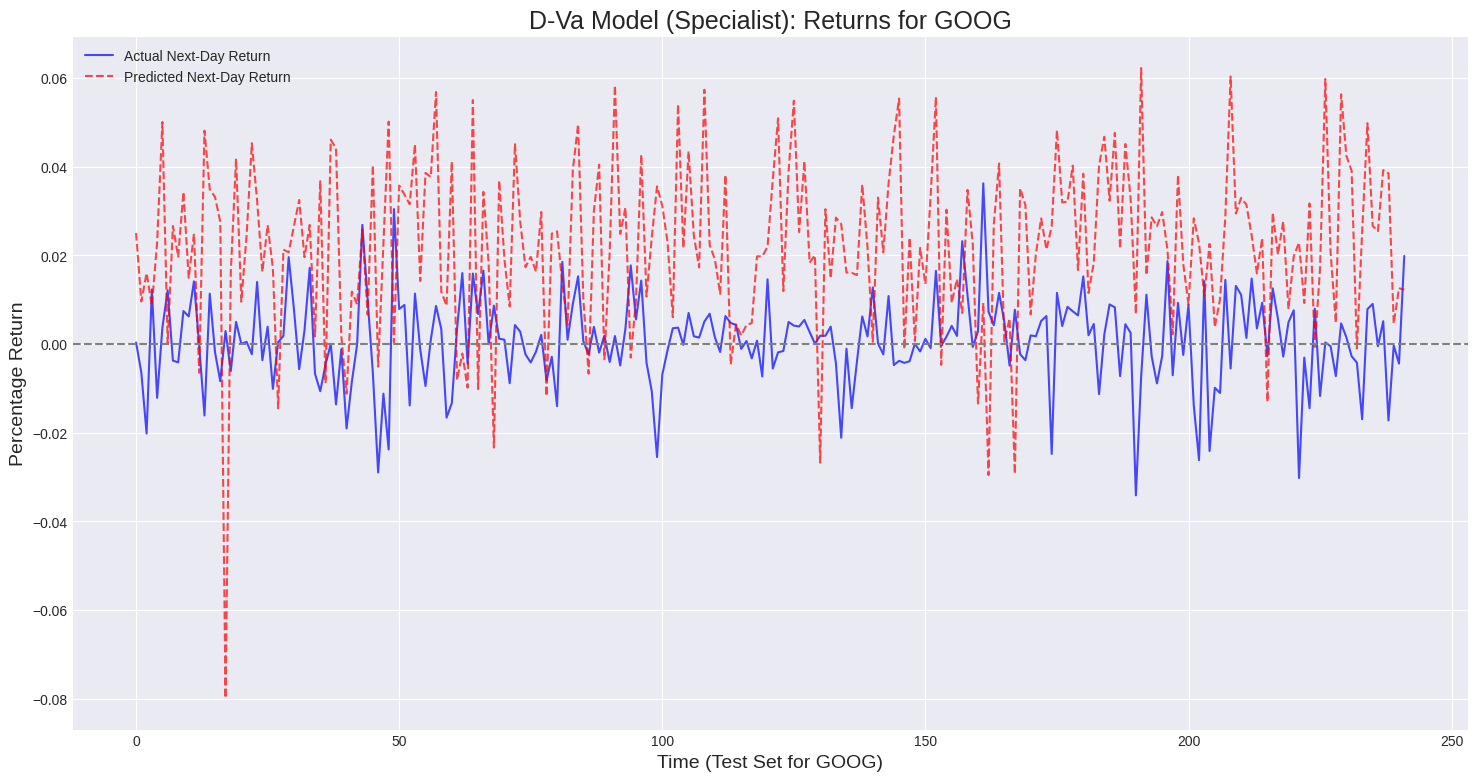
### **Results and Metrics**

After running the final "train separately" script, we obtained the following results. The model was trained to predict the next-day percentage return based on the previous 16 days of data. The graphs below show the model's predictions on the unseen test set for each stock.









| **Metrics** | **AAPL** | **MSFT** | **AMZN** | **GOOG** |
| --- | --- | --- | --- | --- |
| **MSE** | 0.025545 | 0.033607 | 0.022595 | 0.030131 |
| **MAE** | 0.020034 | 0.026333 | 0.017602 | 0.025474 |
| **Directional Accuracy** | 50.00% | 48.76% | 54.96% | 55.37% |

**Comparisons and Conclusions**

In this study, we experimented with three different models for stock price prediction: LSTM, VAE-based, and Diffusion-based models. The performance metrics showed interesting trends:

* LSTM achieved the lowest MSE and MAE (of the order of 10^-3), indicating it was most accurate in predicting the actual stock price values.
* VAE-based models performed better than diffusion models in terms of MSE and MAE but could not surpass LSTM.
* Diffusion-based models showed the highest MSE and MAE among the three.

However, when evaluating directional accuracy—that is, whether the models correctly predicted the movement direction of the stock—all models performed poorly, with accuracy in the range of 50–60%, barely above random guessing.

Possible reasons for this performance include:

1. Limited input features: Using only past prices without including broader market indicators, sentiment, or macroeconomic variables can limit predictive power.

2. Generative models like VAE and diffusion models are optimized to capture distributions and reconstruct sequences rather than predicting precise future values or market direction.

3. Optimization methods: LSTM and other neural networks optimize the loss function well (hence lower MSE/MAE) but fail to optimize directional prediction.

5. Time horizon mismatch: Generative models may excel at simulating realistic sequences over longer horizons but may not focus on next-step price prediction, which is critical for directional accuracy.

**Can GenAI models be used for stock prediction?**

While generative AI models (VAE, diffusion, transformer-based generative models) can learn complex data distributions and generate realistic stock price sequences, they are not ideal for precise next-day stock price or directional predictions. Traditional sequence models like LSTM currently outperform GenAI models in terms of numerical accuracy, but all models struggle with directional prediction due to the stochastic nature of financial markets. GenAI may supplement stock analysis, but it should not be relied upon as a standalone tool for trading decisions.

**References**

[Neural Machine Translation by jointly learning to align and translate](https://arxiv.org/pdf/1409.0473)

[DIFFSTOCK: Probabilistic Relational Stock Market Prediction Using Diffusion Models](https://share.google/tfbseitt4SSnlY5BE)

[Implementation of Diffusion Variational Autoencoder for Stock Price Prediction with Integration of Historical and Market Sentiment Data](about:blank)

<https://arxiv.org/abs/2309.00073>