M2 Deep Learning - Coursework
LoRA with Qwen

Zihan Xu (zx282)

Lent Term 2025

Word Count: 2979

1 Baseline

1.1 (a)

To transform time series data into tokens executable by Large Language Models (LLMs), the LLMTIME preprocessing scheme is implemented. In this approach, a dedicated class is developed, `LLMTIMEPreprocessor`, to manage both the encoding and decoding processes of the time series data.

To standardize the data across all trajectories, apply a global scaling factor that maps all data points to the range $[0, 10]$ is applied. This factor, denoted by α, is computed by first identifying the maximum value in the entire dataset, $x_{\max}$, and then using the formula

$$\alpha = \frac{10}{x_{\max}}$$

For the dataset, α was determined to be approximately 1.3740113258361817. This value is stored in `results/scaling_factor.json`.

Two examples are generated to determine the functionality of the preprocessor:
Example 1:

$$\begin{aligned}
\text{original:} \quad & [[1, 2], [2, 4], [3, 6]] \\
\text{preprocessed:} \quad & \text{``1.67, 3.33; 3.33, 6.67; 5.00, 10.00''} \\
\text{tokenized:} \quad & [16, 13, 21, 22, 11, 18, 13, 18, 18, 26, \\
& \quad 18, 13, 18, 18, 11, 21, 13, 21, 22, 26, \\
& \quad 20, 13, 15, 15, 11, 16, 15, 13, 15, 15]
\end{aligned}$$

Example 2:

$$\begin{aligned}
\text{original:} \quad & [2, 3, 5] \\
\text{preprocessed:} \quad & \text{``4.00; 6.00; 10.00''} \\
\text{tokenized:} \quad & [19, 13, 15, 15, 26, 21, 13, 15, 15, 26, 16, 15, 13, 15, 15]
\end{aligned}$$

Two examples are deliberately chosen with different dimensions to assess and test the functionality of the preprocessor.

1.2 (b)

The forecasting ability of the untrained Qwen2.5-Instruct model is evaluated on a dataset of 200 samples, chosen to balance computational cost and metric reliability. Each sample consists of 100 predator-prey time series pairs, with the first 80 data points used as context. The model is tasked with predicting the final 20 data points for both the predator and prey.

Given that the data represent time series, three metrics are chosen to assess the performance of the forecasting model: mean squared error (MSE), mean absolute error (MAE), and the R^2 score. MSE is chosen for its heavy weight on larger errors due to its squaring operation, making it particularly sensitive to outliers. This property is valuable when assessing a model's ability to minimize significant prediction errors.

MAE is selected because it measures the average magnitude of the errors for predictions. It serves as a complementary metric for the MSE as MAE is able to produce insight into the average magnitude of error in the prediction.

The R^2 score measures how well the model's predictions fit the actual data. This metric is particularly useful for understanding the overall predictive power of the model and its ability to capture the underlying trends in the data.

The metrics are calculated using the script `src/untrained_Qwen.py` and further processed using `src/calculate_Qwen_result.py`. Below are the untrained Qwen2.5-Instruct model's performance assessed using the 3 metrics corrected to 3 significant figures:

$$\text{MSE} : 0.845 \quad \text{MAE} : 0.401 \quad R^2 : -0.483$$

The MSE value reflects that there are notable prediction errors, especially larger deviations, but the model is not performing completely poorly. The value of MAE is relatively lower compared to the MSE, suggesting that the errors are fairly moderate on average. This showcases the poor forcasting ability for untrained Qwen, as the model is consistently achieving poor predictions.

The negative R^2 score of -0.483 indicates that the model's predictions are worse than simply predicting the mean of the observed data. An R^2 score less than 0 suggests that the model fails to capture the underlying trends in the data and does not provide useful predictions, further exposing the poor performance of untrained Qwen.

To better visualize the results generated by untrained Qwen, the prediction for the first trajectory of plotted with the ground truth:

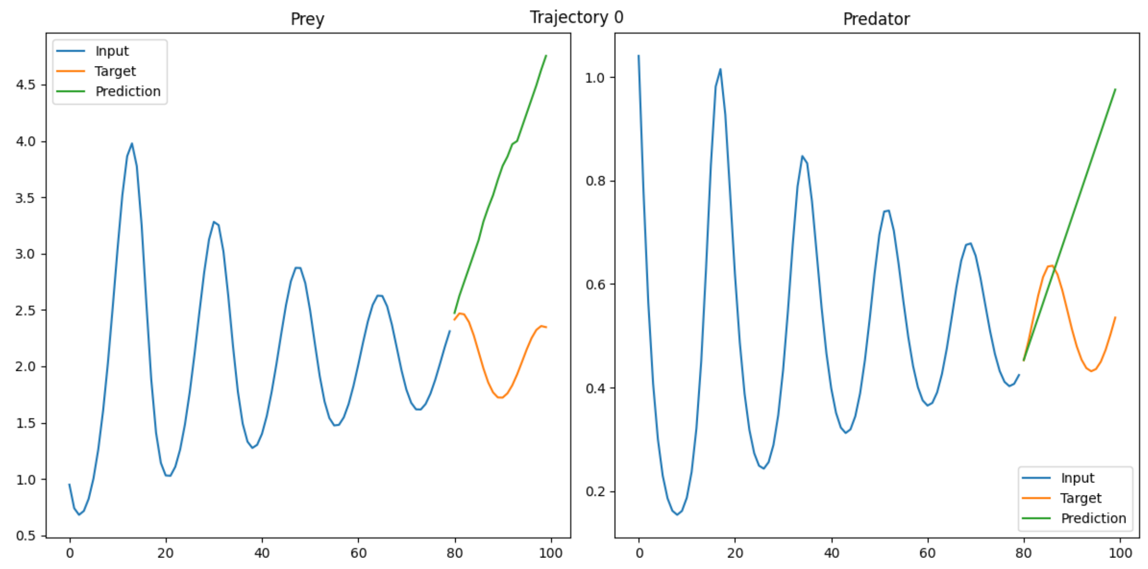

Figure 1: Predator-Prey Trajectory Prediction From Untrained Qwen

It can be observed that, rather than capturing the oscillatory patterns present in the ground truth data, the predictions made by the Qwen model exhibit a simple linear trend. This deviation from the expected behavior is a significant factor for the model's poor performance across the evaluation metrics.

1.3 (c)

To calculate the FLOPs for the forward pass of Qwen2.5-Instruct, the structure of the model must be known. This information is obtained by printing out the model after loading it:

`Qwen2ForCausalLM` consists of the following components:

- **Embedding Layer:**

$$\text{embed\_tokens} : \text{Embedding}(151936, 896)$$

- **Layers:** 24 `Qwen2DecoderLayer` modules
- **Normalization:**

$$\text{norm} : \text{Qwen2RMSNorm}(896, \epsilon = 1e-6)$$

- **Rotary Embedding:** `Qwen2RotaryEmbedding`
- **Linear Layer for Language Modeling:**

$$\text{lm\_head} : \text{Linear}(896, 151936)$$

Each `Qwen2DecoderLayer` consists of the following components:

- **Self-Attention Layer:**

$$\text{self\_attn} : \texttt{Qwen2Attention}$$

  - Query Projection:
  $$\text{q\_proj} : \text{Linear}(896, 896)$$
  - Key Projection:
  $$\text{k\_proj} : \text{Linear}(896, 128)$$
  - Value Projection:
  $$\text{v\_proj} : \text{Linear}(896, 128)$$
  - Output Projection:
  $$\text{o\_proj} : \text{Linear}(896, 896)$$

- **MLP (Multi-Layer Perceptron):**

$$\text{mlp} : \texttt{Qwen2MLP}$$

- Gate Projection:

$$\text{gate\_proj} : \text{Linear}(896, 4864)$$

- Up Projection:

$$\text{up\_proj} : \text{Linear}(896, 4864)$$

- Down Projection:

$$\text{down\_proj} : \text{Linear}(4864, 896)$$

- Activation Function: `SiLU()`

- **Normalization Layers:**
  - Input Layer Normalization:

$$\text{input\_layernorm} : \texttt{Qwen2RMSNorm}(896, \epsilon = 1e - 6)$$

  - Post-Attention Layer Normalization:

$$\text{post\_attention\_layernorm} : \texttt{Qwen2RMSNorm}(896, \epsilon = 1e - 6)$$

The FLOPs should be calculated for each component individually, then combined. Let N represent the number of input tokens. The embedding layer maps each input token to an embedding vector of size 896, producing a matrix of size $N \times 896$. Since the lookup operation is not defined in the FLOPs table, it is assumed to incur 0 FLOPs.

The primary computational component of Qwen is the `Qwen2DecoderLayer` module. The first part of the FLOP calculation focuses on the self-attention mechanism. The Key and Value projections each correspond to matrices of size 896×128, while the Query projection corresponds to a matrix of size 896×896. Therefore, mapping the embedding matrix to the Query vector results in $N \cdot 896 \cdot (2 \cdot 896 - 1)$ FLOPs. Similarly, the Value and Key vector projections yield $N \cdot 128 \cdot (2 \cdot 896 - 1)$ FLOPs.

With 7 attention heads in this architecture, given by $896/128 = 7$, we obtain $N \cdot 7 \cdot (N \cdot 128)$ FLOPs for the multiplication between the Query and the transpose of Key matrices. The scaling and softmax operations contribute $7N^2$ and $7N(N + 10)$ FLOPs, respectively, where the extra +10 in the softmax operation accounts for the exponentiation step.

After obtaining the attention score, it is multiplied with the value matrix, giving a FLOP count of $N7N128$. Finally, the output projection gives an output for further operations, causing a FLOP count of $N \cdot 896 \cdot (2 \cdot 896 - 1)$.

The output of the self-attention layer is inputted into the post-attention layer for normalization. The formula for `RMSNorm` is the following:

$$y_i = \frac{x_i}{RMS(x)} * \gamma_i \qquad RMS(x) = \sqrt{\epsilon + \frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

This operation is applied to the output of the attention layer, which has shape $N \times 896$. This incur a cost of $4 \cdot N \cdot 896 + 10$ FLOPS, which is the same as the cost for the input layer norm.

The Multi-Layer Perceptron consist of 3 projection matrices followed by the SiLU activation function. The gate projection and the up projection both transform the embedding vectors from size 896 to 4864. Then the result from gate projection is passed through the SiLU activation function and then "gate" the matrix from the up projection through element-wise multiplication. The up scaling projection produces FLOPS count $N \cdot 4864 \cdot (2 \cdot 896 - 1)$, same as the gate projection. The SiLU activation function takes the following form:

$$SiLU(x) = x\sigma(x) \qquad \sigma(x) = \frac{1}{1 + \exp(-x)}$$

The SiLU activation function requires 14 FLOPs per individual element, resulting in a total of $N \times 4864 \times 14$ FLOPs. The element-wise multiplication of the gate matrix with the up-scaled matrix incurs $N \times 4864$ FLOPs. The down projection then projects the gated matrix back to its original size, with a cost of $N \cdot 896 \cdot (4864 - 1)$ FLOPs.

Summing all these components gives the total FLOPs required for one pass through the `Qwen2DecoderLayer`. Multiplying this value by 24 provides the total FLOPs for processing the entire decoder layer.

As previously calculated, the final normalization layer incurs a cost of $4 \cdot N \cdot 896 + 10$ FLOPs.

The rotary embedding layer follows the normalization layer, dealing with positional embeddings. These embeddings are then added to the token embeddings, which contributes $N \cdot 896$ FLOPs.

The final linear layer maps the embeddings to the vocabulary through a matrix multiplication between an $N \times 896$ matrix and a $896n \times 151936$ matrix. This results in an additional $N \cdot 151936 \cdot (2 \cdot 896 - 1)$ FLOPs.

For the Qwen2.5-Instruct model, which has been set to have a maximum context length of 256, the total FLOPs is calculated to be $768, 361, 780, 128$ FLOPs.

2 LoRA

2.1 (a)

Using the provided `lora_skeleton.py` code, LoRA is integrated into the Qwen2.5-Instruct model.

To ensure that only the LoRA layers are trained, all parameters of the original Qwen model are frozen by setting their `requires_grad` flag to `False`. In contrast, the parameters in the LoRA wrapper remain trainable.

When defining the Adam optimizer, only parameters with `requires_grad = True` are passed in. This ensures that only the LoRA parameters are updated during training, leaving the base model unchanged.

The chosen LoRA configuration uses a rank of 4 and a learning rate of 10^{-5}. To address the limited size of the dataset (1,000 trajectories), K-fold cross-validation is employed as the training strategy. By using K-fold cross-validation, every data point is used for both training and validation at different stages. This ensures that the model has access to all data points during the training process, improving the quality of learning, when only small number of data is accessible. This approach helps to make more effective use of the available data and improves model generalization [1].

This strategy also mitigate the risk of overfitting, by ensuring that the model is not overly tuned to any one training set. The performance of the model is averaged across all K folds, helping to reduce bias originating from train-validation dataset split and provides a more reliable estimate of the overall performance of the model.

Here, K is set to 5. This number is chosen to balance between training efficiency and model generalization. The maximum number of steps allocated to each fold is hence set to $10000/5 = 2000$ and the number of data available in each fold is $1000/5 = 200$. The validation dataset is set with batch size 8, while the training dataset is set to have batch size 2. The training dataset has a smaller batch size due to the limited data available, the model should tune according to each data accordingly while maintaining a relatively quick learning speed. The validation dataset has a higher number in order to speed up the validation metrics calculation.

The validation metrics of the model is recorded:

$$\text{MSE} : 0.457 \quad \text{MAE} : 0.279 \quad R^2 : 0.686$$

This result is a significant improvement upon the untrained Qwen result, reflected by the value of the MSE. The Mean Absolute Error (MAE) is 0.2794, which provides another view of the model's accuracy. The predictions deviate from the actual values by around 0.28, while the scale of data is between 0 to 10. This relatively low value supports the notion that the model is making reasonably accurate predictions. Lastly, the R^2 value indicates that the model captures a substantial portion of the underlying pattern in the data.

However, hyperparameter tuning is still needed to increase the prediction accuracy for the LoRA integration of the Qwen model.

To visualize the prediction, the ground truth and the prediction of one data trajectory (the first one) is plotted:

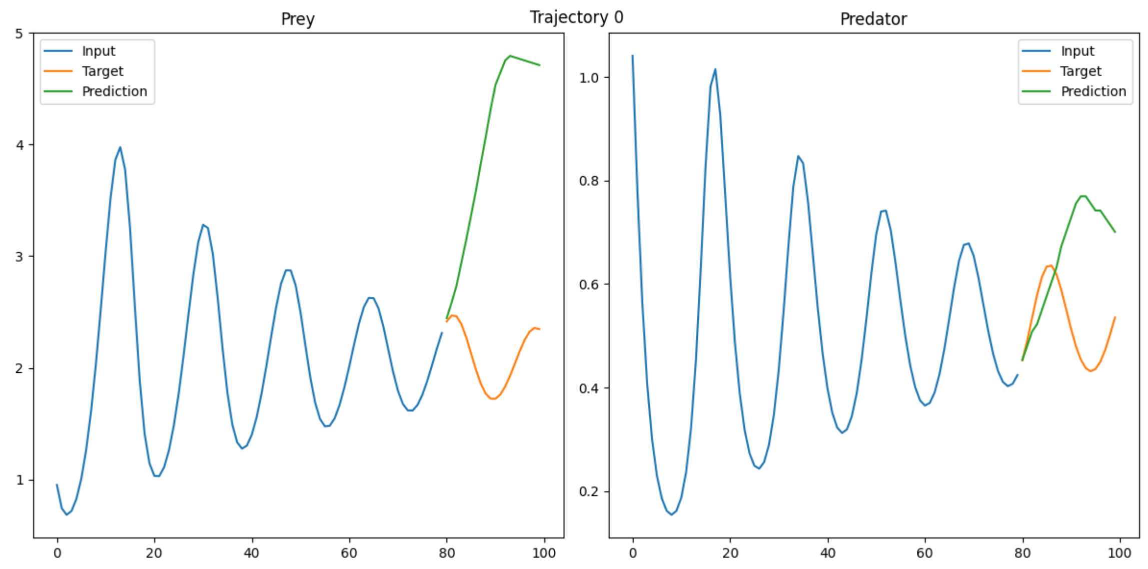

Figure 2: LoRA Rank 4 Learning rate 10^{-5} Prediction for Trajectory 0

Compared with the result from untrained Qwen, where the predicted trajectories are linear, the Lora adapted model exhibits oscillatory behavior in both the Prey and Predator trajectories. This suggests that the LoRA method is starting to capture the cyclical nature of the dynamics. The predicted trajectories align more closely with the true values, even though there is still some lag and deviation. Overall, the LoRA-adapted model offers a substantial improvement over the untrained Qwen, but further training and adjustments could lead to more accurate predictions.

2.2 (b)

To improve the performance of the LoRA-adapted model, a hyperparameter search is conducted over two key parameters: the learning rate and the LoRA rank. The learning rates are selected from the set $10^{-5}, 5 \times 10^{-5}, 10^{-4}$, and the LoRA ranks are chosen from the set $2, 4, 6$. Given that the total number of parameter combinations is relatively small ($3 \times 3 = 9$), a grid search is used to evaluate all possible combinations exhaustively. This approach is straightforward and can produce globally optimal result, given the search space is manageable.

Although the K-fold cross-validation strategy would provide a more reliable estimate of model performance by reducing variance, it increases the computational cost. Consequently, for the hyperparameter search phase, a simpler train-validation-test split is utilized to speed up the process without significantly compromising the quality of the results. The data is split in the following ratio: 60% for training, 20% for validation, and 20% for testing. This split ensures that the model has enough data to train effectively while maintaining a separate validation set to fine-tune the hyperparameters. The test dataset is then used to compare the performance of different model to select the best combination. The chosen ratio provides a balance between training the model and evaluating its generalization performance.

The maximum number of training steps allowed for each model is set to 4000. Then the validation dataset is passed into the models. The maximum context length is fixed to 256. Finally, the test metrics are logged for each model:

Learning Rate	**LoRA Rank 2**	**LoRA Rank 4**	**LoRA Rank 8**
10^{-5}	0.2309	0.2131	0.0914
5×10^{-5}	0.1943	0.1261	0.1375
10^{-4}	0.1192	0.1299	0.0392

Table 1: Test MSE Values for Different LoRA Ranks and Learning Rates

Learning Rate	**LoRA Rank 2**	**LoRA Rank 4**	**LoRA Rank 8**
10^{-5}	0.1702	0.1538	0.0992
5×10^{-5}	0.1371	0.1093	0.1230
10^{-4}	0.1136	0.1133	0.0660

Table 2: Test MAE Values for Different LoRA Ranks and Learning Rates

Learning Rate	**LoRA Rank 2**	**LoRA Rank 4**	**LoRA Rank 8**
10^{-5}	0.8367	0.8492	0.9384
5×10^{-5}	0.8679	0.9117	0.9056
10^{-4}	0.9199	0.9078	0.9744

Table 3: Test R^2 Values for Different LoRA Ranks and Learning Rates

The test metrics for MSE, MAE, and R^2 reveal that higher learning rates generally improve model performance across all LoRA ranks. Specifically, for LoRA Rank 2, increasing the learning rate leads to lower MSE and MAE values, and a higher R^2, indicating better model fit. LoRA Rank 4 also shows improvement with higher learning rates, but the performance gains are less pronounced compared to Rank 2. LoRA Rank 8 consistently outperforms the other ranks, achieving the lowest MSE and MAE values, as well as the highest R^2, particularly at the highest learning rate of 0.0001. Overall, the results suggest that increasing the learning rate enhances the model's predictive accuracy, with LoRA Rank 8 benefiting the most from this configuration.

Hence, the best performing pair of hyperparameter is concluded to be rank 8, learning rate 1×10^{-4}.

To visualize the predictive power of this hyperparameter combination after training, the prediction for trajectory 0 against the ground truth is plotted:

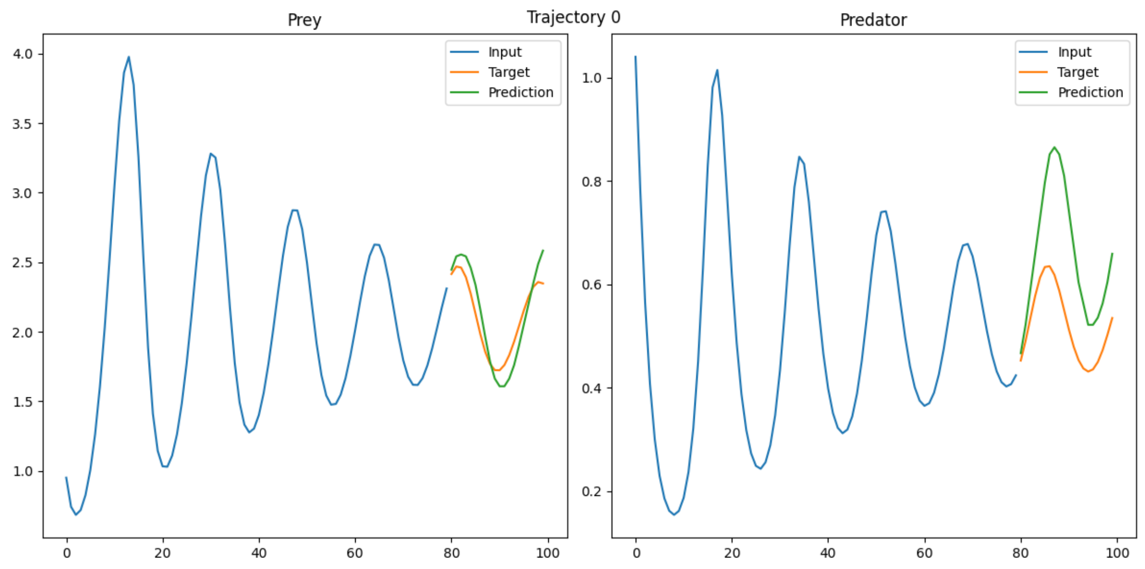

Figure 3: Prediction Visualization for LoRA rank 8 Learning Rate 10^{-4}

This plot shows that the model prediction aligns closely with the target, indicating the model's ability to capture the oscillatory nature of the predator-prey system. The alignment in terms of both amplitude and phase is effective for the prey's behavior. In contrast, the predator dynamics plot shows some divergence between the predicted and target values, with the model struggling to match the amplitude of the target. This suggests that while the model captures some of the predator's oscillatory behavior, more training or tuning is needed to improve precision.

This combination is then used to search for the optimal context length in $(256, 512, 768)$ with the same experiment setting as mentioned before. The results are the following:

Context Length	**MSE**	**MAE**	**R^2**
128	0.1745	0.1366	0.8767
512	0.1734	0.1407	0.8775
768	0.1635	0.1333	0.8829

Table 4: Evaluation metrics for context lengths: LoRA rank 8 learning rate 10^{-4}

It can be observed that the model's performance improves as the context length increases. Among the tested configurations, the shortest context length (128) yields the highest MSE and MAE, along with the lowest R^2, indicating weaker predictive accuracy. In contrast, a context length of 768 results in the best performance across all three metrics, suggesting that providing the model with more contextual information enhances its ability to make accurate predictions. However, the metrics here are worse compared to the results obtained in previous experiments, where the context length was capped at 256. This indicates that further experimentation is necessary to draw more definitive conclusions regarding the relationship between context length and model prediction accuracy.

The number of FLOPs is tracked. The LoRA linear layer contributes to additional FLOPs by introducing matrices A and B into each `Qwen2DecoderLayer` layer. The additional FLOP count can be calculated and summed to the original Qwen FLOP count to produce the final number. Let the LoRA rank of the model be r and the context length be N. The dimension of A is $r \times 896$ and that of B is $896 \times r$, if it is wrapping the query projection, $128 \times r$, if it is wrapping the value projection.

Hence the additional FLOPs are, from LoRA linear layer on Query projection, $N \cdot r \cdot (2 \cdot 896 - 1) + N \cdot 896 * (2 * r - 1)$, from the LoRA linear layer on Value projection, $N \cdot r \cdot (2 \cdot 896 - 1) + +N \cdot 128 * (2 * r - 1)$. This number multiplied by 24 would give the total LoRA contribution. Below are the results:

Rank	**Cost**
2	5,126,185,266,560,000
4	5,130,336,152,960,000
8	5,138,637,925,760,000

Table 5: Cost by Rank

Above are the results with maximum context length 256. Since learning rate has no effect to the FLOPs count, this information is omitted from the table.

Below are the FLOPs count for the maximum length selection with LoRA rank 8:

Configuration	**Cost**
Context Length 128	255,511,600,576,000
Context Length 512	1,039,089,953,728,000
Context Length 768	1,575,678,484,416,000

Table 6: Cost by Context Length and Final Training

These figures account for both training and inference costs. The total FLOPs incurred during hyperparameter selection slightly exceed the budget, but remain within the same order of magnitude.

2.3 (c)

Based on the results of the previous experiments, the following hyperparameters are selected for additoonal training: a LoRA rank of 8, a learning rate of 10^{-4}, and a maximum context length of 256 tokens.

To ensure robust model evaluation and mitigate the risk of overfitting, the K-fold cross-validation training strategy is employed. This approach allows improved generalization and more reliable performance metrics. The maximum number of optimizer steps is set to 30,000, which allows sufficient training for the model to converge under the selected configuration.

Below are the validation metrics:

$$\text{MSE} : 0.1240 \quad \text{MAE} : 0.1140 \quad R^2 : 0.9170$$

These metrics suggest that the model is performing well. The low error values and a high R^2 indicates a good fit between the predicted and actual values.

However, these results show that the model is underperforming compared to the results in section (b). To investigate this unlikely behavior, the performance metrics for each fold is examined separately:

Fold	MSE	MAE	R^2
1	0.1822	0.1440	0.8806
2	0.0714	0.1004	0.9491
3	0.1692	0.1215	0.8842
4	0.0983	0.0989	0.9309
5	0.0990	0.1051	0.9401

Table 7: K-Fold Cross-Validation Performance Metrics

The table shows significant variation in model performance across folds. Fold 2 achieves the best results, while Fold 1 and Fold 3 show poorer performance, with higher MSE, MAE, and lower R^2 values. This suggests the model may be sensitive to data selection, potentially due to issues with data distribution or generalization. Further analysis is needed to explore these factors and improve model stability. This also highlights the benefits of K-fold cross-validation, which helps minimize the impact of dataset splitting and provides more reliable metrics.

Again, the prediction against the ground truth for trajectory 0 is visualized:

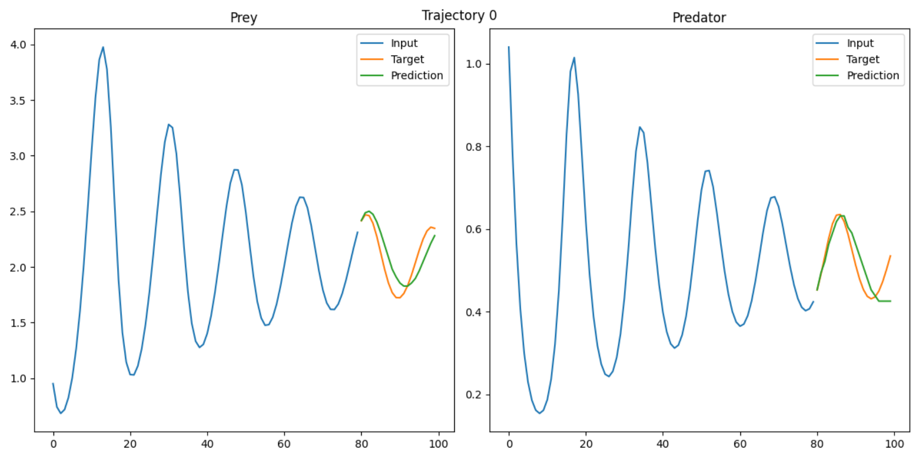

Figure 4: Prediction Visualization: LoRA rank 8, Learning Rate 10^{-4}, context length 256

Unlike the previous trajectory plots, the model prediction here closely aligns with the target. After training for 30,000 steps, it successfully captures the amplitude and phase of both trajectories, demonstrating the effectiveness of the LoRA adaptation for the Qwen2.5-Instruct model in predicting the predator-prey system.

Here, the FLOPs count for the final training is calculated: 7,707,956,888,640,000, which is under-budget.

To conclude, it is recommended to use LoRA rank 8, learning rate 10^{-4} and maximum context length 256 to predict the predator prey times series. Possible next steps in improving the performance involves investigating more models, searching in a wider region of hyperparameter, gathering more data and pretraining Qwen model.

3 Acknowledgment

This report was developed with the assistance from Anthropic's Claude 3.5 Sonnet. Specifically, Claude was used to improve the clarity of written content and assist with debugging Python code implementations. All AI-generated suggestions were manually reviewed.

References

[1] DataCamp. ”K-Fold Cross Validation.” DataCamp, www.datacamp.com/tutorial/k-fold-cross-validation. Accessed 2 Apr. 2025.