Deep Learning Coursework Assignment

Low-Rank Adaptation of Large Language Models for Time Series Forecasting

Laura Just Fung (lj441)

April 7, 2025

Word count: 2996

1 Introduction

As shown by Gruver et al., by encoding time series data as a string of digits, time series forecasting can be directly translated to next-token prediction in text. Thus, large language models (LLMs) can be zero-shot repurposed to this task fairly naturally, helping tackle this challenging machine learning problem. In this report, the Qwen2.5-0.5B-Instruct model [3] along with Low-Rank Adaptation (LoRA) of the q and v projection matrices are used to explore this concept further and demonstrate performance of a LLM that has been fine-tuned towards this task.

2 Compute constraints

A budget of 10^{17} floating point operations (FLOPS) was applied to this coursework. The FLOPS accounting for primitives are defined in Table 1. The extrapolated FLOPS for other operations are shown in Table 2.

Operations	FLOPS
Addition/Subtraction/Negation	1
Multiplication/Division/Inverse	1
ReLU/Absolute value	1
Exponentiation/Logarithm	10
Sine/Cosine/Square root	10

Table 1: Standardised FLOPS for common primitive operations as defined for the rest of this report.

While Qwen2.5-0.5B-Instruct uses SwiGLU activations as well as Grouped Query Attention, the FLOPS for SiLU activations as well as standard multi-headed attention have been used to simplify the calculations. For similar reasons, it has been assumed that the FLOPS of backpropagation are exactly twice that of the forward pass. Additionally, for all reported FLOPS, any operations performed outside of the model are not included. Finally, the cost of computing the rotary positional encodings have been ignored.

Operation	FLOPS
$M_{m\times n} \times M_{n\times p}$ matrix multiplication	$mp(2n-1)$
SiLU($M_{m\times n}$)	$13mn$
RMSNorm($M_{m\times n}$)	$m(5n+10)$
Softmax($M_{m\times m}$)	$m(12m-1)$
Embedding RoPE($M_{m\times n}$)	mn
Rank r LoRA($M_{m\times m}$)	$2rm^2$

Table 2: Extrapolated FLOPS for operations done whilst training and validating the LLM with LoRA.

Using Tables 1 and 2, a Python script to calculate the FLOPS as a function of the model's hyperparameters and input size was created. The script is located in `src/flops.py`.

3 Preprocessing

A text-based numeric encoding method adapted from the time series data preprocesing scheme described by Gruver et al. was implemented for all data given to Qwen2.5-0.5B-Instruct. This preprocessing ensures that the numeric sequences are appropriately formatted for the best performance of the Qwen2.5-0.5B-Instruct model.

First, as the numeric values in the dataset may vary, to standardise the numeric range of the data and control the token length, a simple scaling was applied, as shown in Eq. 1

$$x_t' = \frac{x_t}{\alpha}, \tag{1}$$

where α is chosen based on the distribution of the dataset.

Additionally, the scaled numeric values were rounded to a fixed number of decimal places to ensure uniformity when tokenized.

The data used for this coursework involves multivariate Lotka-Volterra time series data. Thus, the following encoding is used, as shown in Eq. 2:

$$\begin{pmatrix} P_0 & Q_0 \\ P_1 & Q_1 \\ \vdots & \vdots \\ P_t & Q_t \end{pmatrix} \rightarrow p_0, q_0; p_1, q_1; \ldots; p_t, q_t, \tag{2}$$

where P_t refers to the original prey value at time t and Q_t refers to the original predator value at time t. The lowercase p_t and q_t refer to the scaled and rounded values of the prey and predator series. The comma is used to separate the predator and prey values, while the semicolon is used to separate the time steps.

The code for this preprocessing is shown in Listing 1, and this function can be found in the `preprocessor.py` file in the `src` directory.

```python
def scale_and_encode(prey, predator, alpha, decimals):
    """
    Scale and encode the prey and predator data.

    Args:
        prey (np.ndarray): The prey values.
        predator (np.ndarray): The predator values.
        alpha (float): Scaling factor.
        decimals (int): Number of decimal places for scaling.
    Returns:
        encoded (str): The encoded data string.
    """
    prey = np.array(prey)
    predator = np.array(predator)
    data = np.stack([prey, predator], axis=-1)
    rescaled = data/alpha * 10
    rescaled = np.round(rescaled, decimals=decimals)
    series = np.column_stack((rescaled[:, 0], rescaled[:, 1]))
    encoded = ';'.join([','.join(map(str, row)) for row in series])
    return encoded
```

Listing 1: Function to preprocess the time series data. The function takes in the prey and predator values, the scaling factor α, and the number of decimal places to round to. It returns the encoded string representation of the time series data.

Throughout, $\alpha = 5$ and the values were rounded to 3 decimal places. Two examples are shown below in Tables 3 and 4.

Stage	Value
Raw data	$\begin{pmatrix} 1.1335121 & 1.1031258 \\ 0.55542254 & 1.2579137 \end{pmatrix}$
Scaled data	$\begin{pmatrix} 2.267 & 2.206 \\ 1.111 & 2.516 \end{pmatrix}$
Encoded data	2.267,2.206;1.111,2.516
Tokenized data	[17,13,17,21,22,11,17,13,17,15,21,26,16,13,16,16,16,11,17,13,20,16]

Table 3: Example of the preprocessing method for two steps in the time series data. The raw data is scaled, encoded, and then tokenized.

Stage	Value
Raw data	$\begin{pmatrix} 0.8521567 & 0.9479834 \\ 0.6482769 & 0.94272685 \end{pmatrix}$
Scaled data	$\begin{pmatrix} 2.267 & 2.206 \\ 1.111 & 2.516 \end{pmatrix}$
Encoded data	1.704,1.896;1.297,1.885
Tokenized data	[16,13,22,15,19,11,16,13,23,24,21,26,16,13,17,24,22,11,16,13,23,23,20]

Table 4: Another example of the preprocessing method for two steps in the time series data.

4 Baseline

Using the preprocessing method described in Section 3, the untrained model was evaluated on the tokenized Lotka-Volterra time series data. It was decided that the following plots and metrics hereafter would be using the scaled and rounded values of the time series data to compare to the predictions. This is the format that the model is trained upon and rescaling the data back to the original values would not be useful for evaluating the model's performance as it would introduce a source of error that the model cannot account for.

To evaluate the performance of the untrained model, stratified sampling was used to select four systems from the given dataset. Broadly speaking, there are four types of system behaviour present in the dataset: oscillatory, decaying, growing, and stable.

The model was given the first 80 points in the time-series data and asked to predict the next 20 points, as shown in Fig. 1. The model was evaluated using mean absolute error (MAE), the mean absolute percentage error (MAPE), the mean squared error (MSE), the root mean squared error (RMSE), and running mean squared error (RMSE) for both the predator and prey time series data. The metrics for an example of each of these types are shown in Table 5 and in Fig. 2.

	506		20		654		906	
Metric	Prey	Predator	Prey	Predator	Prey	Predator	Prey	Predator
MSE	0.217	0.341	12.207	10.331	0.001	0.000	0.023	0.003
RMSE	0.465	0.583	3.494	3.214	0.030	0.001	0.151	0.050
MAE	0.372	0.471	2.385	2.312	0.028	0.001	0.103	0.032
MAPE	0.301	0.365	1.332	0.760	0.004	0.001	0.031	0.056

Table 5: Metrics for the untrained Qwen2.5-0.5B-Instruct model performance on a subset of the test set. The metrics are calculated for the 20 points comprising the out-of-distribution data only.

As can be seen in Fig. 1, the untrained model is fairly capable at predicting the oscillatory and stable systems 906 and 654 respectively, but is unable to predict decaying and growing systems 506 and 20 to a reasonable degree of accuracy. This is reflected in the metrics shown in Table 5, where the predator and prey metrics for systems 654 and 906 are all very small, but the metrics for systems 506 and 20 are significantly greater. The model is especially terrible at predicting system 20, the growing system. Perhaps this is because oscillatory and stable systems have very regular and obvious patterns, which are easier to predict than the decaying and growing systems.

What is also important is where in the time series that the cut-off point is placed. This is demonstrated especially by system 654, which settled into a fairly stable and predictable state before the cut-off point, and so the model is able to predict the next 20 points with a very small error. This is also shown by sytem 506, where the model was able to predict the

rough shape of the curve, but not the exact values for the prey time series. The cut-off point occurred just after the peak of the prey time series but before the peak of the predator time series. Perhaps it was this distinction that allowed the model to 'know' that the prey time series should be at least oscillatory, yet remain ignorant of the similar nature of the predator time series.

Without fine-tuning, it is clear that the model is unable to learn the more subtle underlying patterns in the time series data. This is likely due to the fact that as it is a large language model, it has probably not been previously trained on time series forecasting tasks.

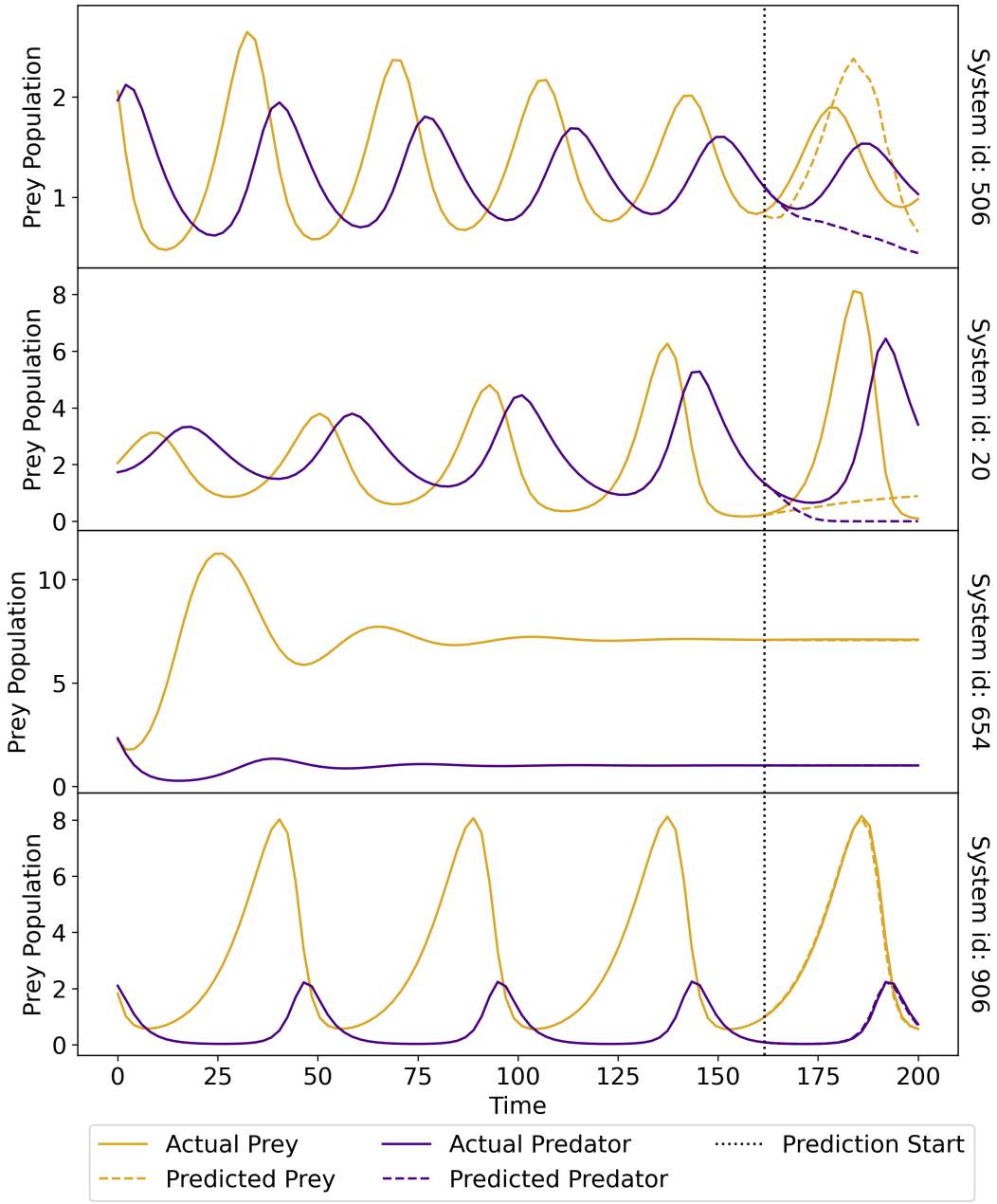

Figure 1: Predictions of the untrained Qwen2.5-0.5B-Instruct model on a subset of the Lotka-Volterra time series data for both predator (purple) and prey (gold). The model's predictions are shown in the dashed lines, while the true values are shown in the solid lines.

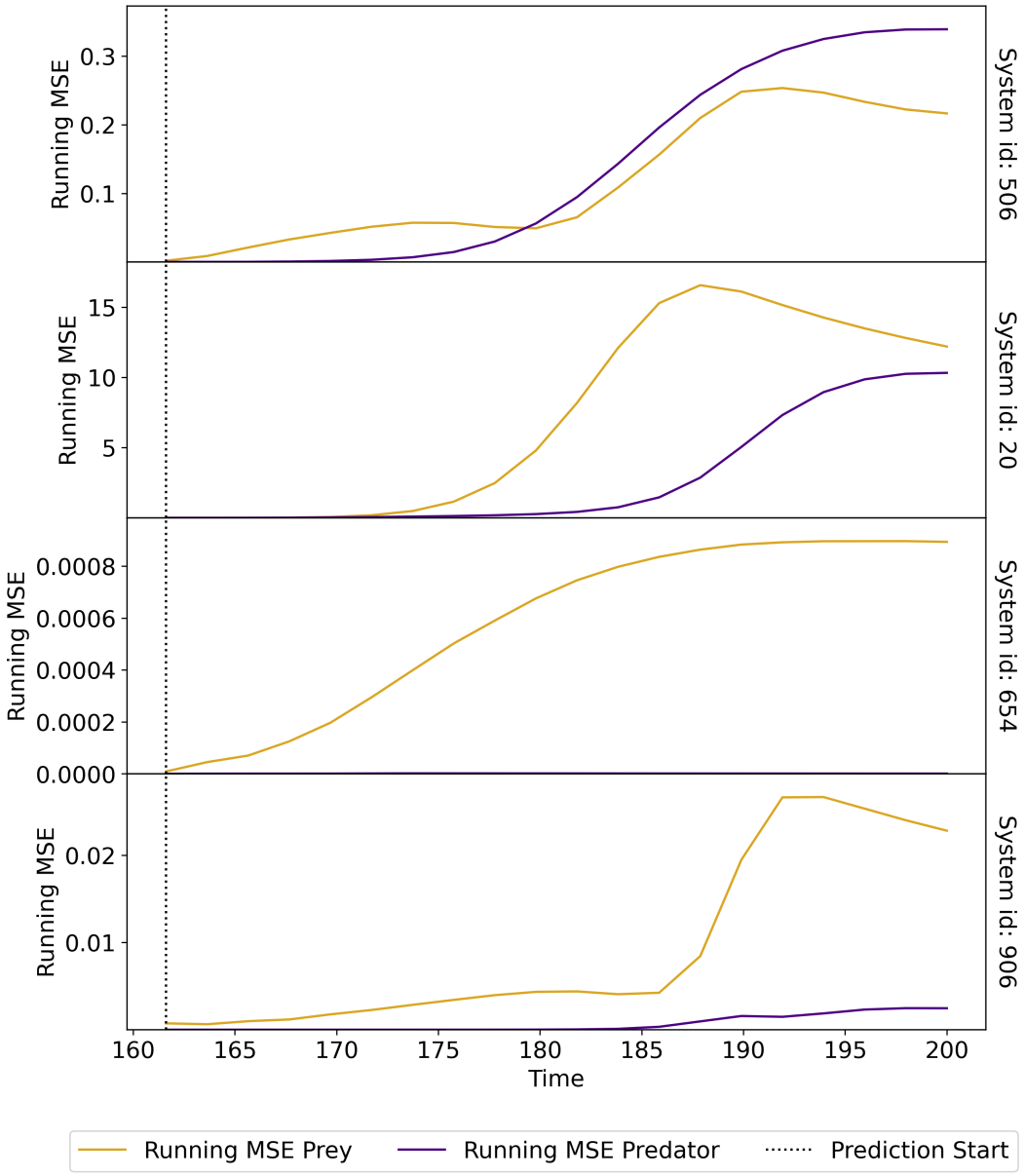

Figure 2: Running mean squared error (RMSE) for the untrained Qwen2.5-0.5B-Instruct model on a subset of the test set over the out-of-distribution data for the predator (purple) and prey (gold) time series.

5 LoRA

The Qwen2.5-0.5B-Instruct model was fine-tuned using the LoRA [2] implementation to wrap the query and value projection layers with LoRALinear layers. This adds low-rank matrices to the model's weights for the query and value matrices, which allows for efficient training and adaptation to new tasks. Then, when training the model, only the LoRA weights are updated whilst the rest of the model's weights are frozen. This allows for efficient training and adaptation to new tasks, as this approach allows for the model to be fine-tuned on a specific task without the need for retraining the entire model, which can be computationally expensive and time-consuming.

The specific parameter blocks that are being tuned when using LoRA are the A and B matrices added to the query and value projections in each transformer layer's self-attention block. The forward pass of the LoRALinear layer then combines the original layer's output with a scaled low-rank output produced by the A and B matrices, as shown in Eq. 3:

$$\text{output} = \text{original}(x) + \frac{\alpha}{r} \cdot (x \cdot A^T \cdot B^T), \tag{3}$$

where r is the rank of the LoRA matrices, and A and B are the low-rank matrices added to the query and value projections, respectively. α is a scaling factor that controls the magnitude of the LoRA output's contribution to the update and is set to be equal to r throughout.

5.1 Training with LoRA

To demonstrate the baseline improvements when adding LoRA, the Qwen2.5-0.5B-Instruct model was fine-tuned using the default hyperparameters shown in Table 6 for 4000 time steps. The model was evaluated on the validation set every 50 steps. The training and validation loss are shown in Fig. 3, and the model's performance on stratified examples from the test set is shown in Fig. 4. The metrics for the model's performance on the test set are shown in Table 7.

Hyperparameter	Value
Learning rate	10^{-5}
Batch size	4
LoRA rank	4
LoRA scaling factor α	4
Maximum context length	512

Table 6: Default hyperparameters used for training the Qwen2.5-0.5B-Instruct model with LoRA on the Lotka-Volterra time series data.

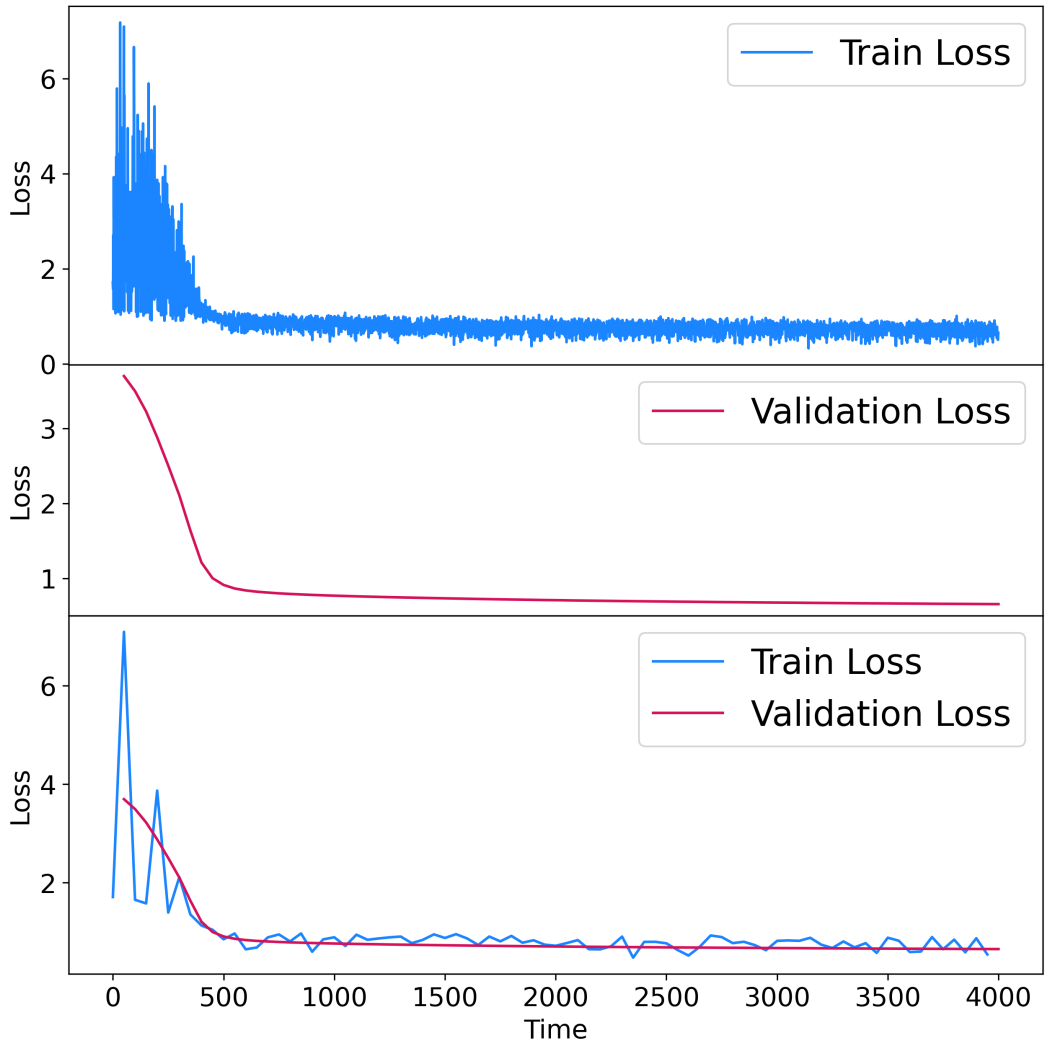

Figure 3: Training and validation loss for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data using default hyperparameters. The bottom figure compares the training (blue) and validation (red) loss using the same resolution of every 50 steps whilst the top figure shows the training loss at every step and the middle figure shows the validation loss at every 50th step.

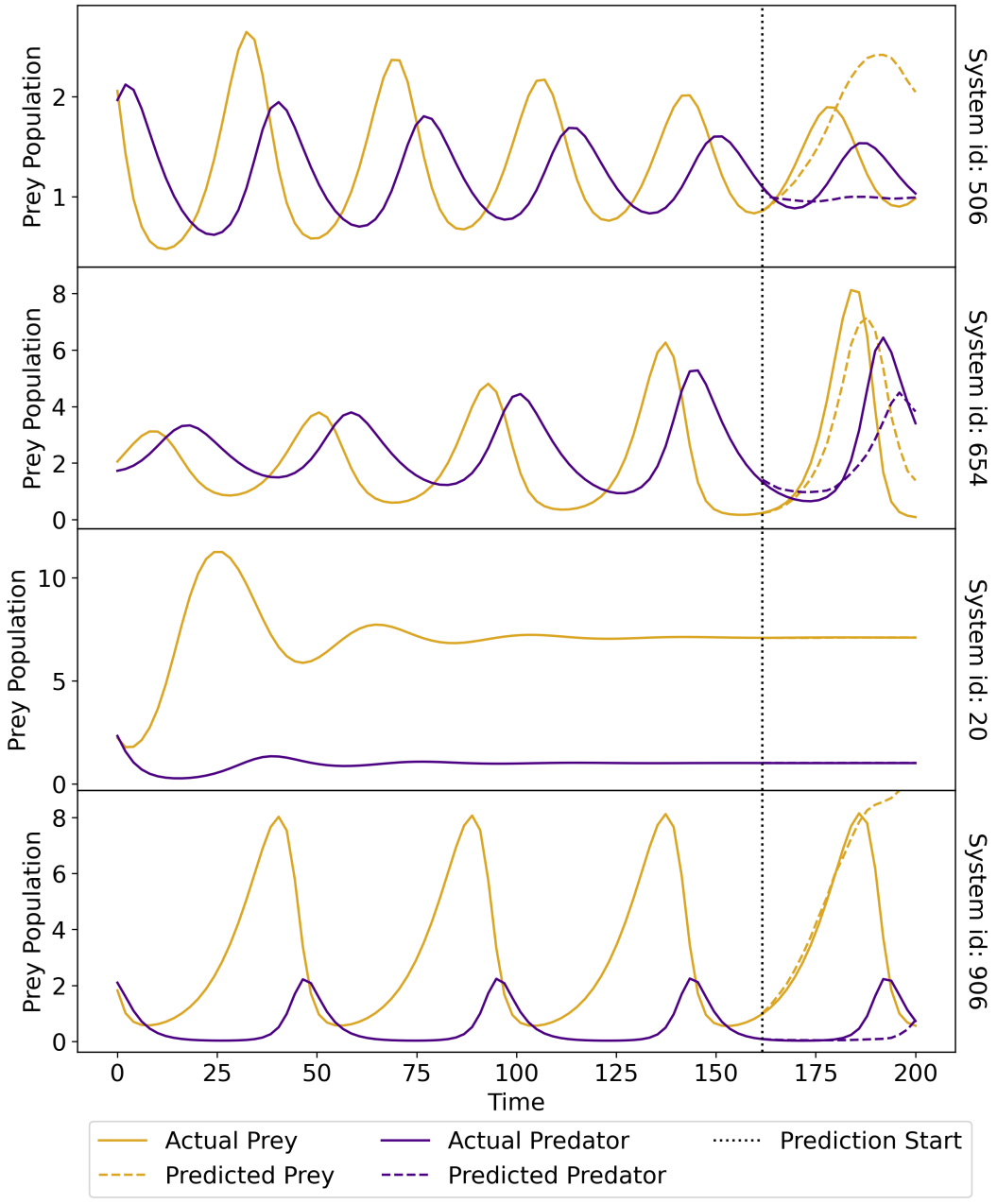

Figure 4: Predictions of the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA using default hyperparameters on a subset of the test set for both predator (purple) and prey (gold). The model's predictions are shown in the dashed lines, while the true values are shown in the solid lines.

	506		654		20		906	
Metric	Prey	Predator	Prey	Predator	Prey	Predator	Prey	Predator
MSE	0.675	0.083	2.914	1.486	0.000	0.000	14.832	0.720
RMSE	0.821	0.288	1.707	1.219	0.009	0.004	3.851	0.849
MAE	0.609	0.216	1.313	0.753	0.007	0.003	2.147	0.473
MAPE	0.562	0.162	2.220	0.276	0.001	0.003	2.152	0.524

Table 7: Metrics for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA using default hyperparameters on a subset of the test set. The metrics are calculated for the 20 points comprising the out-of-distribution data only.

As can be seen in Fig. 3, the training and validation loss both decrease over time, indicating that the model is learning from the data. There is a gradual plateau in the training and validation loss starting around 1,000 steps, indicating that the model is converging to a local minimum. The validation loss is comparable to the training loss but generally higher, which indicates that the model is not overfitting to the training data.

As shown by Fig. 4, the model performs somewhat better on predicting the growing and decaying systems, as it seems to understand that there is an underlying pattern of growing or decaying oscillations. However, it now struggles with applying the same knowledge to the oscillatory system 906, which now has fairly large errors in both the prey and predator time series predictions. This is in large contrast to before, where it was able to predict it almost perfectly, as seen in Figures 1 and 2. This is likely because very few oscillatory systems are present in the entire Lotka-Volterra dataset, with most of the systems being some form of decaying or growing. Thus, the model has learned to predict the more common systems better than the less common systems.

The gradients of the A and B matrices were tracked throughout the training process and it was found that the gradients of the A and B matrices neither exploded nor vanished. Thus, gradient clipping was not applied to the model. As the figure is quite large, comprising of 48 plots, it is not shown in this report and is instead included as supplementary material in the `addendum` folder under the name `lora_default_gradients.png`.

5.2 Hyperparameter tuning

To further improve the performance of the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA, hyperparameter tuning was performed. The hyperparameters that were tuned are shown in Table 8. They were tuned using a grid search approach, and the best combination of hyperparameters was then selected based on the final validation loss.

Hyperparameter	Values
Learning rate	10^{-5}, 5×10^{-5}, 10^{-5}
LoRA rank	2, 4, 8
Maximum context length	128, 512, 768

Table 8: Hyperparameters tuned for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data.

Learning rate	LoRA Rank		
	2	4	8
1×10^{-5}	2.932	2.154	1.046
5×10^{-5}	0.770	0.736	0.706
1×10^{-4}	0.713	0.686	0.661

Table 9: Final validation loss for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data using different combinations of η and r.

5.2.1 Learning rate and LoRA rank

The learning rate η and LoRA rank r were tuned first, with the maximum context length set to 512. The model was trained for 300 steps with each combination of hyperparameters. The training and validation loss are shown in Fig. 5.

As can be seen in Fig. 5, η is has a significant impact on the model's training, with the largest learning rate $\eta = 0.0001$ causing significant decreases in both training and validation loss in in the first 100 steps regardless of r. This is likely because with a larger η, the model is able to make large initial updates to the weights, which allows it to learn quickly from the data. However, this can also lead to instability in the training process, as the model may overshoot the optimal weights and diverge. Fig. 5 also shows that r also has a large impact on the model's training, as the larger the contribution the LoRA matrices are allowed to make to the model's output, the better the model is able to learn to learn from the data. This is unsurprising, as the LoRA weights are the only weights being updated during training.

The final validation loss for each of the combinations of hyperparameters is shown in Table 9. From this it can be seen that the best combination of hyperparameters is be $\eta = 10^{-4}$ and $r = 8$.

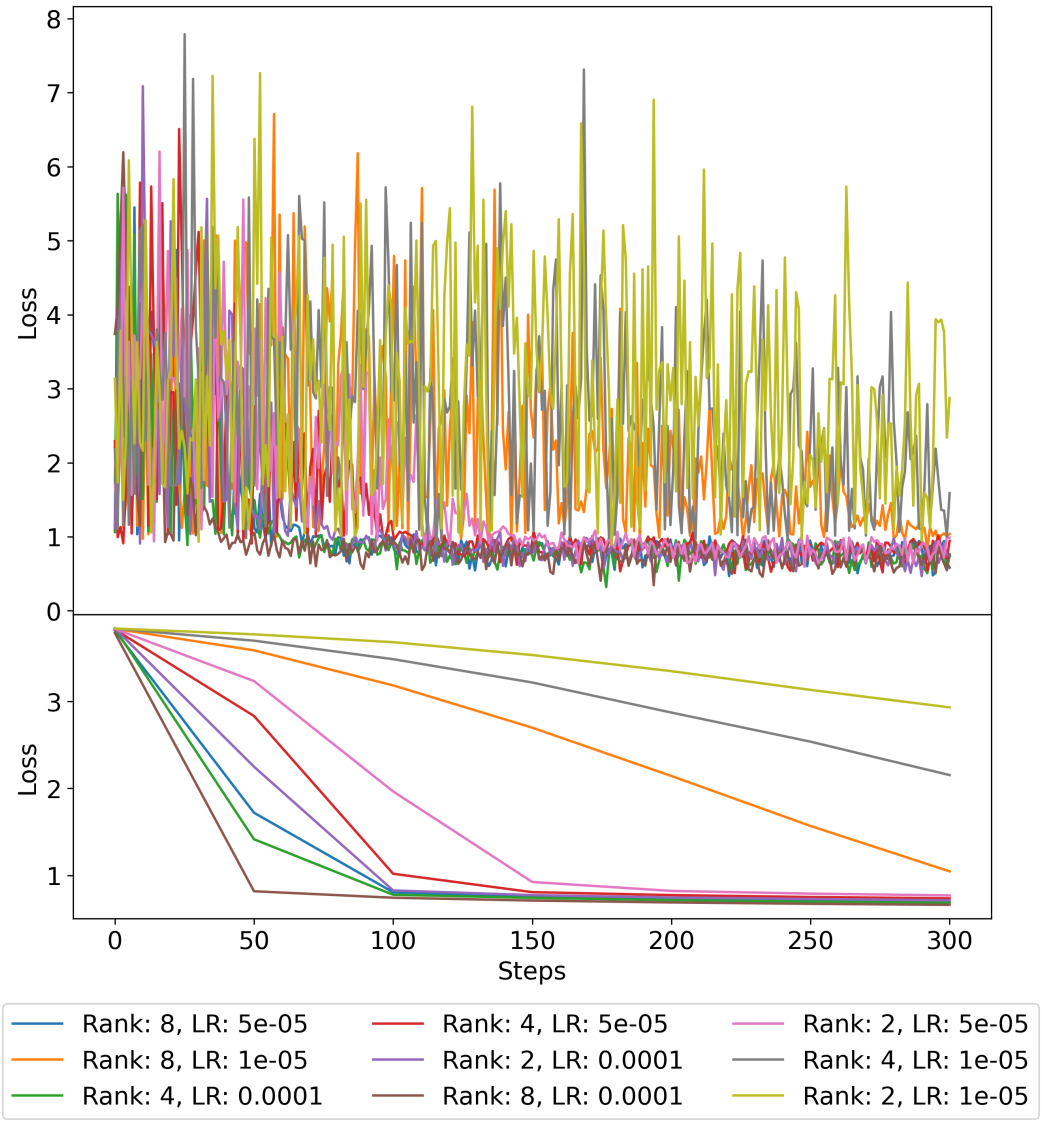

Figure 5: Training (top) and validation (bottom) loss for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data using different η and r.

5.2.2 Maximum context length

The maximum context length was then tuned, setting $\eta = 10^{-4}$ and $r = 8$. The model was trained for 300 steps with each context length. The training and validation loss are shown in Fig. 6.

As can be seen in Fig. 6, the maximum context length has a significant impact on the model's training, up to a point. With a too-small maximum context length, the model is unable to effectively learn from the data, as it is only able to see a small portion of the time series data at any one time. However, the validation loss of the model using a maximum context length of 512 is very similar to that of a maximum context length of 768, indicating that the model is able to learn from the data effectively with either maximum context length. This is also shown in Table 10, the longest maximum context length of 768 results in the lowest final validation loss, but not significantly lower than that of a maximum context length of 512. The training loss shown in Fig. 6 also shows this.

The experiments were only run for a maximum of 300 steps as the model's training and validation loss tends to plateau at around 50–100 steps.

	Maximum Context Length		
	128	512	768
Validation loss	0.895	0.664	0.661

Table 10: Final validation loss for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data using different maximum context lengths with $\eta = 0.0001$ and $r = 8$.

The best maximum context length was found to be 768, which resulted in the lowest validation loss.

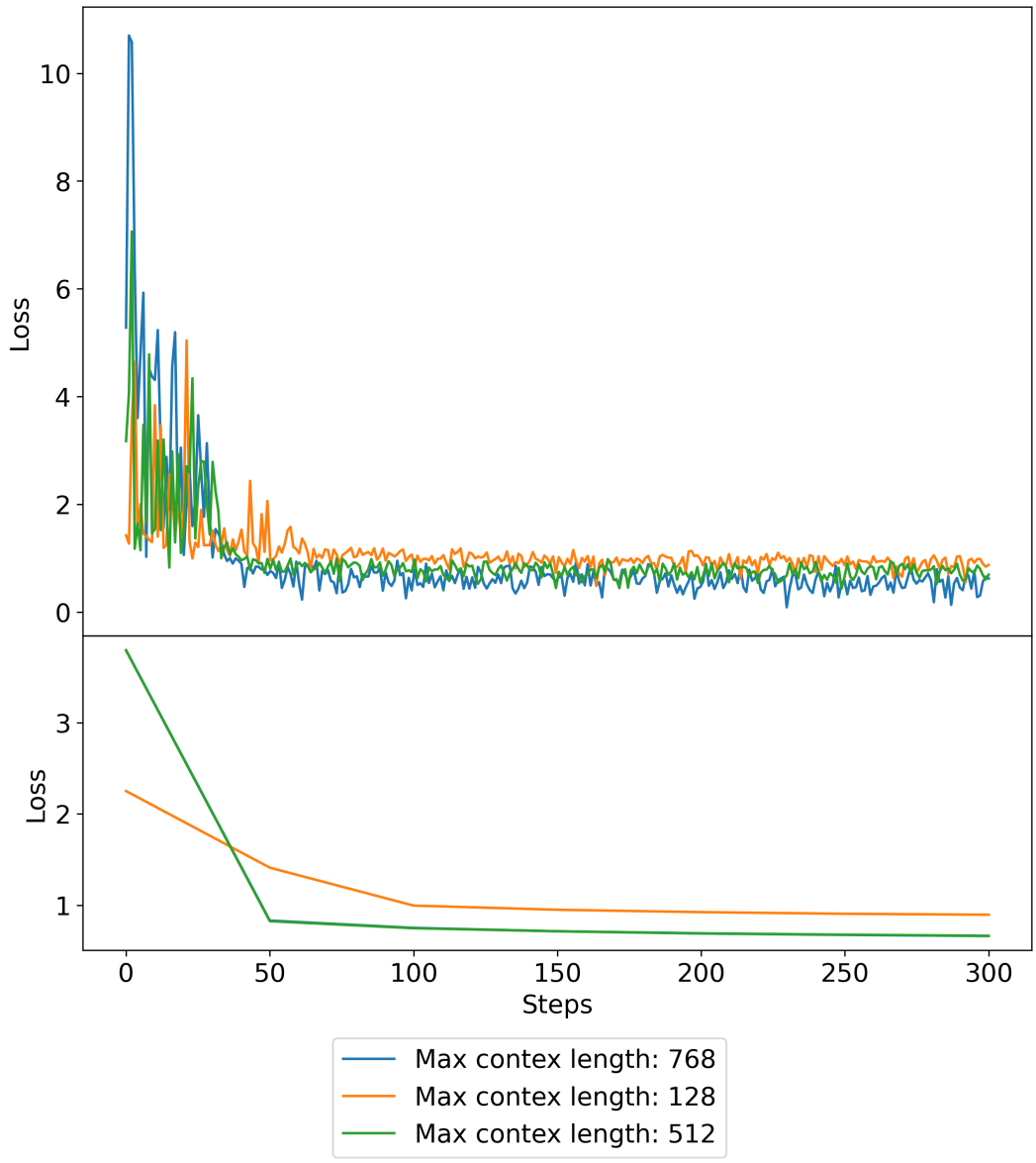

Figure 6: Training (top) and validation (bottom) loss for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data using different maximum context lengths with $\eta = 0.0001$ and $r = 8$.

5.3 Final model

Using the final hyperparameters, shown in Table 11, the Qwen2.5-0.5B-Instruct model was fine-tuned on the Lotka-Volterra time series data for 16,000 steps. The training and validation loss are shown in Fig. 7, and the model's performance on stratified examples from the test set is shown in Fig. **??**. The metrics for the model's performance on a selected stratified subset of test set are shown in Table 12. Additionally, the running MSE for the subset is shown in Fig. 9.

Hyperparameter	Value
Learning rate	10^{-4}
LoRA rank	8
Maximum context length	768
Batch size	4
LoRA scaling factor α	8

Table 11: Final best hyperparameters for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra time series data.

As can be seen in Fig. 7, the validation loss has not yet plateaued by 16,000 steps, indicating that the model was still learning from the data when training was stopped. However, with deference to the computational constraint of this coursework, the model was not trained for longer.

Comparing Figures 1 and 8, the model's performance on the test set is better than that of the untrained model or the LoRA adapted model trained with the default hyperparameters. This is especially shown by the growing and decaying systems 506 and 20, where the model is now able to predict at least the rough shape of the curve, though not the exact values. The oscillatory system 906 is also predicted much better than before, with the model now able to predict the oscillations fairly well. The oscillatory system 906 is also predicted fairly

	506		20		654		906	
Metric	Prey	Predator	Prey	Predator	Prey	Predator	Prey	Predator
MSE	0.008	0.002	2.676	1.249	0.000	0.000	5.296	0.480
RMSE	0.087	0.048	1.636	1.118	0.017	0.001	2.301	0.693
MAE	0.068	0.042	1.031	0.671	0.016	0.001	1.303	0.480
MAPE	0.051	0.037	0.455	0.142	0.002	0.001	0.265	1.456

Table 12: Metrics for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA using the final best hyperparameters on a subset from the test set. The metrics are calculated for the 20 points comprising the out-of-distribution data only.

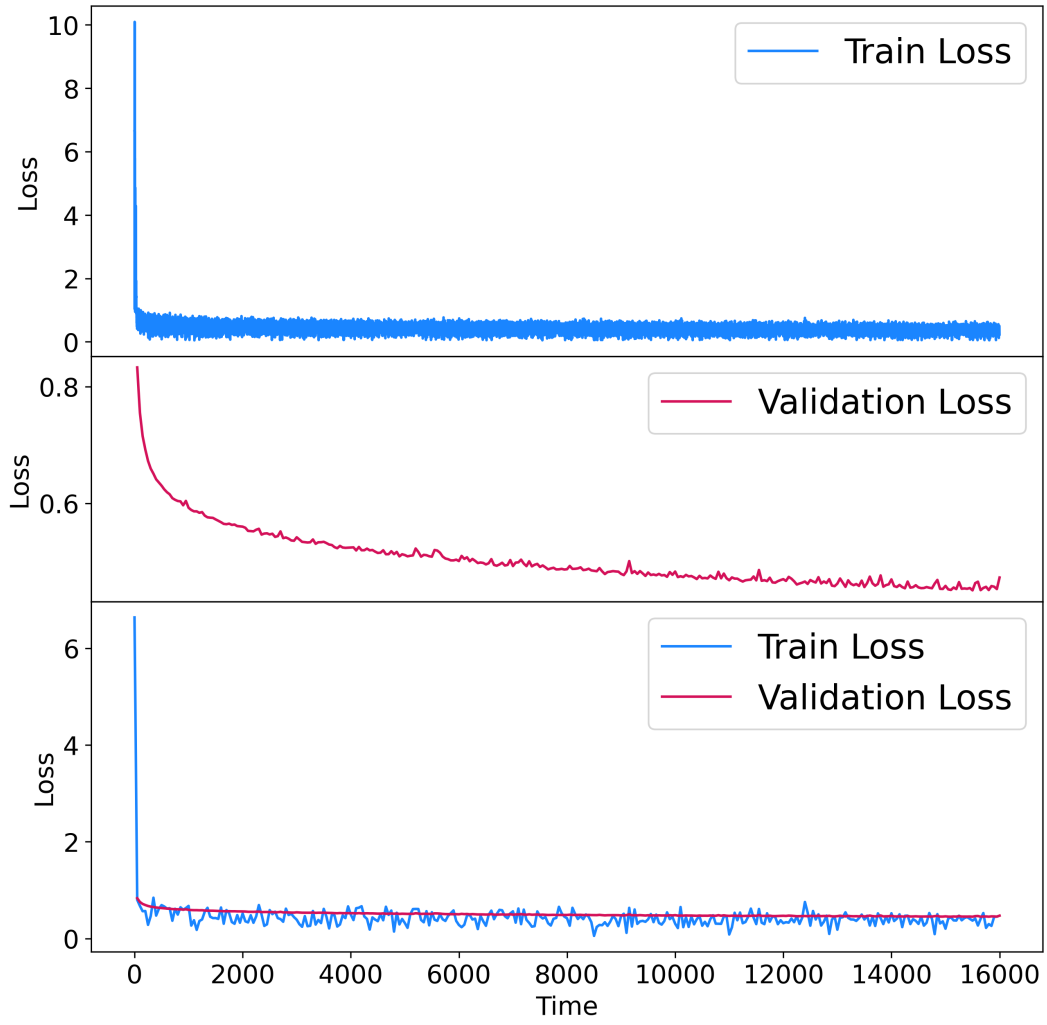

Figure 7: Training (top) and validation (middle) loss for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA on the Lotka-Volterra dataset using the final best hyperparameters. The bottom figure directly compares the training (blue) and validation (red) loss using the same resolution of every 50 steps.

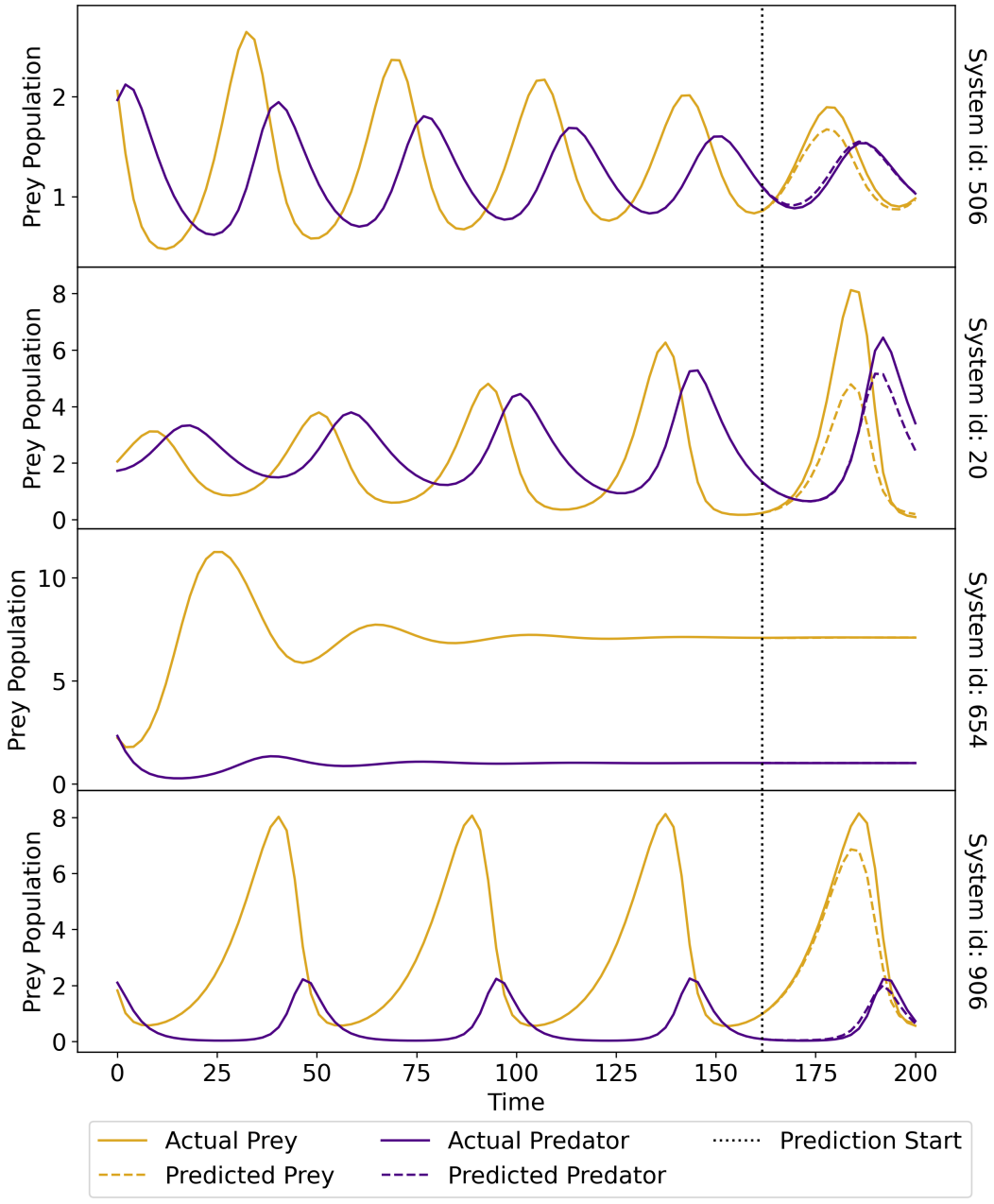

Figure 8: Predictions of the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA using the final best hyperparameters on a subset of the test set for both predator (purple) and prey (gold). The model's predictions are shown in the dashed lines, while the true values are shown in the solid lines.

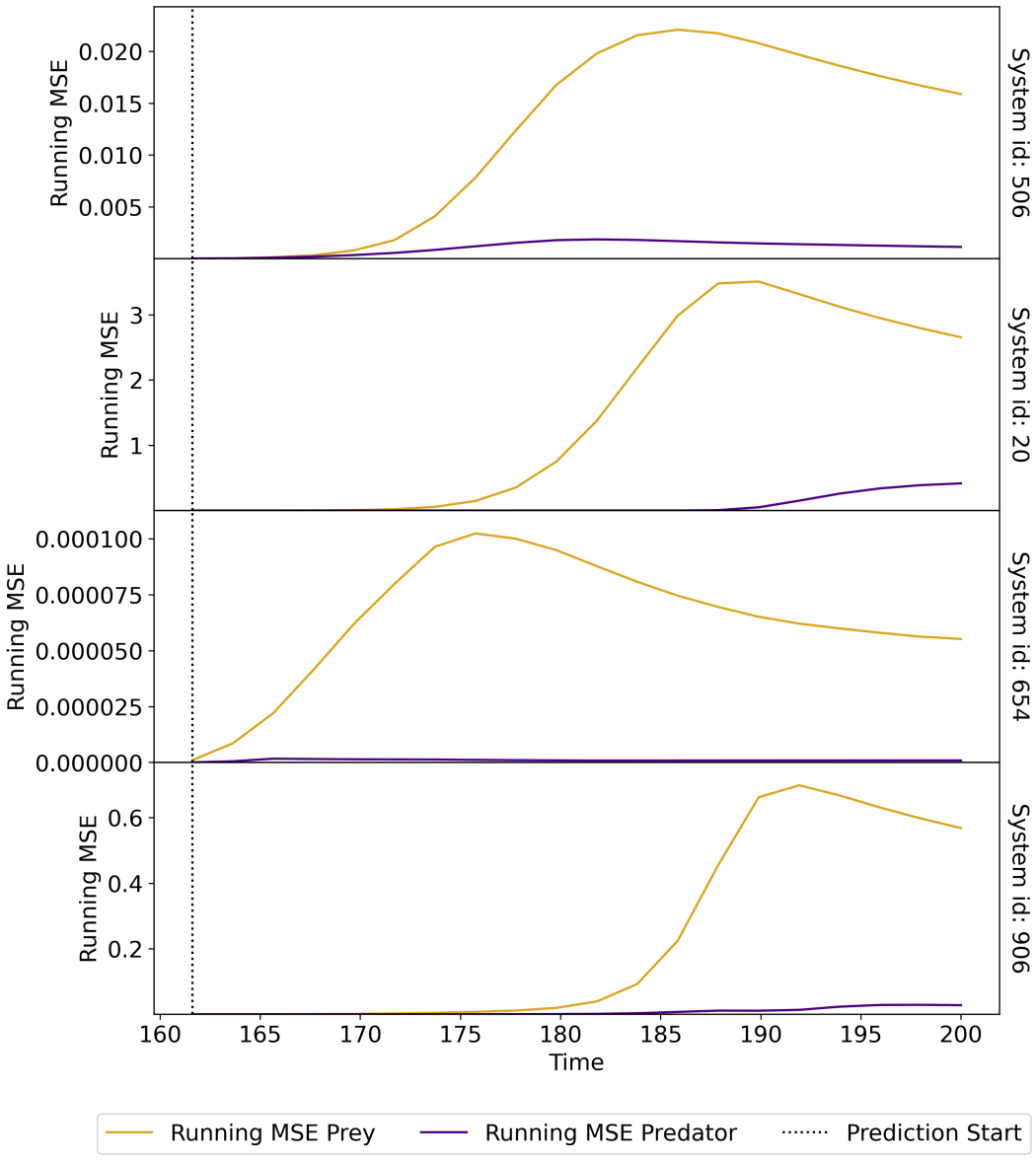

Figure 9: Running mean squared error (RMSE) for the Qwen2.5-0.5B-Instruct model fine-tuned with LoRA using the final best hyperparameters on a subset of the test set over the out-of-distribution data for the predator (purple) and prey (gold) time series.

well, with the model able to predict the overall shape of the curve, though not as well as the untrained model. And the stable system 654 is predicted almost perfectly, as both the prey and predator time series have settled into constant values before the cut-point and so the model is able to predict the next 20 points with very small error.

This is also reflected in the metrics shown in Table 12, where the MSE, RMSE, MAE, and MAPE for the predator and prey time series for all the systems except system 906 have decreased significantly compared to the metrics from Tables 5 and 7. In contrast, the errors recorded for system 906 increased significantly once the model was fine-tuned with LoRA, as seen in Table 7. This was improved with hyperparameter tuning and a longer training time, as the model learns to predict Lotka-Volterra time series data.

6 FLOPS usage

The FLOPS used for every experiment are shown in Table 13. As can be seen, the coursework is well within the stated budget of 1×10^{17} FLOPS.

	FLOPS			
Stage	Training	Validation	Inference	Total
Untrained	–	–	5.416e+12	5.416e+12
Default hyperparameters	1.214e+16	1.618e+14	5.046e+12	1.230e+16
η & r tuning	8.191e+15	1.092e+14	–	8.300e+15
Context length tuning	2.528e+15	3.370e+13	–	2.561e+15
Final run	7.456e+16	9.942e+14	5.047e+12	7.556e+16
Total	9.742e+16	1.299e+15	1.551e+13	9.873e+16

Table 13: FLOPS usage for each stage of the coursework including inference by the untrained model; training and inference of the LoRA adapted model with default hyperparameters; grid search of η, LoRA rank, and context length; and the training and inference of the final model.

When fine-tuning models for time-series forecasting under tight budgets, it is recommended to limit the length of training steps the model is allowed to complete. This is because as seen in Sections 5.2.1 and 5.2.2, it became clear fairly quickly which hyperparameters were best, sometimes being obvious at even just 50 steps. For this coursework, the hyperparameter search runs were made to train for 300 steps to make sure that the initial best hyperparameter stayed as such.

For this coursework, a maximum context length of 512 could have been chosen over that of 768 if the FLOPS budget was becoming a serious concern. This is because as seen in Section 5.2.2, it performed fairly similarly whilst also commanding a lower FLOPS cost.

However, as the FLOPS budget was well-managed, the slightly better performing context length was chosen despite its increased cost.

Another recommendation is to limit the amount of times the validation loss calculation is performed whilst training. This is because it not only takes up more of the FLOPS budget, but also more of the compute time taken to complete the training run. During this coursework, the validation was computed every 50 steps whilst training. This was done so that while it would limit how much the validation loss was calculated and thus save FLOPS as well as time, it would also provide enough information to see how the validation loss changed over time.

7 Further improvements

As seen in Fig. 8 in Section 5.3, the final model still does not predict Lotka-Volterra time series data to a high degree of accuracy for all types of systems. This may be improved by simply increasing the compute by allowing it to train for longer, but there are other optimisations also available.

One way to do this is by considering α as a tunable hyperparameter and allowing that to vary. According to Hu et al., it is recommended that α should be around twice the LoRA rank. Throughout this coursework, α was fixed at the LoRA rank, limiting the impact the LoRA weights had on the model's output. This of course impacts how quickly and how much the model learns from the training data, affecting its performance.

Another way to further improve the performance of the Qwen2.5-0.5B-Instruct model is to include a scheduler. This is because of the problem mentioned in Section 5.2.1, where a large learning rate showed larger dividends compared to the smaller learning rates. However, with this larger learning rate, it may well be that it only shows such a great benefit in only the initial few training steps of the model and in the later stages, a smaller learning rate would be more beneficial. This is especially obvious in Figures 3 and 7, where the training and validation loss curves begin to plateau at around 1,000 steps. This is where a scheduler comes in, where η of the optimizer is allowed to change over the course of the training run. Some tuning would be required to find the optimal scheduler and parameters for that scheduler, again impacting the FLOPS budget, but that would further improve the performance of the model.

8 Summary

The Qwen2.5-0.5B-Instruct LLM was fine-tuned using LoRA matrices and the optimised hyperparameters η, r, and maximum context length. This improved the performance of the model on predicting Lotka-Volterra time series data especially from its default state, but there are several improvements yet to be made.

LLMs may be somewhat competent at time series forecasting, but for more complex systems, it takes a large training dataset and lots of fine-tuning to reach acceptable levels of accuracy.

References

[1] Gruver N, Finzi M, Qiu S, Wilson AG, Large Language Models Are Zero-Shot Time Series Forecasters; 2024. https://arxiv.org/abs/2310.07820.

[2] Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, et al., LoRA: Low-Rank Adaptation of Large Language Models; 2021. https://arxiv.org/abs/2106.09685.

[3] Yang A, Yang B, Hui B, Zheng B, Yu B, Zhou C, et al., Qwen2 Technical Report; 2024. https://arxiv.org/abs/2407.10671.

A Use of auto-generation tools

Auto-generation tools were used to help parse error messages throughout the project, and to help format this LaTeX report.

Auto-generation tools were not used elsewhere, for code generation, writing, or otherwise.