M2 Coursework: Evaluating Large Language Model Performance as Time Series Forecasters

jn492

April 14, 2025

Word count: 2974

1 Introduction

Gruver et al. recently demonstrated potential in using Large Language Models (LLMs) for time series forecasting [2]. However, tuning full weight matrices in transformers can be computationally expensive. This report explores the application of a low rank adaptation(LoRA) to LLM Qwen2.5, with the goal of reducing computational costs when forecasting prey predator dynamics. Computational cost will be tracked using the total number of floating-point operations (FLOPS) for every experiment, and the performance will be evaluated using machine learning loss metrics such as cross-entropy loss, mean squared error (MSE), and mean absolute percentage error (MAPE).

This section is split into two sections. Section 2 establishes a baseline performance for the Qwen model for time series forecasting prior to applying LoRA. Section 3 evaluates LoRA's performance with hyperparameter tuning and presenting the final results.

2 Baseline: Qwen model, Preprocessor and Model performance

2.1 About Qwen2.5

The language model for this report is Qwen2.5 0.5B, developed by Alibaba Cloud and available on Hugging Face. The model's architecture, illustrated in Figure 1, comprises 24 transformer blocks. Each block incorporates key components including multi-headed attention, root mean squared normalization (RMS-NORM), a SwiGLU activation function, and residual connections [1].

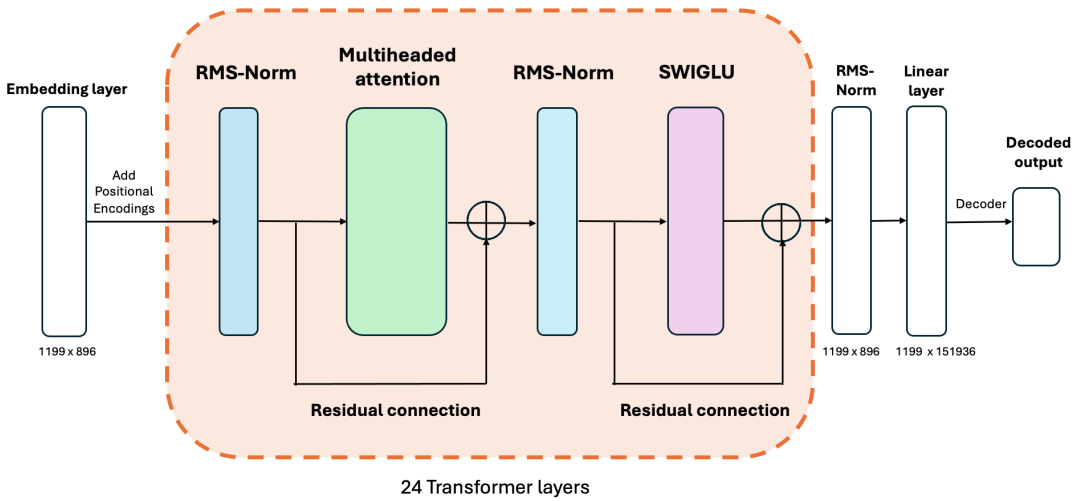


Figure 1: Qwen2.5 Architecture

Below is a more detailed description of these operations, which will form the basis of FLOPS calculations:

1. **Embedding Layer:** Each input token is mapped to a dense vector of size 896. Given a sequence length of 1199, the resulting input matrix has shape 1199×896. Qwen uses Rotary positional encoding, which captures the relative position of each token. For ease of computation, FLOPS will only account for the addition of this Positional encodings to the embedding layer.

2. **Transformer Block (repeated 24 times per forward pass):** Each of the 24 layers contains the following components:

   - **1st RMSNorm:** Root Mean Square Layer Normalization is applied row-wise to each token embedding. This scales the token vector so that its magnitude is close to 1 while preserving direction (i.e. the relative values across dimensions). No learnable weights or biases are introduced at this stage.
   - **Multi-Head Attention (MHA):** Although Qwen uses group query attention, we assume standard multi-head attention in FLOPS calculation for simplicity. Query, Key, and Value matrices are computed through learned projections. With 14 attention heads, the input is split into 14 subspaces.Each head performs attention in parallel, capturing different patterns that a single head might miss. The results are then concatenated and projected back to the model dimension.
   - **1st Residual Connection:** The attention output is added to the input via a skip connection. This allows the transformer to learn only the difference between the input and the transformed output. It avoids overwriting useful representations and helps prevent issues like vanishing or exploding gradients in deep models.
   - **2nd RMSNorm:** A second RMS normalization is applied to the post-attention output to stabilize the scale of the values before entering the next module.
   - **SWIGLU Activation:** SWIGLU is a gated non-linear activation that combines SiLU (also known as Swish) with a Gated Linear Unit. It introduces a gating mechanism where one part of the input controls the flow of the other, enabling more flexible and expressive transformations. The input is split into two parts, and the activation is computed as:

     $$\text{SWIGLU}(x) = \text{SiLU}(xW_1) \odot xW_2$$

     where $\text{SiLU}(z) = z \cdot \text{sigmoid}(z)$. This is followed by an up-projection and a down-projection, allowing the network to transform the input through a higher-dimensional space, which improves its representation power.
   - **2nd Residual Connection:** As with attention, the output of the SWIGLU block is added back to its input using a skip connection to preserve features and support gradient flow.

3. **Final Layers:**

   - **RMSNorm:** A final RMS normalization is applied to the output of the last transformer layer.
   - **Linear Projection:** This projects the hidden dimension from size 896 to the vocabulary size of 151,936, resulting in an output matrix of shape $1199 \times 151{,}936$. This step maps each token's embedding to a vector of vocabulary scores.
   - **Decoder:** A softmax decoder converts logits into probabilities to select the most likely token, thus determining which token in the vocabulary is most likely to come next.

This forms the framework for calculating floating-point operations (FLOPS) across experimental configurations. Table 1 offers a mathematical breakdown of computational steps for a single forward pass of the original Qwen model. An example of running a forward pass in unmodified Qwen, given 1199 x 896 input matrix, is given in Appendix B, where a total 1.27×10^{12} FLOPS were computed. These are calcaulated based on the number of FLOPS per mathematical operation, provided in Appendix **??**.

Table 1: FLOP breakdown for each component of the Qwen Transformer. $S \times D$ is the input shape (sequence length $\times$ hidden size), H is the number of attention heads, z is the SwiGLU up-projection size, and v is the vocabulary size. FLOPs from QKV Projection to the second Residual Connection should be multiplied by 24 to account for the 24 stacked layers.

Transformer Step	**Equation**	**FLOP Expression**
Add Positional Encodings	Embedding + Positional Encodings	$S \times D$
RMSNorm	$\text{RMS}(x) = \sqrt{\text{mean}(x^2)}$	$(S \times D) + (S \times (D-1)) + D + D$
	$\frac{x}{\text{RMS}(x)} \cdot w$	$(S \times D) + (S \times D) + (S \times D \times (2D-1))$
QKV Projection	$xW_Q, xW_K, xW_V + b_Q, b_K, b_V$	$3 \times (S \times D \times (2D-1)) + 3 \times (S \times D)$
Multi-head Attention	$QK^\top$	$H \times (S \times S \times (2 \times D/H - 1))$
	$\text{Softmax}\left(\frac{QK^\top}{\sqrt{D/H}}\right)$	$H \times (S \times S \times 10)$
	$V_h \cdot \text{Softmax}\left(\frac{K_h^\top Q_h}{\sqrt{D/H}}\right)$	$H \times (S \times S \times (2 \times D/H - 1))$
	Concatenation + Projection	$S \times D \times (2D-1)$
Residual Connection	$x + \text{Attention}(x)$	$S \times D$
RMSNorm	$\text{RMS}(x) = \sqrt{\text{mean}(x^2)}$	$(S \times D) + (S \times (D-1)) + D + D$
	$\frac{x}{\text{RMS}(x)} \cdot w$	$(S \times D) + (S \times D) + (S \times D \times (2D-1))$
SwiGLU	$xW_{\text{up}}, xW_{\text{gate}}$	$2 \times (S \times D \times (2z-1))$
	$\text{SiLU}(xW_{\text{gate}}) \cdot (xW_{\text{up}})$	$S \times z \times (10+1+1) + S \times z$
	Output Projection W_{down}	$S \times z \times (2D-1)$
Residual Connection	$x + \text{FFN}(x)$	$S \times D$
RMSNorm	$\text{RMS}(x) = \sqrt{\text{mean}(x^2)}$	$(S \times D) + (S \times (D-1)) + D + D$
	$\frac{x}{\text{RMS}(x)} \cdot w$	$(S \times D) + (S \times D) + (S \times D \times (2D-1))$
Linear layer	$x \cdot v$	$S \times v \times (2D-1)$

During training, the backward pass is assumed to requires twice the number of FLOPS as the forward pass. As a result, the total FLOPs per update is three times that of a single forward pass, as shown in Equation 1, where total flops for 1 pass is denoted as F.

$$\begin{aligned} \text{Forward Pass (Original Qwen)} &= F \\ \text{Backward Pass (Original Qwen)} &= 2F \\ \text{Total Update FLOPs (Original Qwen)} &= F + 2F = 3F \end{aligned} \tag{1}$$

This assumption in Equation 1 and the operations enumerated in Table 1 is used in the `flops()` function defined in `src/flops.py`, which computes FLOPs based on input size parameters. The function

accepts positional arguments of input dimensions (e.g., sequence length, hidden dimension) and keyword arguments such as number of heads (default: 14), vocabulary size (default: 151,936), and SwiGLU up-projection size (default: 4864). Additional flags like `training=True` or `validation=True` can be set, along with the number of steps. If enabled, the total FLOPs are scaled by the number of training or validation steps accordingly.

2.2 Data Preprocessing

Prey and predator population density data were collected across 1,000 ecological systems and sampled over 100 consecutive time steps. The distribution of prey and predator values across these systems is illustrated in Figure 2.

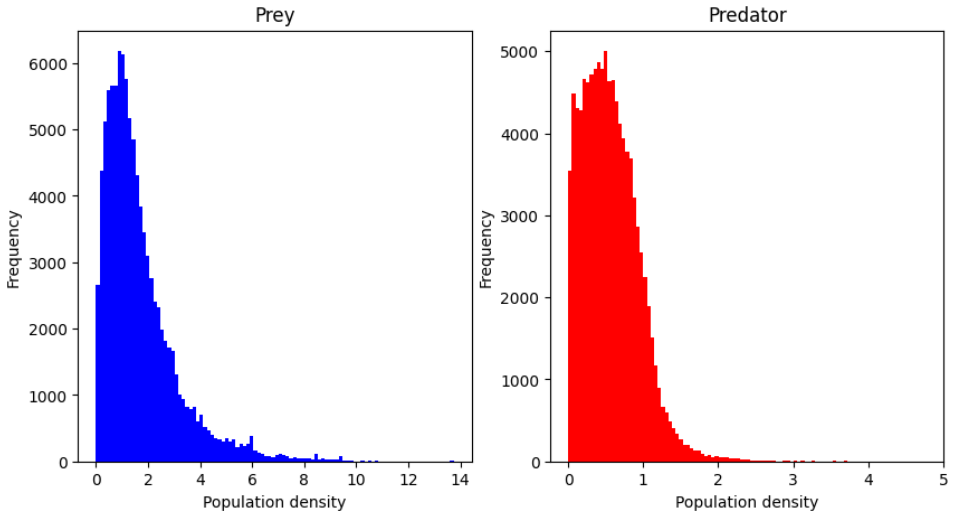


Figure 2: Histograms showing the frequency for prey and predator population density values in 1000 sequences

2.3 Data Preprocessing

The data was processed using the LLMTIME scheme. First, prey and predator values were normalized by dividing by $\alpha = 2.53$, ensuring that 90% of values fall between 0 and 1, as suggested by Gruver et al. [2]. While this prevents token wastage on large values, it was less critical here due to already small population sizes. Like Gruver et al., I avoided max-normalization to allow the model to observe digit variations that may help extrapolate beyond the training range.

Next, population trajectories were formatted as strings:

$$[\text{Prey}_0, \text{Predator}_0 \ ; \ \text{Prey}_1, \text{Predator}_1 \ ; \ \ldots]$$

Values were rounded to three decimal places to balance token count and numerical precision. Each character, digit, and delimiter became a token, producing 1199 tokens per sequence. This was implemented in `src/preprocessor.py`, which returns a preprocessed list of 1000 sequences from the `.h5` file.

2.4 Evaluating an untrained Qwen model

In the code, sequence 330 and 167 are randomly selected with a random seed (423), and their respective preprocessed and tokenized values are displayed in Table 2.

Both sequences were used to evaluate the untrained Qwen model. The PyTorch command `.generate()` was used to perform inference, where the first 80 prey-predator pairs served as input, and the remaining 20 were autoregressively generated. The output tokens were decoded back into a string format, then parsed to extract the predicted prey and predator values.

Table 2: Selected Sequence Preprocessing and Tokenization

Sequence	**Preprocessed Values**	**Token Values**
330	$0.417, 0.360; 0.360, 0.284; \ldots; 0.484, 0.067$	$[15, 13, 19, 16, 22, 11, 15, 13, 18, 21, 15, 26, \ldots, 22]$
167	$0.339, 0.350; 0.286, 0.297; \ldots; 0.505, 0.186$	$[15, 13, 18, 18, 24, 11, 15, 13, 18, 20, 15, 26, \ldots, 21]$

Ellipses (...) indicate truncated data. Full sequence values are available in the original code.

Figure 3 and Figure 4 compare ground truth and forecasted trajectories for systems 330 and 167. The model performs well on system 330 but poorly on 167, likely due to system 330's stable periodic amplitudes giving rise to more accurate predictions. These results are in agreement with Gruver et al. [2], such that pretrained LLMs like Qwen can forecast well on regular systems. However, they may struggle with more complex patterns. This highlights the need for tuning, which will be discussed in Section 3.

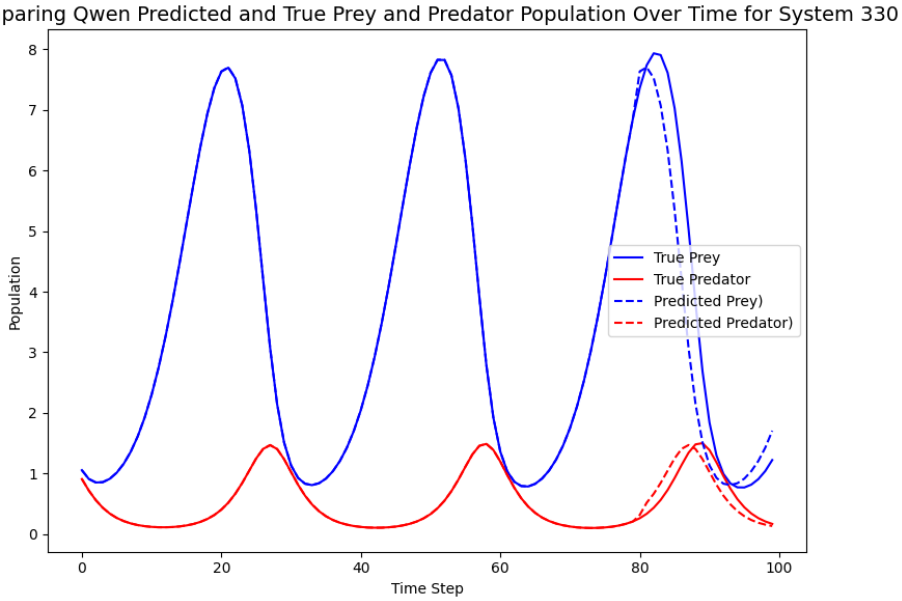


Figure 3: Prey (blue) and predator (red) population densities over 100 time steps for predicted (dotted) and true (solid) values of system 330

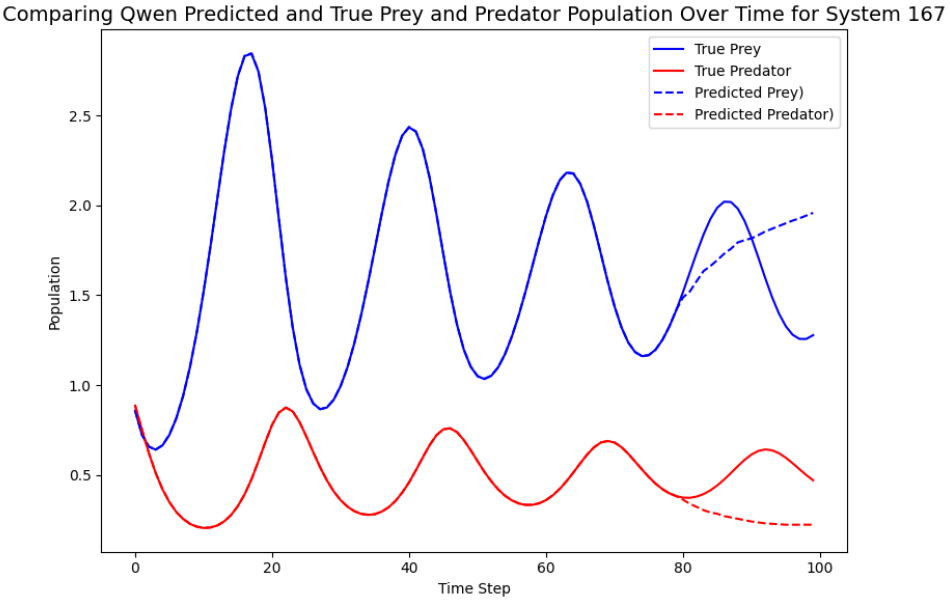


Figure 4: Prey (blue) and predator (red) population densities over 100 time steps for predicted (dotted) and true (solid) values of system 167

This study uses MSE and MAPE to evaluate model performance. MSE captures absolute prediction errors for prey and predator separately, while MAPE provides a scale-invariant view, which may offer better insights into model performance since prey populations typically outnumber predators. Although it is worth nothing that both populations are of similar magnitude, so the discrepancy between MAPE and MSE should be small.

Metric	System	Prey	Predator
MSE	330	0.923	0.053
	167	0.150	0.082
MAPE (%)	330	23.491	30.559
	167	21.858	46.143

Table 3: Comparison of Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) between ground truth and untrained Qwen forecasts over the final 20 time steps.

Table 3 shows that prey MSE is higher than predator MSE, especially for system 330, due to its larger population scale, thus larger absolute errors. However, visual inspection reveals that predictions for system 330 track the true prey trajectory better than in system 167, highlighting the value of case-by-case visualisation. Overall, MAPE suggests the model predicts prey more accurately than predators.

3 LoRA: LoRA-Qwen Model, performance, Hyperparameter search, Final model

3.1 About the LoRA Qwen Model

.

LoRA is a less expensive alternative to updating the weights in a multi-headed attention matrix. When wrapping a layer with LoRA, the weight update can be defined in Equation 2.

$$W_{\text{LoRA}} = W_0 + BA \tag{2}$$

LoRA operates by freezing the original weight matrix (W_0), and updating smaller matrices A and B to reduce computational load per update. For my implementation, only the Query and Value layers are wrapped with LoRA. For example, if LoRA rank is 4, then rather updating a full Query weight matrix W_Q of size 896 x 896 (8.03×10^5 elements), only two smaller 896 x 4 weight matrices (7.168×10^3 elements) are updated. Hence, ony the smaller A and B matrices are being tuned when implementing LoRA. The LoRA implementation only changes the QKV projection as noted in Table 4.

Table 4: LoRA-adjusted QKV Projection in the Transformer block, where r is the chosen LoRA rank, S is the sequence length, and D is the hidden dimension.

QKV Projection	$W_{Q_{\text{lora}}} = W_Q + BA$	$D \times D + D \times D \times (2r - 1)$
	$W_{V_{\text{lora}}} = W_K + BA$	$D \times D + D \times D \times (2r - 1)$
	$xW_{Q_{\text{lora}}},\ xW_{K_{\text{lora}}},\ xW_V$	$3 \times (S \times D \times (2D - 1))$
	$+ b_Q,\ b_K,\ b_V$	$3 \times (S \times D)$

Note that the original FLOPs calculation does not fully capture LoRA-Qwen behavior. LoRA freezes the main weights, reducing backward pass costs, while introducing extra forward operations via low-rank matrices A and B. Applying the standard FLOPs formula exaggerates total cost by ignoring these savings. A more accurate estimate would scale the backward FLOPs by the ratio of LoRA's trainable parameters, as shown in Equation 3.

$$\begin{aligned} \text{Forward Pass (LoRA-Modified Qwen)} &= F \\ \text{Backward Pass (LoRA-Modified Qwen)} &= 2F \cdot \frac{N_{\text{LoRA}}}{N_{\text{Original}}} \\ \text{Total Update FLOPs (LoRA-Modified Qwen)} &= F + 2F \cdot \frac{N_{\text{LoRA}}}{N_{\text{Original}}} \end{aligned} \tag{3}$$

However, since FLOPs in this coursework are used primarily to track computational load rather than to precisely measure LoRA's efficiency, I will retain the simplified assumption of $1F$ for the forward pass and $2F$ for the backward pass per update.

3.2 LoRA-Qwen Performance

3.2.1 Training and Evaluation methodology

Due to computational constraints, only 100 out of the 1000 available systems were used to evaluate the LoRA-modified model for this section. These systems were randomly selected, and then divided into 70 training, 15 validation, and 15 test sets using `train_test_split()`. FLOPS will be calculated for experiment, accounting for both training and validation operations. FLOPS are calculated for each experiment and summarized in Section 3.5. The model was trained in 500 steps with a batch size of 4, using a learning rate of 1×10^{-5}, a context length of 512 and a stride of 256. Validation loss was logged every 10 steps with stride=512 to monitor for potential overfitting using Wandb.

Two modes of evaluation were carried out: 1) using `model.eval()` to calculate the average cross-entropy loss when the model performs autoregressive predictions on the test set, and 2) using `model.generate()` to predict the next 20 values given the first 80 values of each test sequence, with average MSE and MAPE calculated across all 15 test sequences. Individual MSE and MAPE results for both the trained and untrained models for each system are presented in Appendix C.

3.2.2 Trained and untrained model performance

Since identical test sets were used for evaluation of both models, it is valid to calculate and compare the average MSE and MAPE across all test systems, as shown in Table 5. Table 5 shows demonstrates a sig-

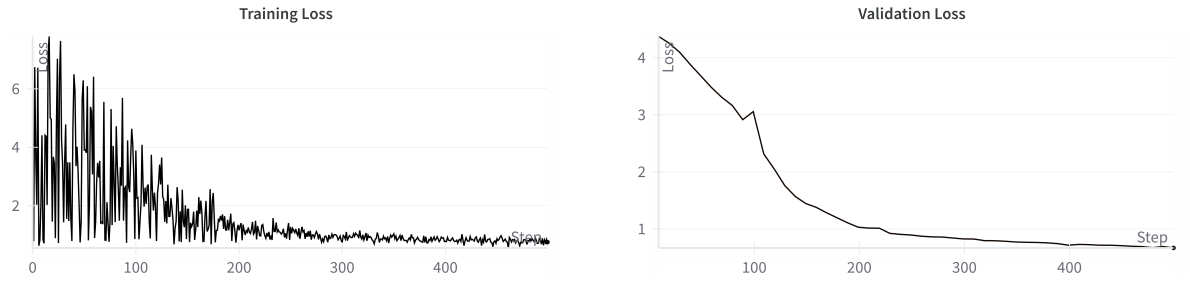
nificant improvement in cross-entropy loss (reduction of 3.803) after training. While predator predictions improved slightly, prey predictions improved more, likely due to higher population densities.

Table 5: Performance Comparison and Improvements from Training

Metric	**Untrained Model**	**Trained Model**	**Improvement**
Cross-Entropy Loss	4.523	0.720	3.803
Average MSE (Prey)	0.562	0.396	0.166
Average MAPE (Prey) [%]	32.211	27.750	4.461
Average MSE (Predator)	0.091	0.091	0.000
Average MAPE (Predator) [%]	66.859	63.926	2.933

Training and validation loss curves generated by Wandb in Figure 5a and Figure 5b show a steady decline, implying the model did not overfit to the training data.

The number of FLOPS used is 9.32×10^{14}.



(a) Training Loss

(b) Validation Loss

Figure 5: Training and Validation Loss for 500 steps of default LoRA-Qwen model

3.3 Hyperparameter tuning

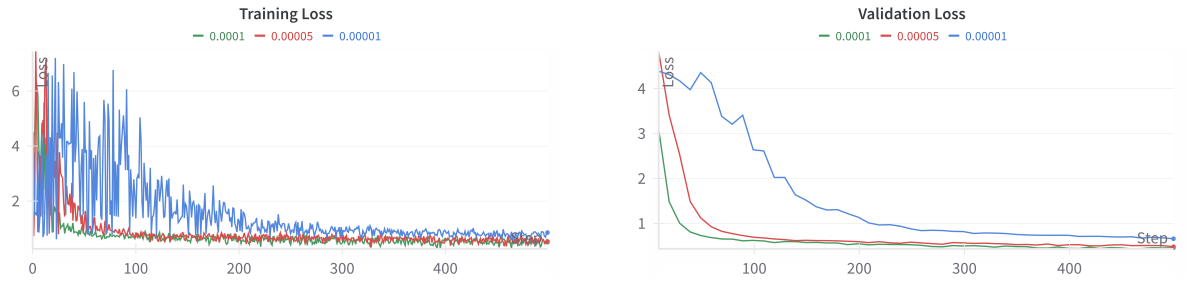
3.3.1 Learning Rate

To improve upon the trained LoRA-Qwen model, hyperparameter tuning was conducted. Specifically, three learning rates were tested at rank 4: 1×10^{-5}, 5×10^{-5}, and 1×10^{-4}. Cross-entropy loss was computed for each configuration, with 1×10^{-4} yielding the lowest loss, as shown in Table 6.

Table 6: Cross Entropy Losses for Different Learning Rates (Rank = 4), with the best learning rate 0.0001 highlighted in green.

Learning Rate	**Train Loss**	**Validation Loss**	**Test Cross Entropy Loss**
1e-5	0.856	0.659	0.711
5e-5	0.532	0.481	0.527
1e-4	0.508	0.460	**0.495**

When examining the validation and training loss curves in Figure 6, all three learning rates exhibited a concurrent decrease in both training and validation loss, indicating that overfitting did not occur.



(a) Training Loss

(b) Validation Loss

Figure 6: Training and Validation loss for different learning rates

As highlighted in green in Table 6, the 1×10^{-4} learning rate outperformed the smaller rates, likely because it more effectively avoided local minima and achieved better convergence. LLMs do not intuitively understand numbers as quantities, but as tokens like 'banana' or '%'. Choosing a larger step size likely helped LLMS focus on the larger patterns in tokens, such as patterns in the time series sequence, rather than on nuanced patterns like digits within a single prey or predator count. This advantage is further supported by the training and validation loss curves, where the model trained with the larger step size demonstrated faster initial learning and fewer fluctuations, suggesting reduced susceptibility to local minima.

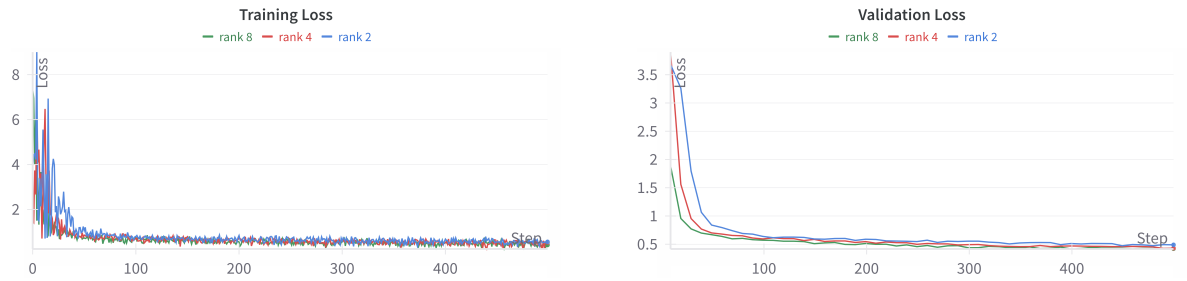
The number of FLOPS used was 2.80×10^{15}.

3.3.2 LoRA rank

The LoRA rank was then tuned for 2, 4, 8, using the best learning rate of 1×10^{-4}. The best rank was found to be 8 as shown in Table 7. and there was no sign of overfitting as observed from validation and training loss curves in Figure 7.

Table 7: Cross Entropy Losses for Varying LoRA Ranks (LR $= 1 \times 10^{-4}$), with the best LoRA rank $=$ 8 highlighted in green.

LoRA Rank	Train Loss	Validation Loss	Test Cross Entropy Loss
2	0.560	0.489	0.528
4	0.530	0.429	0.496
8	0.421	0.421	**0.453**



(a) Training Loss

(b) Validation Loss

Figure 7: Training and Validation loss for different ranks

The number of FLOPS used was 2.80×10^{15}.

3.3.3 Context length

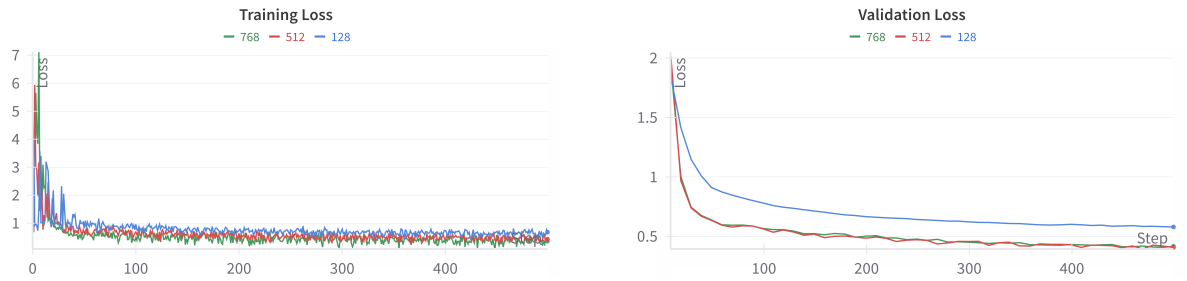
Using the optimal learning rate of 1×10^{-4} and LoRA rank 8, we evaluated the model on context lengths of 128, 512, and 768. As shown in Table 8, context lengths 512 and 768 yielded comparable validation and test performance, with 512 slightly outperforming. In contrast, a context length of 128 resulted in significantly higher test loss.

Table 8: Cross Entropy Losses for Varying Context Lengths (LR $= 1 \times 10^{-4}$, LoRA Rank $= 8$), with the best context length $= 512$ highlighted in green.

Context Length	Train Loss	Validation Loss	Test Cross Entropy Loss
128	0.680	0.580	0.628
512	0.429	0.406	**0.457**
768	0.360	0.416	0.460

Some model overfitting with context length 768 can be seen in Table **??**, where the train loss has improved from context length 512, but validation lsos has stayed the same. This can be similarly observed in Figure 8. The overfitting is likely due to longer contexts exposing the model to more tokens per sequence but fewer unique sequences, encouraging memorization over generalization.

In contrast, shorter contexts like 128 likely fail to capture full oscillatory cycles of the Lotka-Volterra dynamics, limiting the model's ability to learn long-term temporal dependencies. Training and validation losses for context length 128 plateaued earlier compared to longer contexts, suggesting the model's capacity to improve was constrained by insufficient sequence length. This shows longer contexts help attention-based models learn temporal dependencies.



(a) Training Loss

(b) Validation Loss

Figure 8: Training and Validation loss for different context lengths

The limited generalisation observed at context length 768 may stem from insufficient training diversity, with the model overfitting to repeated exposure of the same sequences. To test this, the experiments were repeated with an expanded dataset of 300 sequences (Appendix **??**). Results remained consistent, such that context lengths 512 and 768 yielded similar validation/test losses, while 768 achieved lower training loss. This supports using 512 as the best balance of generalisation and efficiency.

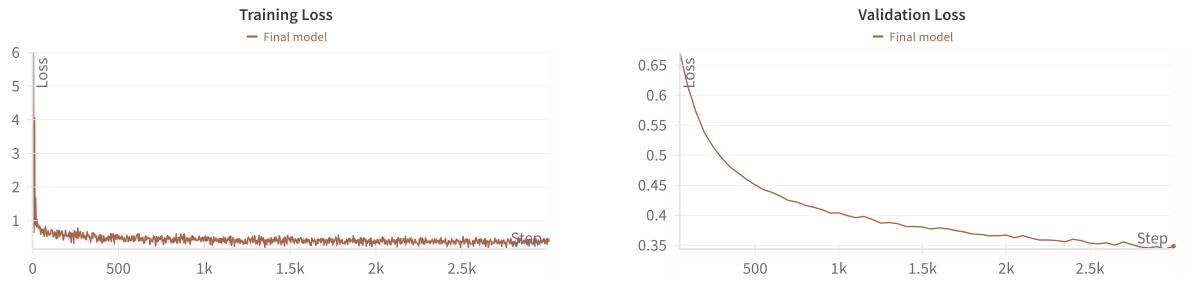
The number of FLOPS used for context length 128, 512 and 768 was 2.25×10^{14}, 9.32×10^{14} and 1.43×10^{15} respectively. The additional experiment in the appendix also costed 2.59×10^{15} FLOPS. This totals to 5.17×10^{15} FLOPS for this section.

3.4 Final Model

The final model was trained on all 1000 systems for 3000 steps using a learning rate of 1×10^{-4}, a rank of 8, and a context length of 512. While 30,000 steps was the intended upper limit, training was capped at 3000 due to resource constraints, and is justified especially since early experiments showed that validation loss plateaued around this point. Compared to the untuned model, the tuned version achieved improved cross-entropy loss (by 0.377), better MSE for both prey and predator, and improved MAPE for predator, but slightly worse MAPE for prey, as shown in Table 9. No overfitting was observed, as indicated by the training and validation curves in Figure 9.

Table 9: Performance Comparison: Default vs Tuned Model

Metric	**Default Model**	**Tuned Model**	**Change**
Cross-Entropy Loss	0.720	0.343	+0.377
Average MSE (Prey)	0.396	0.275	+0.121
Average MAPE (Prey) [%]	27.750	36.611	-8.861
Average MSE (Predator)	0.091	0.040	+0.051
Average MAPE (Predator) [%]	63.926	39.461	+24.465



(a) Training Loss

(b) Validation Loss

Figure 9: Training and Validation loss for the final tuned model for 3000 steps across all 1000 sequences.

One challenge I encountered was hallucination, where the model generated words instead of numbers when using the full dataset. This happens because large language models treat words and numbers similarly. To fix this, I added a regular expression after decoding to filter out anything that was not a properly formatted coordinate pair. The increase in hallucination may have been caused by the higher variability in the larger dataset, which made learning more difficult.

Although tuning and extended training improved overall performance by helping the model learn the prey and predator patterns more effectively, it was surprising to see MAPE for prey get worse even though MSE improved. This can be explained by the difference between the two metrics: MSE penalizes absolute errors, while MAPE penalizes relative errors. The model likely improved at predicting larger prey values, such as predicting 4 when the true value is 8, which lowers MSE. But it may have struggled more with small values, like predicting 0.2 when the true value is 0.4. Although both examples have a 50 percent error in MAPE, the absolute error is much smaller in the second case. This suggests the model focused more on reducing large absolute errors, possibly because it saw more large prey values during training.

The total FLOPs required is 5.19×10^{15}.

3.5 FLOPS

To conclude, Table 10 summarizes the total FLOPS used in all the experiments mentioned in this report. The total number of flops is below 10^{17}.

Table 10: Comparison of Qwen model variants with LoRA tuning across learning rate, rank, and context length. FLOPs are estimated per full training run.

Model Variant	**# Seq.**	**Input Len**	**Hidden Dim**	**Train Steps**	**Val Steps**	**FLOPs**
Qwen	100	512	896	500	50	$9.32 \cdot 10^{14}$
LoRA: LR tuning	100	512	896	500	50	$2.80 \cdot 10^{15}$
LoRA: Rank tuning	100	512	896	500	50	$2.80 \cdot 10^{15}$
LoRA: Context = 128	100	128	896	500	50	$2.25 \cdot 10^{14}$
LoRA: Context = 512	100	512	896	500	50	$9.32 \cdot 10^{14}$
LoRA: Context = 768	100	768	896	500	50	$1.43 \cdot 10^{15}$
LoRA: Context (300 seq)	300	768	896	500	50	$2.59 \cdot 10^{15}$
Final Model	1000	512	896	20000	2000	$5.19 \cdot 10^{15}$
Total FLOPs	–	–	–	–	–	$1.69 \cdot 10^{16}$

4 Conclusion and future work

Running these experiments with LLMs proved computationally intensive, with FLOPs analysis highlighting the Query-Key-Value projection step as particularly costly. LoRA mitigates this by reducing the number of trainable parameters during backpropagation. This was supported by the strong performance of using the LoRA-adapted Qwen model, with the best performance observed with a learning rate of 1e-4, LoRA rank of 8, and context length of 512.

However, despite these optimizations, validation loss plateaued around step 2000, indicating structural or representation limitations. One simple improvement lies in the choice of scaling factor during preprocessing, for example scaling values to the range 1 to 10 rather than 0 to 1. When scaled to 0 to 1, most numbers have leading zeros, which carry little meaningful information about the time series. Removing them makes space for more representative tokens. Furthermore, since LLMs are not inherently designed for time series forecasting, pretraining on general time series data may help the model better understand temporal and numerical relationships, rather than treating numbers as isolated tokens.

Many recent research have proposed improvements that can be made to LLM time series forecasting to better peformance. Insights from Tang et al. [4] suggest that LLMs struggle with sequences lacking clear periodicity, which echos the poor performance of the Qwen model when faced with systems with variable periodicity. To address this, Tang et al. proposed enriching inputs with external knowledge. Moving forward with this project, integrating knowledge such as the Lotka-Volterra equations could help improve forecasts.

Alternatively, adopting Prompt-as-Prefix methods as shown by Jin et al. [3], where a task is framed with a natural language description, may help provide more context for the patterns and mathematical relationships in a time series. These targeted changes to input strategy and data representation offer promising paths to improving performance time-series forecasting where computational resources are limited.

A FLOPS Operation Table

Table 1: FLOPS Accounting for Primitives

Operation	**FLOPS**
Addition/Subtraction/Negation (float OR integer)	1
Multiplication/Division/Inverse (float OR integer)	1
ReLU/Absolute Value	1
Exponentiation/Logarithm	10
Sine/Cosine/Square Root	10

Figure 10: FLOPS operation table

B FLOPS Calculation

Transformer step	**Equation**	**FLOP Expression**	**FLOPs (24 layers)**
Add positional encodings	Embedding + Positional encodings	1199×896	1.073×10^6
RMSNorm	$\text{RMS}(x) = \sqrt{\text{mean}(x^2)}$	$(1199 \times 896) + (1199 \times (896 - 1)) + 896 + 896$	5.158×10^7
	$\frac{x}{\text{RMS}(x)} \cdot w$	$(1199 \times 896) + (1199 \times 896) + (1199 \times 896 \times (2 \times 896 - 1))$	4.623×10^{10}
QKV Projection	xW_Q, xW_K, xW_V	$3 \times (1199 \times 896 \times (2 \times 896 - 1))$	1.385×10^{11}
	$+ b_Q$, b_K, b_V	$3 \times (1199 \times 896)$	7.735×10^7
Multi-headed Attention	$QK^\top$	$14 \times (1199 \times 1199 \times (2 \times 64 - 1))$	6.135×10^{10}
	$\text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$	$14 \times (1199 \times 1199 \times 10)$	4.830×10^9
	$\mathbf{V}_h \cdot \text{Softmax}\left(\frac{\mathbf{K}_h^\top \mathbf{Q}_h}{\sqrt{d_k}}\right)$	$14 \times (1199 \times 1199 \times (2 \times 64 - 1))$	6.135×10^{10}
	$\text{MhSa}[\mathbf{X}] = \mathbf{\Omega}_c \left[\text{Sa}_1[\mathbf{X}]^T, \text{Sa}_2[\mathbf{X}]^T, \ldots, \text{Sa}_H[\mathbf{X}]^T\right]^T$	$1199 \times 896 \times (2 \times 896 - 1)$	2.310×10^{10}
Residual Connection	$x + \text{Attention}(x)$	1199×896	2.578×10^7
RMSNorm	$\text{RMS}(x) = \sqrt{\text{mean}(x^2)}$	$(1199 \times 896) + (1199 \times (896 - 1)) + 896 + 896$	5.158×10^7
	$\frac{x}{\text{RMS}(x)} \cdot w$	$(1199 \times 896) + (1199 \times 896) + (1199 \times 896 \times (2 \times 896 - 1))$	4.623×10^{10}
SwiGLU	xW_{up}, xW_{gate}	$2 \times (1199 \times 896 \times (2 \times 4864 - 1))$	2.506×10^{11}
	$\text{Swish}(xW_{\text{gate}}) = x \cdot \sigma(x)$	$(1199 \times 4864 \times (10 + 1 + 1)) + (1199 \times 4864)$	7.569×10^7
	$\text{Swish}(xW_{\text{gate}}) \cdot (xW_{\text{up}})$	1199×4864	5.960×10^6
	$\text{Output} \cdot W_{\text{down}}$	$1199 \times 4864 \times (2 \times 896 - 1)$	1.254×10^{11}
Residual Connection	$x + \text{FFN}(x)$	1199×896	2.578×10^7
RMSNorm	$\text{RMS}(x) = \sqrt{\text{mean}(x^2)}$	$(1199 \times 896) + (1199 \times (896 - 1)) + 896 + 896$	5.158×10^7 (1 Layer)
	$\frac{x}{\text{RMS}(x)} \cdot w$	$(1199 \times 896) + (1199 \times 896) + (1199 \times 896 \times (2 \times 896 - 1))$	4.623×10^{10} (1 Layer)
Linear layer	$x \cdot v$	$1199 \times 151936 \times (2 \times 896 - 1)$	3.267×10^{11} (1 Layer)
Total FLOPS			1.27×10^{12}

Table 11: FLOP Breakdown for Each Component of a 24-Layer Transformer (1199 tokens, 896-dim)

C MSE and MAPE for Each Test System

System ID	MSE Prey (Untrained)	MSE Prey (Trained)	MAPE Prey (Untrained)	MAPE Prey (Trained)	MSE Predator (Untrained)	MSE Predator (Trained)	MAPE Predator (Untrained)	MAPE Predator (Trained)
783	0.175	0.251	11.714	13.738	0.032	0.011	14.659	7.958
398	1.478	0.570	38.115	24.950	0.016	0.015	67.739	65.636
99	0.845	0.199	31.770	18.543	0.339	0.247	289.209	251.226
763	1.105	1.542	37.646	57.992	0.008	0.008	52.893	68.451
583	1.784	0.423	135.318	52.215	0.256	0.356	292.873	346.929
173	0.083	0.158	12.173	16.754	0.016	0.000	56.773	9.594
944	0.260	0.230	15.714	14.283	0.001	0.002	8.202	11.578
107	1.502	0.054	34.205	5.392	0.148	0.014	29.415	8.562
52	0.339	0.439	16.817	19.325	0.125	0.033	33.458	16.427
919	0.619	1.685	27.575	45.929	0.380	0.588	83.958	106.430
584	0.045	0.049	20.511	23.206	0.009	0.013	14.244	16.010
67	0.075	0.077	64.307	72.352	0.000	0.000	33.251	18.232
723	0.013	0.009	5.466	4.674	0.011	0.003	8.248	4.693
755	0.072	0.231	23.815	43.126	0.018	0.081	11.846	22.702
608	0.029	0.023	8.015	3.768	0.002	0.001	6.111	4.458

Table 12: MSE and MAPE values for each test system in Section 3.2

D Additional experiment for 300 sequences

Context Length	Train Loss	Validation Loss	Test Cross Entropy Loss
128	0.63089	0.62106	0.607
512	0.50899	0.44294	0.432
768	0.42268	0.44312	0.431

Table 13: Cross Entropy Losses for Varying Context Lengths ($\text{LR} = 1 \times 10^{-4}, \text{LoRA Rank} = 8$)

E Generational tools remarks:

Generation tools such as ChatGPT and Claude.ai are used to debug and refine the code associated to this report. It was also used to help facilitate LaTeX formatting for figures and tables used in the report, as well as performing spellchecks.

References

[1] Alibaba Cloud. Qwen2.5-0.5b: A large language model, September 2024.

[2] Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters, 2024.

[3] Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-llm: Time series forecasting by reprogramming large language models, 2024.

[4] Hua Tang, Chong Zhang, Mingyu Jin, Qinkai Yu, Zhenting Wang, Xiaobo Jin, Yongfeng Zhang, and Mengnan Du. Time series forecasting with llms: Understanding and enhancing model capabilities, 2024.