M2 coursework report; Word Count:

Yihao Liu; CRSid:yl2063

April 2025

Contents

1 Introduction

Large Language Models (LLMs) have shown potential beyond natural language processing, notably in time series forecasting. In particular, untrained LLMs can forecast time series when suitably tokenized [1]. This project investigates the application of LoRA (Low-Rank Adaptation) [2] to fine-tune Qwen2.5-Instruct [3] for forecasting dynamics governed by the Lotka-Volterra predator-prey equations. Throughout all experiments, we used the 0.5B parameter variant of Qwen2.5-Instruct [3].

2 Compute Constraint

This coursework imposes a strict compute budget of 10^{17} floating point operations (FLOPs) across all reported experiments.

Only operations listed in Table 1 are counted; others (e.g., tokenization, memory access) are excluded. FLOPs are calculated for the forward pass, with backpropagation assumed to cost exactly twice as much. In addition, only training a model is counted towards the FLOPs budget.

Table 1: FLOPS Accounting for Primitives

Operation	**FLOPs**
Addition / Subtraction / Negation (float or int)	1
Multiplication / Division / Inverse (float or int)	1
ReLU / Absolute Value	1
Exponentiation / Logarithm	10
Sine / Cosine / Square Root	10

3 Load and preprocessing the data

3.1 Dataset Structure

`lotka_volterra_data.h5` contains time series data simulating predator-prey population dynamics based on the Lotka-Volterra equations.

It consists of two components:

- `/trajectories`: A 3-dimensional array of shape $(1000, 100, 2)$, where:
  - The first dimension indexes 1000 independent predator-prey systems,
  - The second dimension corresponds to 100 time steps per system,
  - The third dimension contains two variables: prey population (`[:,:,0]`) and predator population (`[:,:,1]`).
- `/time`: A 1-dimensional array of length 100, representing the shared timeline across all systems.

3.2 Preprocessing the data

To make the multivariate numerical time series compatible with the Qwen2.5-Instruct tokenizer, we implemented a text-based preprocessing pipeline based on the LLMTIME scheme [1].

3.2.1 Scaling and Precision

Each numerical value x_t was rescaled to $x'_t = \frac{x_t}{\alpha}$, where α is a scaling constant selected to ensure that most values fall within a consistent numeric range ($[0.00, 9.99]$ in our implementation). The scaled values were then rounded to a fixed number of decimal places (two in this coursework), ensuring uniform token lengths and predictable tokenization.

3.2.2 Multivariate Encoding

We adopted a format that encodes multivariate time series as token-friendly strings. The encoding follows these rules:

- Prey and predator at the same timestep are separated by a comma ‘,‘,
- Different timesteps are separated by a semicolon ‘;‘.

3.2.3 Data splitting

The dataset was randomly partitioned into training, validation, and test sets using a 7:1.5:1.5 split ratio. The indices of the 1000 trajectories were first shuffled with a fixed random seed to ensure reproducibility.

- The **training set** was used for model training.
- The **validation set** serves to evaluate models' forecasting ability.
- The **test set** provides an unbiased evaluation of the final model's generalization ability.

3.2.4 Numeric Tokenization

Each string element is tokenized by the Qwen tokenizer. Inputs for both training and evaluation should be tokenized. Please note that our range $[0.00, 9.99]$ converted every value into exact three digits, so that the numbers of tokens generated from the string-form sequences are always consistent.

3.2.5 Examples

We examined the second sequence from the training set as an example with the first 40 characters.

The preprocessed sequence is:

```
0.92,0.74;0.56,0.77;0.34,0.70;0.23,0.59;
```

The tokenized sequence is:

```
[15, 13, 24, 17, 11, 15, 13, 22, 19, 26, 15, 13, 20, 21, 11, 15,
13, 22, 22, 26, ...]
```

In both cases, we confirmed that the total sequence length was 999, as the final semicolons were intentionally omitted.

4 Model Architecture, LoRA implementation and FLOPs Accounting

4.1 Model Architecture

The Qwen2.5-Instruct model is a decoder-only transformer architecture, which consists of 24 transformer blocks, each with a embedding dimension (hidden size) of 896 and 14 attention heads. The vocabulary size is 151,936. The model architecture proceeds as Figure 1. This is a simplified description of the model architecture with the primary goal of supporting our FLOPs estimation.

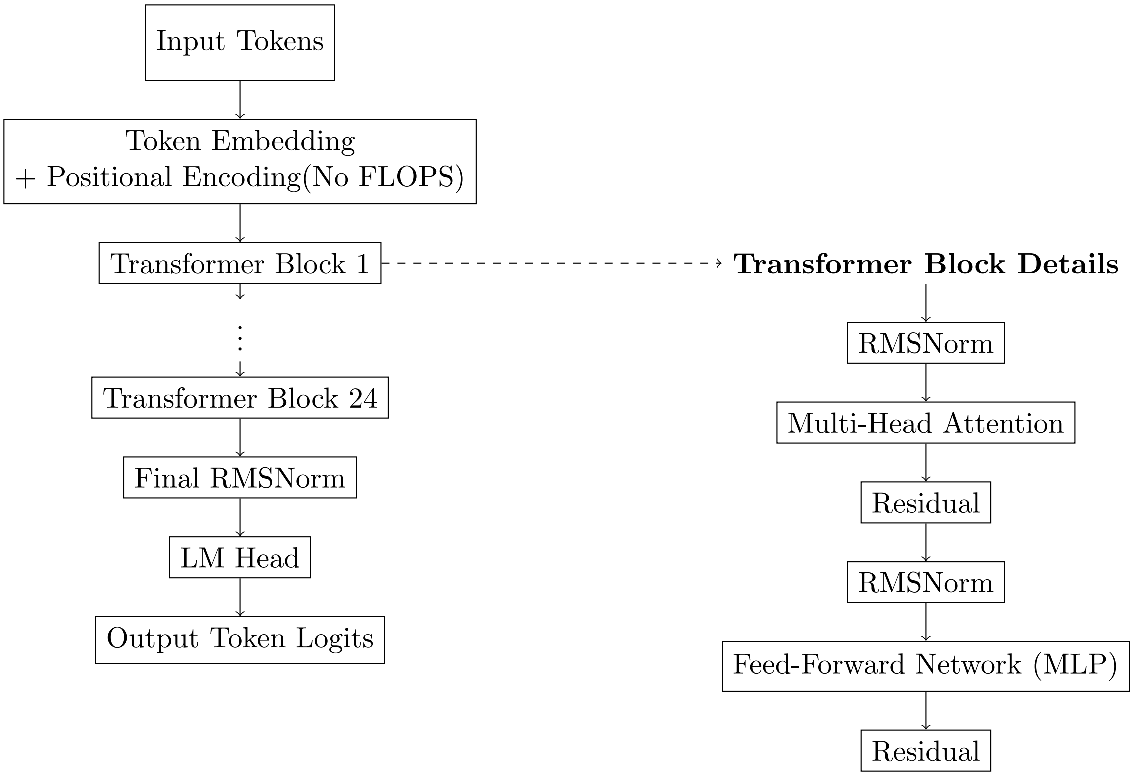

Figure 1: Architecture of the Qwen2.5 model and the structure of a single Transformer Block.

4.2 LoRA implementation

LoRA [2] is a fine-tuning technique that introduces trainable low-rank matrices into pre-trained transformer architectures. Instead of updating the full weight matrices of a large model, LoRA freezes the original weights and injects two smaller matrices $A \in R^{r \times d}$ and $B \in R^{d \times r}$ such that their product approximates the additional weight updates:

$$W_{\text{adapted}} = W_0 + \Delta W = W_0 + BA$$

where W_0 is the frozen pre-trained weight, and $r \ll d$ denotes the LoRA rank.

In this project, LoRA was applied to the `q_proj` and `v_proj` linear layers inside the self-attention modules of Qwen2.5-Instruct. Only the matrices A and B are trained.

4.3 FLOPs Accounting

The full details of the FLOPS accounting implemention can be found in the python sript `flops.py`. Here some notable examples are illustrated with the rule in Table 1.

Matrix multiplications are the dominant contributor to FLOPs in transformer models: multiplying a matrix of shape $(a \times b)$ with one of shape $(b \times c)$ requires $2abc - ac$ FLOPs, accounting for b multiplications and b - 1 additions.

RMSNorm is defined as:

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{d} \sum x_i^2 + \epsilon}} \cdot \gamma$$

The total cost of RMSNorm per token is thus approximately $4d + 11$, where d is the hidden size.

The **SwiGLU** activation function is defined as:

$$\text{SwiGLU}(x_1, x_2) = \text{SiLU}(x_1) \cdot x_2, \quad \text{with} \quad \text{SiLU}(x) = \frac{x}{1 + e^{-x}}$$

FLOPS account is 14 FLOPs per intermediate hidden unit.

While Qwen2.5 uses *Grouped Query Attention* (GQA), we simplify our accounting by assuming the same cost as standard **Multi-Head Attention**. Positional encodings are not included in the FLOPs count, but we do include the cost of *adding* them to the token embeddings.

5 Methodology of Model Evaluation

5.1 Context-Target Splitting

We evaluate the model in an autoregressive forecasting setting by splitting each tokenized input sequence into a context segment and a target prediction segment. The context ratio is set to be 0.7, so that 700 tokens per sequence will be used as model input. The rest is used as target and we only predict the first 100 tokens for each sequence for efficient computation.

5.2 Autoregressive Generation and Loss Computation

We use `model.generate()` with greedy coding to produce tokens conditioned on the context. This is an ideal method as it generates future tokens based solely on past context. It appends one new token at each step and feeds it back into the model for the next prediction.

The output scores (logits) are returned for each step, allowing us to compute the per-token cross-entropy loss against the ground truth target tokens. The average token-level loss is recorded per sequence.

5.3 Numerical Decoding and MSE Calculation

After receiving the token predictions, we decode them into text using Qwen tokenizer's built-in `decode` method. This converts a predicted token sequence back to a string following the LLMTIME format[1].

Next, we split the string by semicolons (‘`;`‘) to separate individual timesteps, and then by commas (‘`,`‘) to isolate the prey and predator values at each timestep. Each component is then converted to a floating-point number. Any non-numeric element are safely ignored.

The same decoding procedure is applied to the ground truth target tokens. We then compute the Mean Squared Error (MSE) between the predicted and true numeric sequences. The model may return illegal elements instead of a number, so we ignore these illegal elements and then calculate MSE for sequences with the same length between predicted system and ground truth system only.

MSE is the primary evaluation metric, as it directly reflects the discrepancy between the predicted and ground-truth trajectories at the sequence level. The average token-level validation loss is used as a secondary indicator to provide insights into the model's token generation behavior.

5.4 Preliminary Evaluation of an untrained model

The first step is to make an initial trial evaluation of the performance of the untrained `Qwen2.5-Instruct` model without any LoRA adaptation. This experiment serves as a baseline on the model's ability to forecast time series.

Figure 2 shows the resulting cross-entropy loss curves, indicating a general upward trend over prediction steps, which should be expected because of the nature of forecasting errors.

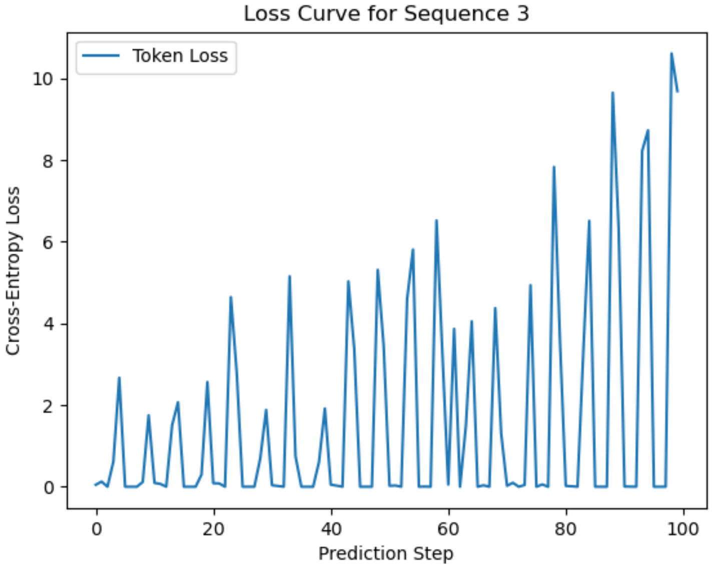

Figure 2: Validation token-level cross-entropy loss curve for Sequence 3.

Notably, the loss curves exhibit an oscillatory pattern due to the presence of formatting tokens such as commas and semicolons, which are easy for the model to predict.

To further analyze forecasting behavior, we visualized predicted versus ground truth trajectories. Even in the best-performing case in Sequence 9 of the validation dataset, as shown in Figure 3, the predicted trajectory diverges from the ground truth shortly after the start of prediction.

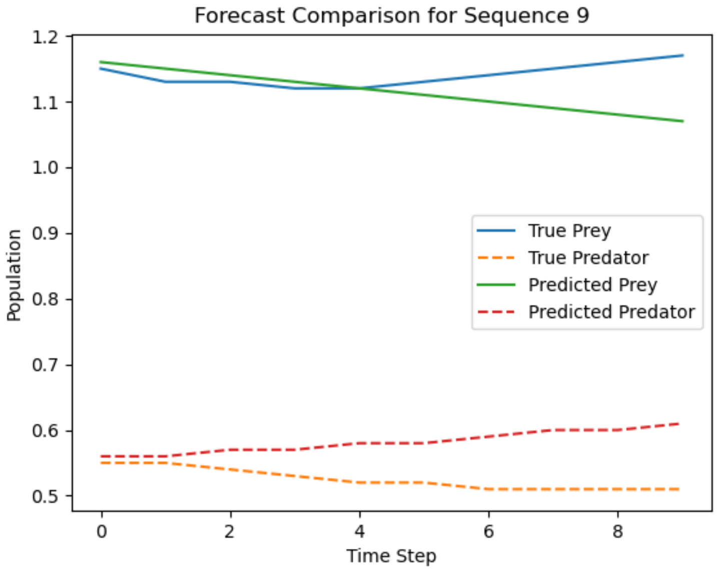

Figure 3: Forecast comparison on Sequence 9 (untrained model)

Figure 4 shows the predictions of Sequence 7, which significantly diverges and fails to capture the true dynamics of the system. Other sequences have similar results as Figure 4.

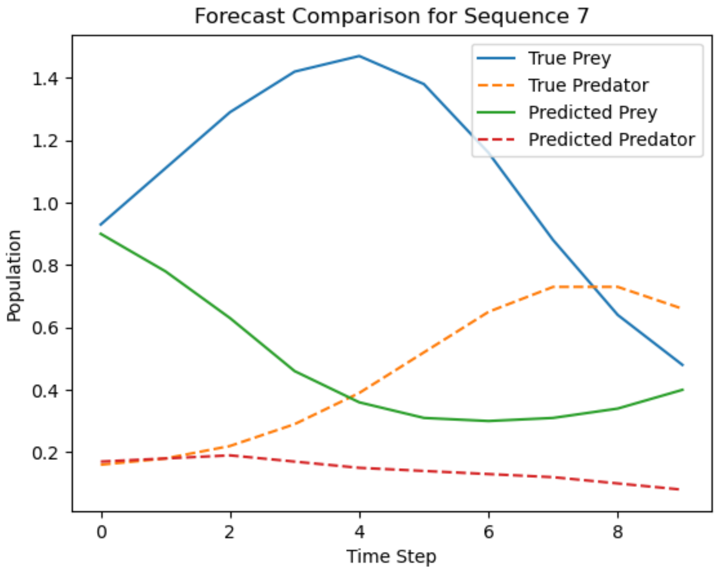

Figure 4: Forecast comparison on Sequence 7 (untrained model)

The evaluation of the untrained model on the validation set gives an average MSE of 0.2702 and an average cross-entropy validation loss of 1.785. These values will be compared with later experiments.

Overall, this baseline evaluation highlights the possibility and limitations of using an unadapted LLM for forecasting time series and motivates the use of LoRA.

6 Experiments

Formal experiments include the fine-tuning and training of`Qwen2.5-Instruct` using LoRA. We explored ranges of hyperparameters over learning rates and LoRA ranks. In addition, by selecting the best hyperparameter ranges, the model was trained with different context length. Finally, a longer experiments was conducted with all the best choices.

6.1 Training Methodology

6.1.1 Data Chunking

The first step before training a model is to further process the training set in a form of fixed-length inputs, so that we can facilitate batch training using PyTorch's `DataLoader`, which ensures efficient parallelization. The tokenized sequences are sliced into overlapping segments of fixed maximum context length, with a specified stride (256 tokens in this coursework). For each segment, if its length is shorter than this fixed context length, the sequence is padded on the left using the model's designated padding token. This ensures that all input tensors have consistent shape.

6.1.2 Training Procedure

The training procedure starts with the injection of LoRA adapters into the query and value projection layers of each self-attention block. When the model is ready, the chunked dataset is loaded into batches (batch size is 4 in this coursework).

Each training batch is passed through the model using the standard forward interface `model(input_ids, labels=...)`. This method is different from the `model.generate(...)` interface used during evaluation. The training-time forward pass enables a form of *teacher forcing* [4], in which the model is given the entire input sequence and asked to predict the next token at each position. Cross-entropy loss is computed between these predictions and the true next-token labels. In addition, to prevent the model from learning from padded positions, all padding tokens in the label tensor are masked by setting their value to `-100` for ignored loss positions. The model is optimized using the Adam optimizer.

6.2 A Trial Experiment with default hyperparameters

6.2.1 Training

We trained the Qwen2.5-Instruct model with a LoRA rank of 4 and a learning rate of 10^{-5} for 500 optimization steps. This is a trial experiment, served to validate the training pipeline and provide an initial sense of the model's ability to learn the dynamics of time series.

Figure 5 demonstrates the training loss over 500 optimizer steps. The light blue curve shows the raw loss at each step, which fluctuates dramatically, likely due to the inherent variability in the sampled sequences. This high variance is expected in early-stage autoregressive training of a LLM. A moving average over a window of 10 steps (shown in dark blue) is also visualized. The smoothed curve reveals a clear downward trajectory, indicating that the model gradually learns useful forecasting behavior despite noisy updates.

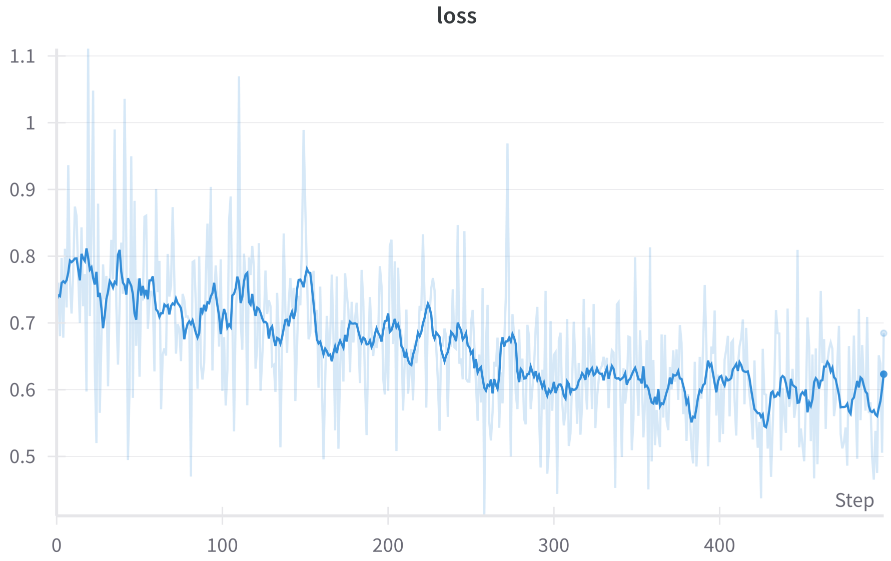

Figure 5: Training loss during trial experiment. Raw step-wise losses (light blue) and smoothed 10-step average (dark blue) are shown.

6.2.2 Evaluation

Among the 10 validation sequences evaluated in this trial training setting, the vast majority exhibited minimal improvement over the untrained model. Figure 6 shows the results of Sequence 9. Compared with Figure 3, the improvement is negligible.

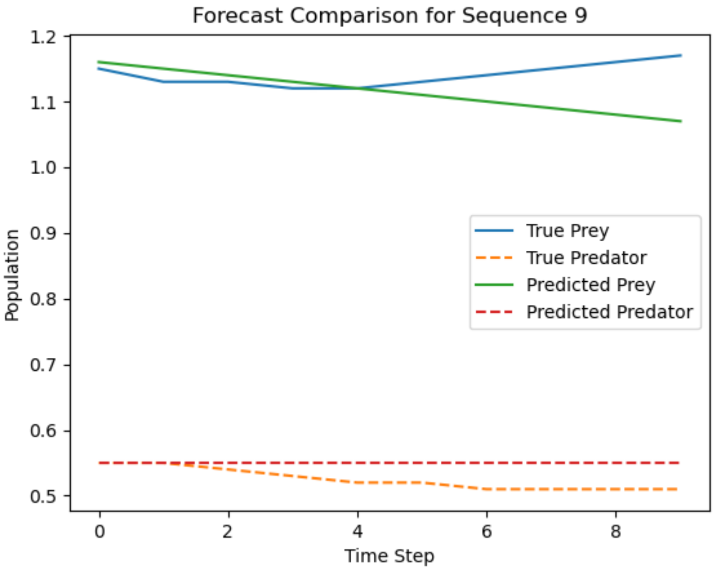

Figure 6: Forecast comparison on Sequence 9 (trial trained model)

However, one notable exception was Sequence 7, which demonstrated a substantial improvement in both prey and predator trajectories as shown in Figure 7. The model successfully captured the correct underlying dynamical behavior, indicating improved qualitative forecasting capability after LoRA fine-tuning. Compared with Figure 4, this improvement starts making the forecasting meaningful.

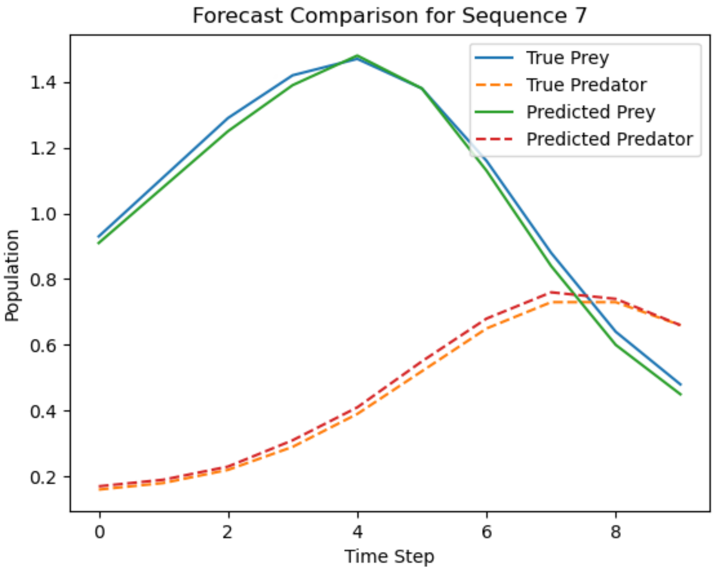

Figure 7: Forecast comparison on Sequence 7 (trial trained model)

Table 2 reports the metrics of evaluation of the untrained model and the trial experiment. The LoRA-trained model achieves a substantial reduction in MSE, indicating improved numerical forecasting accuracy. Meanwhile, the cross-entropy loss shows a moderate decrease, suggesting more confident token-level predictions.

Table 2: Validation performance of the untrained and trained models (Task 3a).

Model	**Validation MSE**	**Validation Loss**
Untrained Qwen2.5-Instruct	0.27025	1.78500
LoRA-trained (rank = 4, lr = 10^{-5}, 500 steps)	0.12480	1.65628

6.2.3 Computational Cost

For this experiment, each optimizer step is estimated to consume approximately $10^{12.683}$ FLOPs. Given the training was conducted for 500 steps, the total training cost amounts to approximately $10^{15.382}$ FLOPs.

6.3 Experiments with Hyperparameter Tuning

To further improve the forecasting performance, we conducted a grid search over two hyperparameters: the *rank* of LoRA adaption and *learning rate*. We explored three different ranks ($r \in \{2, 4, 8\}$) and three learning rates ($\alpha \in \{10^{-5}, 5 \times 10^{-5}, 10^{-4}\}$), resulting in a total of nine configurations. Each configuration was trained for 1000 optimizer steps and subsequently evaluated on the validation set using MSE and token-level cross-entropy loss.

6.3.1 Training

Figure 8 presents the training loss curves for all nine combinations of LoRA rank and learning rate. For each run, the step-wise loss was smoothed using a 10-step moving average to reveal overall trends. All configurations demonstrate a consistent downward trend, indicating successful convergence over the 1000 optimization steps.

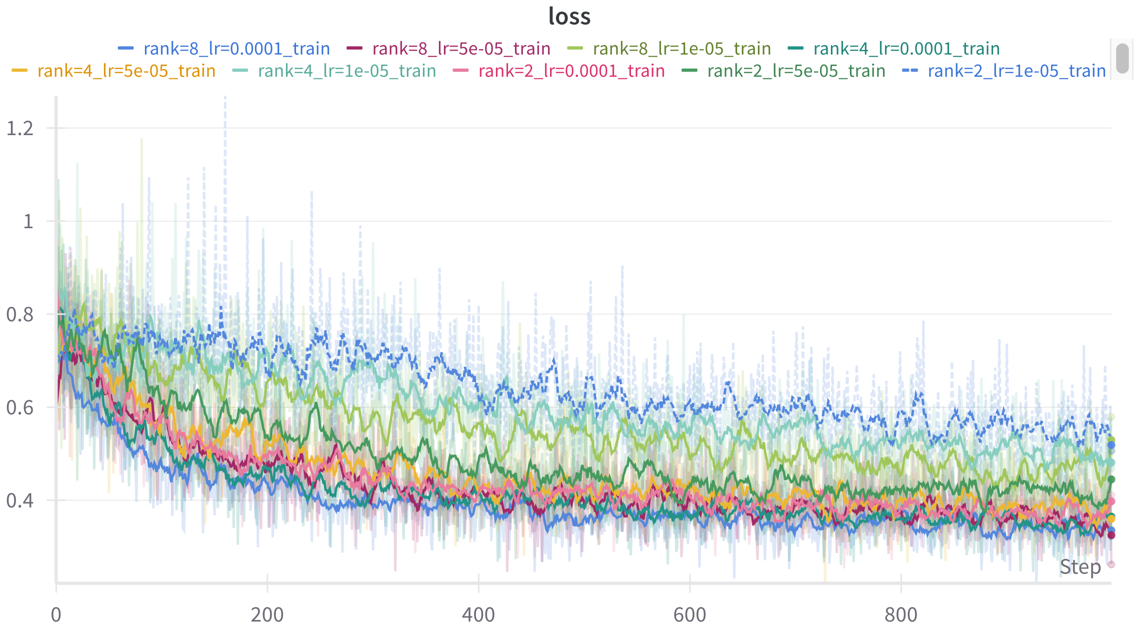

Figure 8: Training loss curves for the nine hyperparameter configurations.

6.3.2 Evaluation

Table 3 presents the validation MSE for each experiment across different ranks and learning rates. The best overall performance is achieved at the largest rank and learning rate combination (rank = 8, learning rate = 10^{-4}), with an MSE of **0.0071**, indicating the highest ability in forecasting. The overall trend shows that increasing either the rank or the learning rate generally leads to better forecasting performance. This is consistent with the intuition that higher rank enables larger capacity in the low-rank adaptation, while higher learning rates accelerate convergence within the limited optimization steps.

Table 3: MSE for each experiment.

Rank \ LR	1×10^{-5}	5×10^{-5}	1×10^{-4}
2	0.0561	0.0100	0.0148
4	0.0375	0.0251	0.0133
8	0.0328	0.0256	**0.0071**

However, one notable exception arises at learning rate = 5×10^{-5}, where the model with rank = 2 significantly outperforms those with higher ranks. This irregular behaviour may be attributed to the stochastic nature of initialization.

Validation cross-entropy loss values, as shown in Table 4, exhibit limited variation across configurations in these experiments.

Table 4: Validation cross-entropy loss (token-level) for each LoRA configuration.

Rank \ **LR**	1×10^{-5}	5×10^{-5}	1×10^{-4}
2	1.7700	1.4630	1.6718
4	1.6040	1.7418	1.8379
8	1.6141	1.9905	1.6746

As shown in Figure 9, this model consistently achieved the lowest training loss, indicating strong learning ability on the training data, which aligns well with the evaluation results.

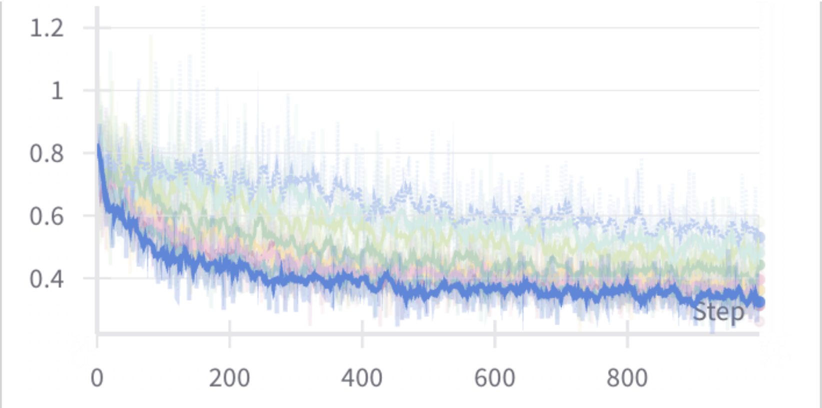

Figure 9: Highlighted training loss curve of the model with the best evaluation performance (`rank=8, lr=`10^{-4})

Across all ten validation sequences, the predicted values now exhibit strong alignment with the ground truth. Figure 10 shows the system of Sequence 5 as an example. The predicted trajectories closely follow the true ones throughout the given time range.

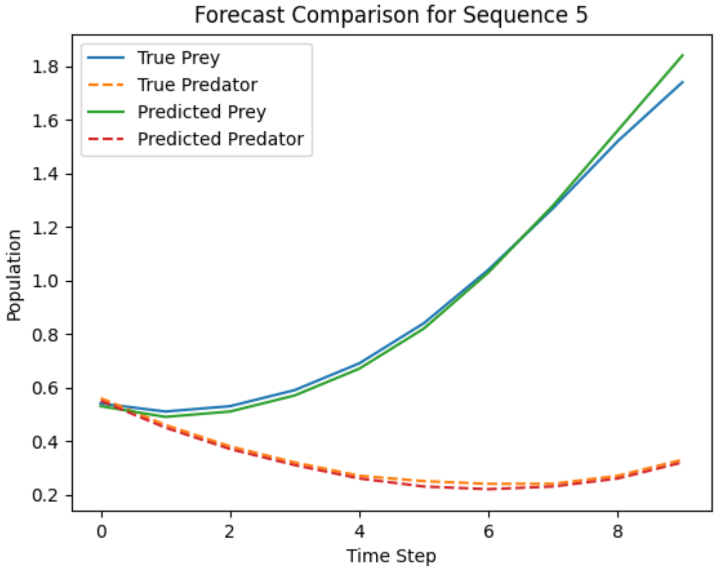

Figure 10: Forecast comparison on Sequence 5 from the best-performing model.

6.3.3 Computational Cost

Since LoRA only adds small-rank linear adapters to selected weight matrices, the additional FLOPs per step remain several orders of magnitude smaller than the base model's inference cost. It is very safe to ignore the impact of different LoRA adapation with respect of computational cost.

In our experiments, we estimate that one training run of 1000 optimization steps costs approximately $10^{15.683}$ FLOPs. Across all nine hyperparameter configurations in our grid search, the total estimated FLOPs sum to roughly $10^{16.637}$.

6.4 Experiments with different context length

6.4.1 Training

To investigate how the amount of historical information affects model learning, we conducted experiments varying the context length per sequence during training. The best-performing hyperparameter setting (rank 8 and learning rate 10^{-4}) was used.

Figure 11 shows the training loss curves when the model was trained with three different context lengths: 128, 512, and 768. All three curves exhibit a clear downward trend, indicating successful training. Models trained with longer context lengths (512 and 768) converge to lower training losses than those trained with shorter context (128), suggesting that longer historical context improves the model's ability to fit the training data. Nevertheless, the difference between 512 and 768 is marginal, implying a possible saturation point where additional context provides diminishing returns.

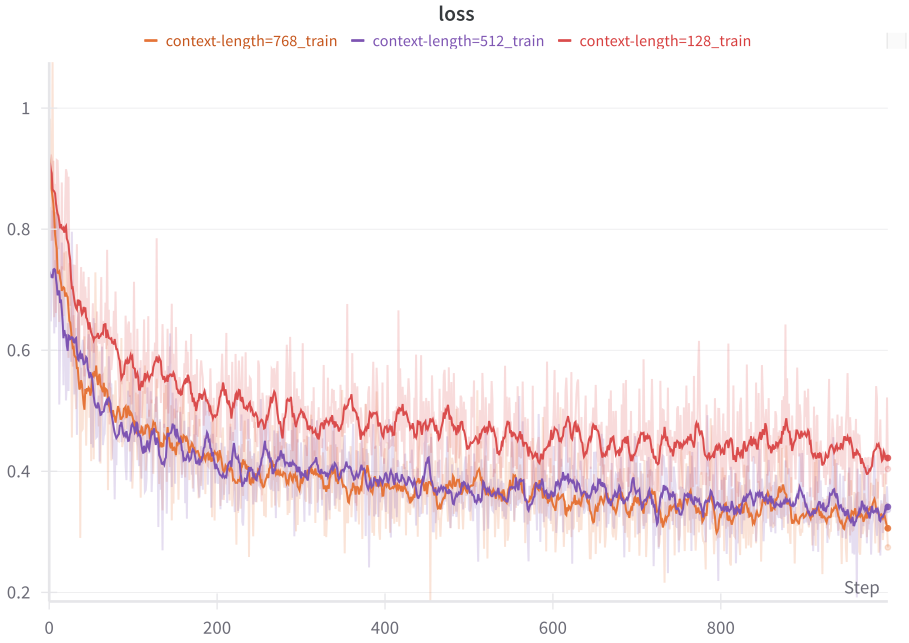

Figure 11: Training loss curves under different context lengths.

6.4.2 Evaluation

Table 5 demonstrates the effect of different context lengths. As the context length increases from 128 to 768, both validation loss and forecast MSE consistently decrease. Notably, the forecast MSE drops dramatically, indicating significantly improved prediction accuracy. This trend suggests that longer context provides the model with richer historical information, which is crucial for accurately modeling dynamics. Validation loss also improves at the token level.

Table 5: Evaluation metrics across different context lengths.

Context Length	Validation Loss	Forecast MSE
128	2.6090	0.1177
512	2.0401	0.0230
768	1.5362	0.0051

However, when comparing the actual trajectory predictions, the improvement from context length 512 to 768 appears less substantial. Models trained with context length 512 are already capable of capturing the correct dynamical behavior, and although the

768-context model achieves better numerical alignment on average, it does not provide perfectly predicted trajectory. The best case, as shown in Figure 12, is similar as Figure 10, except for the perfect alignment during the initial prediction window.

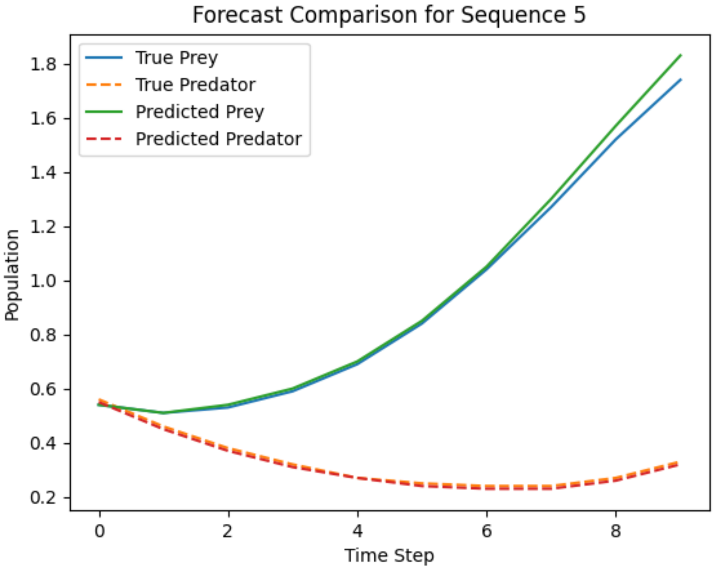

Figure 12: Forecast comparison on Sequence 5 (context length=758).

6.4.3 Computational Cost

The FLOPs for each experiment were approximately $10^{15.061}$, $10^{15.683}$, and $10^{15.871}$, respectively. The total computational cost for this context-length analysis phase was therefore around $10^{16.127}$ FLOPs.

6.5 Final Experiment with longer optimizer steps

6.5.1 Training

To obtain the final forecasting model, we selected the best-performing hyperparameter configuration identified in earlier experiments: LoRA rank 8, learning rate 10^{-4}, and context length 768. We extended the training range to 3000 optimization steps to allow for further convergence.

Figure 13 shows the training loss of the final model. It indicates that the model has largely converged after the first 1000 steps.

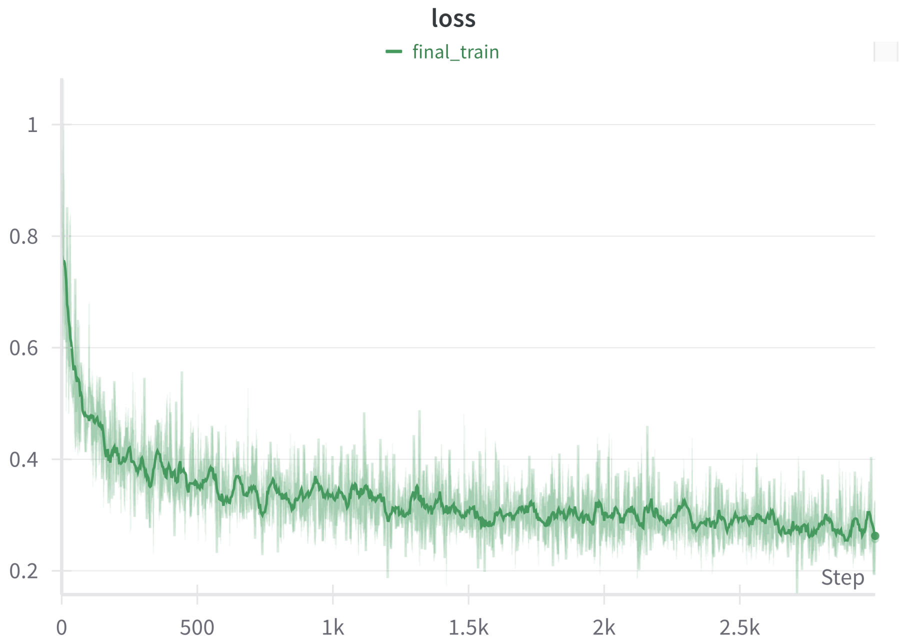

Figure 13: Training loss curve of the final model (rank=8, lr=10^{-4}, context length=768, 3000 steps).

6.5.2 Evaluation

Evaluation on both the validation set and the held-out test set were performed. This design ensures that the results are directly comparable to earlier experiments, while also enabling us to assess the model's generalization performance on other data. The results are shown in Table 6.

Compared to the previous experiments (Table 5), both the cross-entropy loss and MSE on the validation set show a clear reduction, indicating improved token-level and forecast-level performance. Furthermore, the test set results demonstrate similarly low values for both metrics, suggesting that the trained model generalizes well to other systems.

Dataset	**Cross-Entropy Loss**	**MSE**
Validation set	1.2128	0.0021
Test set	1.9080	0.0071

Table 6: Final model performance on validation and test sets.

Extending training to 3000 steps results in some improvement in trajectory prediction among 10 sequences, although the improvement diminishes. The best case, as shown in Figure 14, maintains perfect alignment with the ground truth trajectories over a longer prediction horizon than Figure 12.

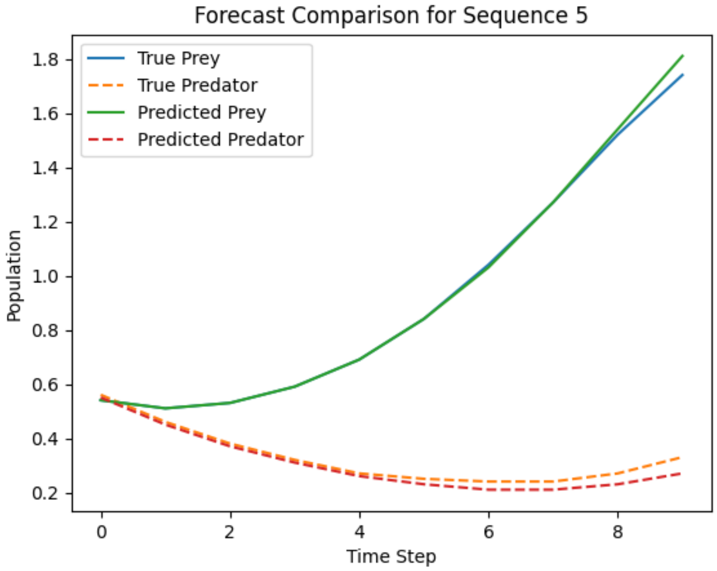

Figure 14: Forecast comparison on Sequence 5 (final model).

6.5.3 Computational Cost

In the final experiment, the estimated FLOPs is roughly $10^{16.350}$.

7 Conclusion, Recommendation and Further Improvement

Our experiments illustrate that using LLMs for time series forecasting can be highly efficient, especially when trained with lightweight fine-tuning techniques such as LoRA. Unlike full model training, LoRA fine-tuning significantly reduces computational cost while enabling the model to adapt to the specific forecasting task.

By observing the trend across our experiment, the following recommendations are provided for fine-tuning LLMs on time series under tight compute budgets. It is generally beneficial and safe to prioritize the increasing of the LoRA rank, learning rate, context length, and training steps, though we note that the marginal gains diminish and are not always guaranteed.

This leads to further improvements in forecasting performance. The effects of hyperparameters should be further explored. In particular, exploring a broader and denser range of values for rank and context length could find the best configurations. Additionally, the use of adaptive learning rate schedules should improve convergence stability and generalization.

Finally, another improvement is to preserve more decimal precision, which may allow the model to better capture information on the underlying dynamics.

8 Total FLOPS

Summing the total FLOPS of this coursework is estimated to be:

$$\text{Total FLOPs} \approx 10^{15.382} + 10^{16.637} + 10^{16.127} + 10^{16.350} \approx 10^{16.911}$$

This is 81.47 percent of the 10^{17} budget.

References

[1] N. Gruver, M. Finzi, S. Qiu, and A. G. Wilson, "Large language models are zero-shot time series forecasters," 2024.

[2] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[3] Q. Team, "Qwen2.5 technical report," 2025.

[4] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in neural information processing systems*, vol. 28, pp. 1171–1179, 2015.

9 Appendix

9.1 More plots of trajectories from grid search to the final longer experiment will be placed in the folder "Trajectories".

9.2 AI generation tool

Declaration of AI generation tools for the report part is made here:

Although some of the advise is not accepted, ChatGPT was used to help this coursework:
1. The format of the mathematical calculations was helped by it to make the process in a publication quality.
2. The format of plots, tables and itemized expression was helped to make things in a publication quality.
3. It suggested some alternative wording
4. It helps understand the model architechture of Qwen 2.5 in a simplified version for FLOPs calculation. Followed by that, it helps implement the description of FLOPs calculation.
5. It provided proofreading for some of text, including the description of evaluation, training, model comparison and conclusion.