Conclusion

1 Baseline vs LSTM

We compared the training time as well as memory requirement for the baseline and LSTM models. While the VAR took a fraction of a second and ARIMA took under 5 seconds, the LSTM models took significantly longer, taking up to 88.7 seconds on the CPU. This notable difference comes down to the model complexities. LSTMs process batches of data sequentially, involving intensive gradient calculations in backpropogation where the model adjusts its weights based on how much each weight contributes to the error seen at the output layer. The VAR on the other hand computes a system of linear combinations using the last p (lagged) data points, which is inherently less costly in practice. These complexities are also highlighted in the memory allocation. Most interestingly, the ARIMA required more memory space compared to the VAR. This could be caused by ARIMA's optimisation technique of maximum likelihood estimation, which is an iterative process, or the fact it has to keep hold of the residuals from the past p lagged values for moving average calculations. Instead, VAR uses ordinary least squares optimisation which involves solving a system of equations for the p lagged variables to minimise the sum of squared errors. The LSTM models on the other hand were a lot more memory costly, and even split the work between the CPU and GPU for efficiency. This cost makes sense because the LSTM takes in sequences of 10 and trains between 64-128 batches together (depending on choice) as well as storing weights, gradients, and errors in memory for the model to choose the optimal weights. It should be noted that, to the best of our knowledge, we tried to instruct the model to be trained specifically on CPU or GPU, to compare performance, it was only until we monitored the memory usage we noticed this wasn't being done as hoped.

2 ARIMA vs VAR

Despite the VAR model just being the multivariate case of the ARIMA model, in the model training phase, the VAR model was substantially faster. More specifically, the execution time for the VAR model was 6ms compared to 1.9s for the ARIMA model (this data was taken from VS code). While this is a 600x speedup, the absolute time difference is somewhat insignificant. This may come

down to the specific implementations of the models in the various libraries instead of inherent differences in the approaches as they are, in essence, the same thing. Further, the scaling might mean that for larger data, the ARIMA model becomes faster in training.

As these models are autoregressive, they use a linear combination of previous values. These correct weights are fit using linear algebra calculations. Due to this, it might be possible to speed up the training times of these models by parallelising these calculations.

We note that these models were trained on the whole dataset but only used a fixed number of lags and so it is hard to see how more data would help improve these models if they only use a very small portion of it.

3 Univariate LSTM vs Multivariate

It is clear from comparing the computation time between the two models that univariate would be preferred if the dataset was to be scaled significantly. Where the multivariate model took almost over 30 seconds longer on a relatively small dataset, there was no evidence that the extended time even resulted in better prediction accuracy. The univariate had an rmse of 22.0 compared to the multivariate's 43.5 which informs us the added features don't necessarily help in making predictions. Instead, the additional features only added complexity in the dataset, increasing the intensity of computations for the model. Overall, the univariate LSTM would be preferred if the dataset was scaled in volume because the added complexity of multivariate alone would make computation time scale at a higher rate compared to the univariate.

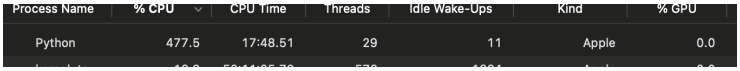
The comparison of time taken to run the models with GPU support against without did not output the results expected. We saw an increase in time to run the models with the GPU support, however we believe this to be because of the sequential nature of the LSTM. As this is a relatively small data set, the time taken to transfer data to the CPU outweigh the benefits of the parallel processing in the mini-batches. Therefore we cannot make supported conclusions as to how this might scale, however it is likely that the GPU support will have a positive effect using parallelisation for larger batch sizes, however this requires more memory usage.

Although the univariate model achieves faster computational time taken to run the model, it appears to underfit slightly the data. This could be a pitfall to its real-world usage as companies or the government would want to be able to predict the spikes in pollution for better implementation of policies, rather than have a conservative prediction. So it may be useful to have a mixture of two

models to better capture complexities in the data as to not tend towards the mean too much, while reducing time taken to run the model for better scalability purposes.

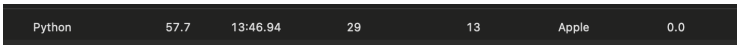
4 Appendix

The following is the memory allocation involved during the training process of each of our "best" models.



Process Name	% CPU	CPU Time	Threads	Idle Wake-Ups	Kind	% GPU
Python	477.5	17:48.51	29	11	Apple	0.0

Figure 1: ARIMA

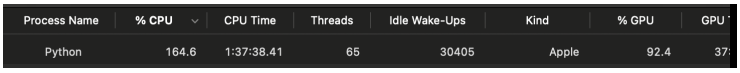


Python	57.7	13:46.94	29	13	Apple	0.0

Figure 2: VAR

Process Name	% CPU	CPU Time	Threads	Idle Wake-Ups	Kind	% GPU	GPU
Python	140.0	40.58	70	14725	Apple	93.2	

Figure 3: UV LSTM



Process Name	% CPU	CPU Time	Threads	Idle Wake-Ups	Kind	% GPU	GPU
Python	164.6	1:37:38.41	65	30405	Apple	92.4	37

Figure 4: MV LSTM