M2: Lora Finetuning

Xueqing Xu (xx823)

Department of Physics, University of Cambridge

April 7, 2025

Introduction

This coursework explores the application of Low-Rank Adaptation (LoRA) to fine-tune the Qwen2.5-Instruct Large Language Model (LLM) for forecasting predator-prey population dynamics. Building on the observation that LLMs can function as time series forecasters without explicit training[1], we investigate how targeted fine-tuning can enhance their forecasting capabilities while maintaining efficiency.

LoRA represents a parameter-efficient fine-tuning technique that dramatically reduces the number of trainable parameters by injecting small, trainable low-rank matrices into existing model weights without modifying the original parameters. This approach is particularly valuable when working with large models under computational constraints.

The target application is forecasting the Lotka-Volterra predator-prey system, a classic ecological model that describes the dynamic interaction between two species. This system exhibits oscillatory behavior that presents a challenging forecasting task requiring understanding of non-linear dynamics and the ability to model complex interdependencies.

Methodology

Qwen2.5-Instruct model architecture

The Qwen2.5-0.5B-Instruct model implements a decoder-only transformer architecture with 494 million parameters (Table 1). With a hidden dimension of 896 across 24 transformer layers, the model balances depth and computational efficiency.

A key architectural feature is Grouped Query Attention (GQA), which employs 14 query heads but only 2 key-value heads—a 7:1 ratio that substantially reduces memory usage during inference while maintaining attention capabilities. For normalization, Qwen2.5 uses RMSNorm with $\epsilon = 10^{-6}$, which offers better training stability than traditional LayerNorm. The model implements Rotary Position Embeddings (RoPE) with a base frequency of 1,000,000.0 to encode relative positions effectively, supporting its 32K token context window. As shown in Table 1, each layer consists of self-attention and feed-forward blocks, both with pre-normalization and residual connections. The feed-forward network uses SwiGLU activation and an intermediate dimension of 4864 (approximately 5.4× the hidden size).

Key Components with Mathematical Definitions

RMSNorm (Root Mean Square Normalization): Unlike traditional LayerNorm, RMSNorm eliminates mean-centering and focuses only on variance normalization:

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 + \epsilon}} \cdot \gamma \tag{1}$$

where $\epsilon = 10^{-6}$ is a small constant for numerical stability, and γ represents trainable scale parameters.

Parameter	**Value**
Model Type	Decoder-only Transformer
Parameters	0.5 Billion
Hidden Size	896
Attention Heads	14
Head Dimension	64
Number of Layers	24
Intermediate Size (MLP)	4864
Vocabulary Size	151,936
Layer Structure (Repeated 24 times):	
Pre-Attention	RMSNorm
Attention	Multi-head attention with RoPE
	- Q, K, V projections
	- Rotary Position Embeddings
Post-Attention	Residual connection
Pre-MLP	RMSNorm
MLP	Gate and Up projections
	SwiGLU activation
	Down projection
Post-MLP	Residual connection
Final Output	RMSNorm + Linear (LM head)

Table 1: Qwen2.5-0.5B Model Architecture

SiLU (Swish) Activation: The SiLU activation function used in the feed-forward networks is defined as:

$$\text{SiLU}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}} \tag{2}$$

where $\sigma(x)$ is the sigmoid function. This activation combines properties of both ReLU and sigmoid functions.

Rotary Position Embeddings (RoPE): RoPE[2] encodes position information through rotation matrices in the complex plane:

$$\mathbf{q}_{m,i}^{\theta} = \begin{pmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix} \mathbf{q}_{m,i} \tag{3}$$

where $\mathbf{q}_{m,i}$ represents query vector components, m is the position index, and θ_i denotes frequencies constructed with base $\theta = 1,000,000.0$.

SwiGLU Feed-Forward Network: The FFN in each layer consists of three projections:

1. **Gate Projection:** $G(x) = W_g x$, where $W_g \in \mathbb{R}^{4864 \times 896}$
2. **Up Projection:** $U(x) = W_u x$, where $W_u \in \mathbb{R}^{4864 \times 896}$
3. **Down Projection:** $D(x) = W_d x$, where $W_d \in \mathbb{R}^{896 \times 4864}$

The complete FFN operation is defined as:

$$\text{FFN}(x) = D(\text{SiLU}(G(x)) \odot U(x)) \tag{4}$$

where $\odot$ represents element-wise multiplication.

Property	Value
Vocabulary Size	151,936 tokens
BOS Token ID	151,643
EOS Token ID	151,645
Word Embeddings	Tied with output layer

Table 2: Qwen2.5-0.5B-Instruct Vocabulary Details

The model implements these components with a vocabulary of 151,936 tokens and tied word embeddings between input and output layers (Table 2). This architecture is particularly suitable for parameter-efficient fine-tuning methods like LoRA, as we can target high-leverage components (query and value projections) while keeping most parameters frozen.

Lora Finetuning

Low-Rank Adaptation (LoRA) [?] represents a parameter-efficient fine-tuning approach that substantially reduces the number of trainable parameters while maintaining model performance. The key innovation lies in decomposing weight updates into low-rank matrices instead of fine-tuning the entire weight matrices.

Mathematical Formulation

In standard fine-tuning of a pre-trained model, the weight matrix $W_0 \in \mathbb{R}^{d\times k}$ is updated to $W = W_0 + \Delta W$ during training. LoRA parameterizes the update ΔW using two low-rank matrices:

$$W = W_0 + \Delta W = W_0 + BA \tag{5}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ with rank $r \ll \min(d, k)$. During the forward pass, for input x, the output is computed as:

$$h = W_0x + \Delta W x = W_0x + BAx \tag{6}$$

The weight update is typically scaled during training by α/r, where α is a constant hyperparameter:

$$h = W_0x + \frac{\alpha}{r}BAx \tag{7}$$

Application to Qwen2.5

For the Qwen2.5-0.5B-Instruct model, we apply LoRA specifically to the query and value projection matrices in the multi-head attention mechanism. These matrices are chosen because:

- Query projections ($W_q \in \mathbb{R}^{896\times 896}$) directly influence the model's ability to focus on relevant context
- Value projections ($W_v \in \mathbb{R}^{896\times 128}$) affect how the model represents information for aggregation
- Modifying these components provides substantial adaptation power while minimizing parameter count

The LoRA matrices are injected into the model architecture, allowing for efficient adaptation without altering the original weights. The choice of low-rank matrices enables us to maintain a small number of trainable parameters while still achieving significant performance improvements.

Training Dynamics

During training, only the LoRA parameters (A and B) are updated while W_0 remains frozen. This approach offers several advantages:

- **Memory efficiency**: Only the gradients for LoRA parameters need to be stored
- **Composability**: Multiple task-specific LoRA modules can be trained and swapped without changing the base model
- **Preservation of general knowledge**: The original pre-trained weights remain intact

In our implementation, we utilized the transposed form of the LoRA matrices during the forward pass, computing the update as `self.lora_A.T @ self.lora_B.T`. This approach is mathematically equivalent to the original formulation but offers implementation advantages within the PyTorch framework, particularly for gradient computation efficiency.

Additionally, we deviated from the standard LoRA approach by making the language model head trainable as well. This decision was motivated by the need for adaptation specifically at the vocabulary distribution level, which is crucial for accurate numerical predictions. The LM head contains $896 \times 151,936 = 136,134,656$ parameters, significantly increasing our trainable parameter count, but targeting this layer directly improves the model's ability to produce precise numerical outputs in text form.

The total trainable parameter count thus becomes:

$$\begin{aligned} \text{LoRA parameters} + \text{LM head parameters} &= 344,064 + 136,134,656 \quad (8) \\ &= 136,478,720 \quad (9) \end{aligned}$$

This still represents a significant reduction (approximately 72.4%) compared to full fine-tuning of all 494 million parameters.

Predator-Prey Dynamics

The Lotka-Volterra model represents a classic mathematical framework for studying predator-prey interactions in ecological systems. It describes the oscillatory dynamics between two populations: a prey species (x) that has abundant food and can reproduce exponentially, and a predator species (y) that relies on consuming the prey for survival.

Mathematical Model

The standard Lotka-Volterra equations are defined as:

$$\frac{dx}{dt} = \alpha x - \beta xy \quad (10)$$

$$\frac{dy}{dt} = \delta xy - \gamma y \quad (11)$$

In these equations, $x(t)$ and $y(t)$ represent the prey and predator populations at time t, respectively. The parameter α denotes the prey's natural growth rate, while β represents the predation rate, capturing the effect of predators on prey population decline. For predators, δ quantifies their reproduction rate per prey consumed, and γ represents their natural mortality rate in the absence of prey.

Data Analysis

Our exploratory analysis of the Lotka-Volterra dataset revealed important statistical properties across 1,000 trajectories. Each trajectory contained 100 time points with consistent sampling intervals. Data quality checks verified the absence of duplicates and negative values.

Using K-means clustering on trajectory features (including period, amplitude, and phase relationships), we identified four distinct dynamic patterns in the dataset as displayed in Figure 1:

- **Classic oscillatory dynamics (14.6%)**: Characterized by sustained oscillations with relatively balanced prey and predator populations, showing clear cyclic interactions.
- **Prey-dominant systems (69.3%)**: Systems where prey populations maintain higher relative values with predator populations showing dampened oscillations.
- **Equilibrium systems (14.9%)**: Trajectories that stabilize after initial oscillations, reaching a relatively steady state.
- **Predator collapse systems (1.2%)**: A small subset where predator populations drop dramatically, allowing prey populations to grow significantly without constraint.

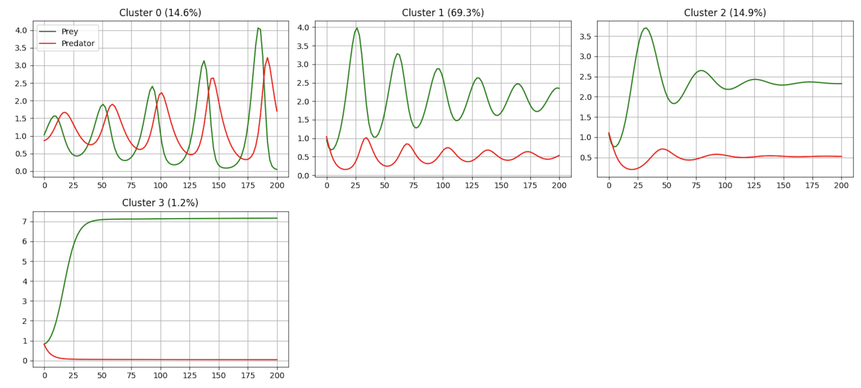

Figure 1: **Cluster Analysis**. Four distinct clusters identified in the Lotka-Volterra dataset through K-means clustering, showing different regime types: classic oscillatory dynamics (14.6%), prey-dominant systems (69.3%), equilibrium systems (14.9%), and predator collapse systems (1.2%).

The clustering revealed that the majority of systems (69.3%) fall into prey-dominant dynamics, suggesting parameter combinations that favor prey survival across much of the parameter space explored.

Figure 2 showcases individual trajectories representing diverse dynamic behaviors within the dataset. These examples highlight the variability in amplitudes, frequencies, and phase relationships that our forecasting model must learn to predict accurately.

Through peak detection analysis, we calculated oscillation periods for both prey and predator populations, finding average periods of approximately 20-25 time units for prey and 20-30 time units for predators, as shown in Figure 3.

Further analysis of the phase relationships between prey and predator oscillations showed that predator peaks typically lag behind prey peaks by about 0.2 time units (20% of the cycle period), as illustrated in Figure 3. This characteristic phase difference reflects the biological reality that predator populations grow in response to increasing prey availability, then decline as prey becomes scarce.

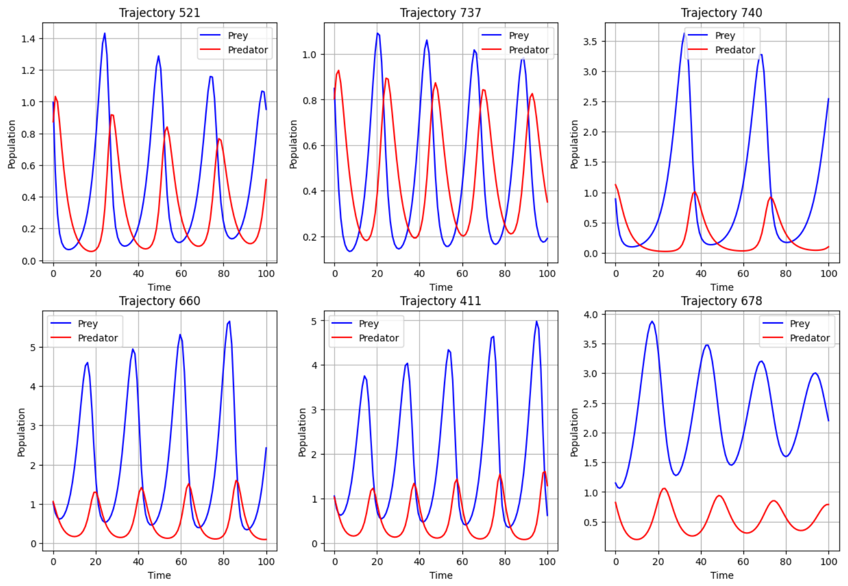

Figure 2: **Individual Trajectories**. Sample trajectories of prey and predator populations in the Lotka-Volterra dataset, illustrating the oscillatory dynamics characteristic of the model.

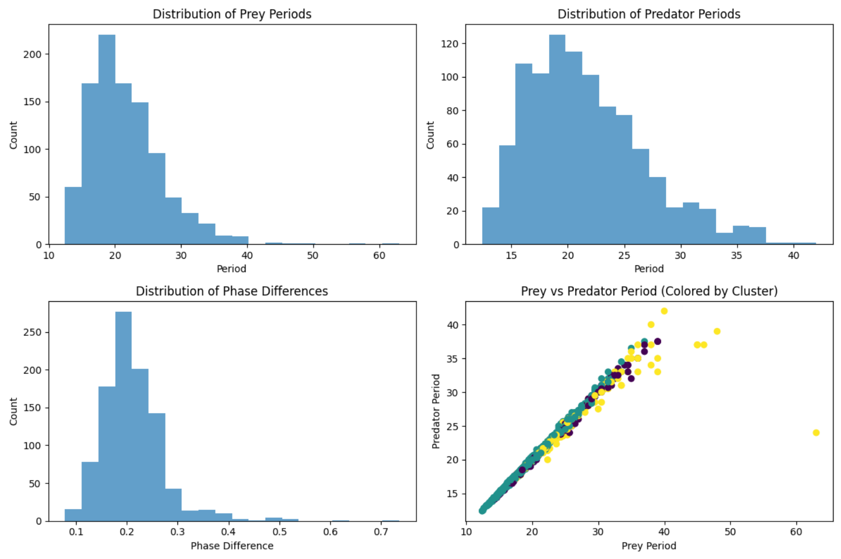

Figure 3: **Statistical Distributions**. Distribution of oscillation periods for prey and predator populations in the Lotka-Volterra dataset, highlighting the variability in dynamics across different parameter regimes.

Figure 4 reveals typical system behavior with prey populations maintaining higher values (averaging ≈ 1.7 units) compared to predator populations (averaging ≈ 0.6 units) after initial transient dynamics.

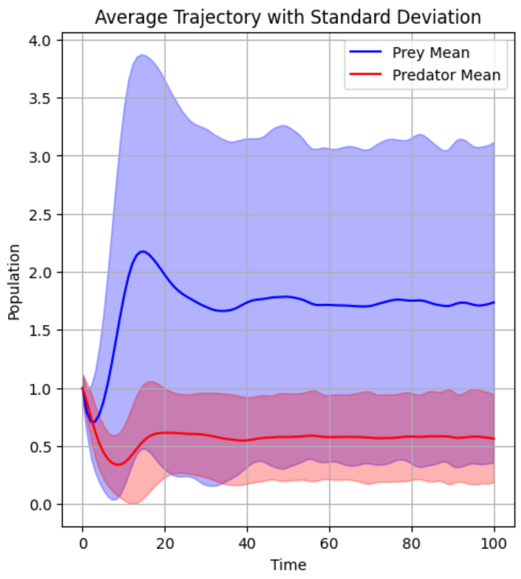

Figure 4: **Average Trajectories**. Average population trajectories with uncertainty: Mean prey (blue) and predator (red) population trajectories over time with shaded regions representing standard deviation.

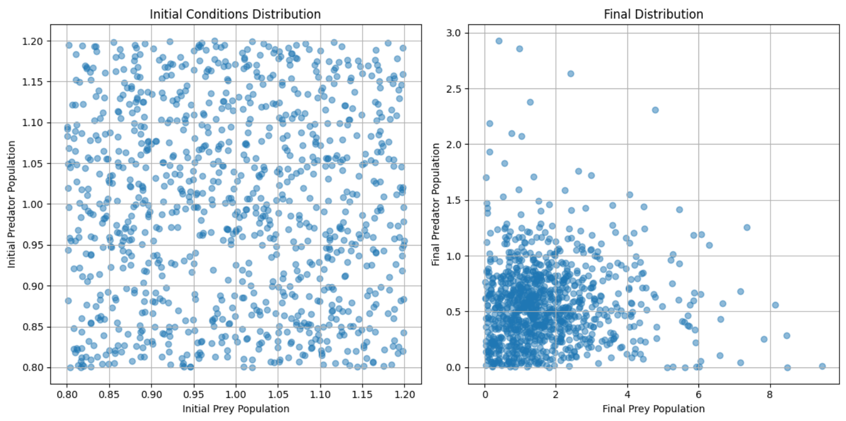

Figure 5: **Initial vs Final Distribution**. Comparison between initial conditions (left) and final states (right) of the predator-prey systems after simulation.

A critical insight into the system's sensitivity to initial conditions is provided by Figure 5, which compares the initial and final states of the simulations. Despite starting from a tightly constrained parameter space (prey and predator populations both between 0.8-1.2 units), the final states display remarkable divergence. This sensitivity is a hallmark of nonlinear dynamical systems, where small differences in initial conditions can lead to vastly different outcomes over time. The final distribution shows a concentration of points at lower predator values (0.1-1.0) with prey populations spread across a much wider range (0-8), indicating that many trajectories evolve toward states where prey populations are less constrained by predator influence. This divergence from similar starting points underscores the challenge in long-term forecasting of such systems, as minor variations in inputs can amplify into major differences in outcomes.

For the current study with our limited dataset of 1,000 trajectories, we implemented a simple random split into training (70%), validation (15%), and testing (15%) sets. Given the time constraints and dataset size, this approach provided a practical balance between model training and evaluation needs. However, the observed dynamics suggest several potential improvements for future work with larger datasets.

Future Improvements for Data Handling

The observed sensitivity to initial conditions and the diverse range of dynamic behaviors have important implications for more sophisticated approaches to training, validation, and testing splits in future work. With a larger dataset, a stratified sampling approach based on the four identified clusters would better maintain the distribution of dynamic behaviors across all splits.

Such stratification would be particularly valuable because the complexity and diversity of the dynamics require the model to learn from examples across all behavioral regimes. For example, the rare "predator collapse" trajectories (1.2% of cases) should be proportionally represented in all splits to prevent the model from treating such cases as anomalies during evaluation. Similarly, the phase relationships and oscillation characteristics vary systematically across clusters, making it essential that the model train on and be evaluated against the full spectrum of possible dynamics.

Forecasting Challenge

The Lotka-Volterra system presents an ideal test case for evaluating language model forecasting capabilities due to its nonlinear dynamics that challenge simple forecasting methods, its requirement for simultaneous prediction of interdependent variables, the importance of long-range historical patterns for accurate prediction, its inherent periodicity that aligns with language models' pattern recognition strengths, and its well-understood mathematical properties that provide a strong basis for evaluation. Our forecasting task requires the model to predict future values ($x_{t+1}...t+n, y_{t+1}...t+n$) given past values ($x_{t-m}...t, y_{t-m}...t$), effectively testing the model's ability to identify and extrapolate complex numerical patterns.

LLMTIME Preprocessing Approach

FLOPs Calculation

Implementation

Hardware

Data preprocessing implementation

- Tokenization approach
- Numeric to text conversion details
- Scaling and precision considerations

LoRA implementation details

include hardware

- Target modules selection (Q and V projections)
- LoRA hyperparameters

Model training setup

- Optimization approach
- Learning rate and batch size considerations

- FLOP tracking

Baseline Evaluation

- Untrained model performance analysis
- Tokenization examples and results
- FLOP accounting for the Qwen2.5 model operations
- Analysis of accuracy metrics (MSE, MAE) for the baseline

LoRA Experiments

Hyperparameter search experiment

Analysis of results

Final Model Performance

- Detailed analysis of the best model configuration
- Comparative evaluation against baseline
- Visualization of forecasting performance
- Error analysis for prey and predator populations

Discussion

- Analysis of the trade-offs between computational cost and accuracy
- Impact of different hyperparameters on performance
- Strengths and limitations of the approach
- Recommendations for time-series fine-tuning under tight compute budgets

Conclusion

- Summary of key findings
- Suggestions for future improvements
- Final FLOP accounting table

Appendix

Detailed Flop Calculation

References

[1] Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36:19622–19635, 2023.

[2] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.