Appendix A: Traditional Machine Learning

this appendix details the reasoning behind different hyper parameters in Statistical & Traditional machine

Random Forest	n_estimators	min_samples_split	min_samples_leaf	max_features	max_depth	bootstrap				
Optuna	100	4	4	None	11	FALSE				
RGS	150	10	4	None	15	FALSE				
XGBoost	subsample	reg lambda	reg alpha	number of estimator	max depth	learning rate	gamma	colsample bytree		
Optuna	1	1	0.1	150	5	0.1	0	0.8		
RGS	1	2.444	0.015	200	3	0.0172	0.4	0.4		
CatBoost	subsample	random strength	random seed	learning rate	l2 leaf reg	iteration	depth	colsample by	border count	bagging
Optuna	0.72	8	55	0.029	1.088	1000	3	0.946	203	0.838
RGS	1	-	-	0.1	5	50	5	-	128	

***Figure 1**. ML Models Hyperparameters*

Random Forest

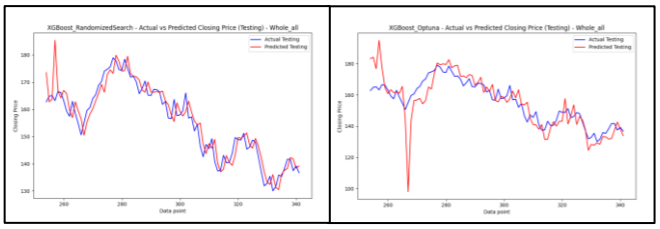
Figure 3 shows the hyperparameters chosen by the two methods. RGS selects a higher "n_estimators", indicating more trees, which reduces bias and variance but increases computational cost. RGS also has higher values for "min_samples_split" and "max_depth", expanding tree breadth and depth. Both methods have bootstrapping set to "false", meaning each tree uses the full training data, which increases the risk of overfitting.

XGBoost

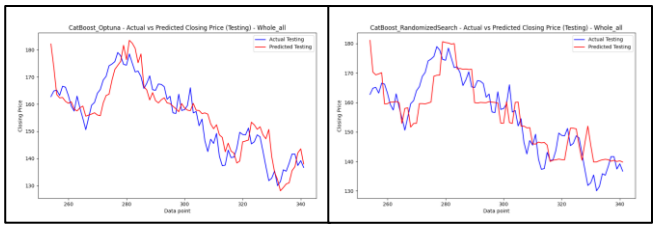
In both methods, "subsample" is set to 1, meaning each tree uses the entire training dataset. RGS has a higher "reg_lambda" (L2 regularization), while Optuna has a higher "reg_alpha" (L1 regularization), both of which penalise complex trees to prevent overfitting. RGS also has a higher "n_estimators", enabling more iterations to refine the tree. Optuna's higher "max_depth" allows it to capture complex interactions, though with an increased overfitting risk. Optuna also has a higher learning rate, enabling faster weight updates. RGS's higher "gamma" imposes a stricter threshold for leaf node partitioning, and RGS's higher "colsample_bytree" uses a larger proportion of features in each tree.

CatBoost

The "subsample" parameter controls the fraction of the dataset used for training each tree. Higher values lead to more trees, which can improve stability but may cause overfitting. "Random strength" adjusts the noise added to the tree to reduce overfitting, though higher values may reduce model accuracy. The "learning rate" affects convergence speed; a higher rate speeds convergence but may prevent reaching the local minimum. "L2 leaf reg" adds penalties to leaf weights to avoid overfitting but can cause underfitting if set too high. Other hyperparameters, such as "depth", adjust each tree's structure.



***Supplementary figure 2.** Optuna and RGS comparison for XGBoost*



***Supplementary figure 3.** Optuna and RGS comparison for CatBoost*

Appendix B: Statistical & Traditional Machine Learning

This appendix contains important hyperparameter selection choices using optima and RGS

ANN	batch size	epochs	layers	neurons	activation	learning rate	dropout rate	optimizer
Optuna	64	44	3	64	relu	0.000224	0.156	adam
RGS	64	50	5	16	relu	0.01	0.5	adam
RNN	**batch size**	**epochs**	**layers**	**hidden units**	**grad clip**	**learning rate**	**dropout rate**	**optimizer**
Optuna	32	24	2	69	3	0.0104	0.067	adam
RGS	32	200	3	50	5	0.00005	0.25	adam
GRU	**batch size**	**epochs**	**layers**	**hidden units**	**grad clip**	**learning rate**	**dropout rate**	**optimizer**
Optuna	32	10	1	116	5	0.0247	0.244	adam
RGS	128	50	3	128	3	0.001	0.3	adam
LSTM	**batch size**	**epochs**	**layers**	**number of units**	**optimizer**	**learning rate**	**dropout rate**	
Optuna	64	20	7	64	adam	0.00500	0.2	
RGS	64	20	5	100	adam	0.005	0.2	

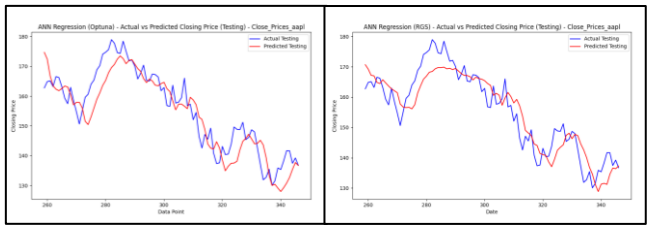
Figure 2. *DL Models Hyperparameters*

ANN - The hyperparameter tuning methods, Random Grid Search (RGS) and Optuna, produce similar values for some parameters but differ significantly for others. Both methods select the same batch size, while RGS uses slightly more epochs, indicating a slower convergence rate. Optuna's configuration, however, includes fewer layers but significantly more neurons per layer, along with a much smaller learning rate. This allows for precise convergence but at a higher computational cost.

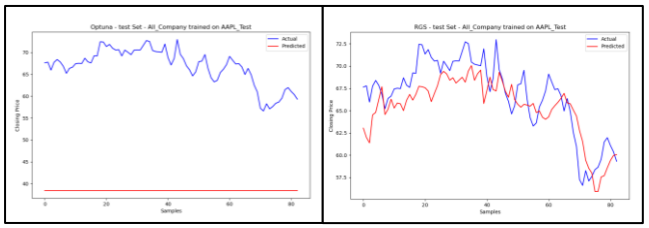
RNN - For RNN, Optuna and RGS show similar patterns to ANN tuning. Batch sizes are identical, with RGS choosing more epochs but with greater magnitude differences. RGS also opts for more layers and fewer neurons per layer. In contrast, Optuna uses a higher learning rate and lower gradient clipping, favouring more precise convergence, while RGS's higher gradient clipping accelerates updates. Additionally, Optuna's lower dropout rate indicates less regularization, likely due to the model's simplicity compared to RGS's configuration.

GRU - In GRU tuning, Optuna favours quicker convergence and lower complexity, using a smaller batch size to capture finer details. RGS, with more epochs, layers, and neurons, aims to capture more complex patterns. Optuna's higher learning rate and gradient clipping lower computational demands, emphasizing the balance between accuracy and efficiency in tuning.

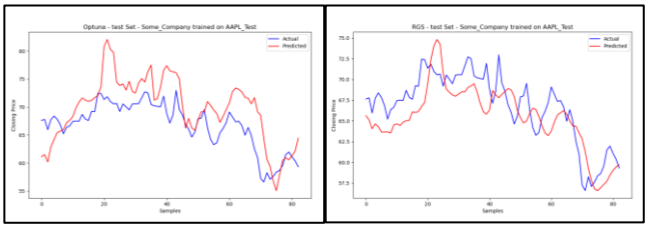
LSTM - In the final tuning, Optuna and RGS match on batch size, epochs, and dropout rate. Unlike previous configurations, Optuna now opts for more layers with fewer neurons per layer, differing from earlier patterns.



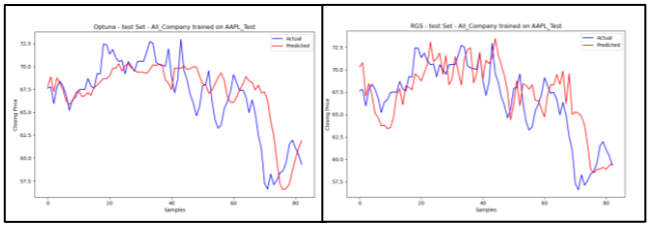
Supplementary figure 4. *Optuna and RGS comparison for ANN*



Supplementary figure 5. *Optuna and RGS comparison for RNN*



Supplementary figure 6. *Optuna and RGS comparison for GRU*



Supplementary figure 7. *Optuna and RGS comparison for LTSM*