M2 Coursework Report

Keying Song (ks2146)

4th April 2025

Word Count: 2,927 words

Abstract

This coursework explores Low-Rank Adaptation (LoRA) for fine-tuning the Qwen2.5-Instruct Large Language Model (LLM) to predict multivariate time series data under tight compute budgets. Firstly, a baseline forecasting experiment on the untrained LLM, using the LLMTIME preprocessing scheme [1] to preprocess data, was performed and evaluated. Subsequently, LoRA was integrated into the Qwen2.5-Instruct architecture by applying LoRALinear wrappers to the query and value projection layers in the attention blocks, followed by training the 0.5B parameter model with LoRA and then comparing the evaluated performance to the untrained model. After that, this work also investigated the best hyperparameter configuration for LoRA fine-tuning, as the result of searching over the learning rate, the LoRA rank and the context length. Exploiting the best performing set of hyperparameters, the final model was trained and evaluated. Finally, a FLOPS calculator was created to track the FLOPS used in all experiments, ensured that they did not exceed the expected compute constraints.

Contents

1) Introduction

This essay reports on the implementation and evaluation of the Qwen2.5 model fine-tuning with LoRA. In section 2, I will provide the details of FLOPS analysis through the forward pass process of Qwen2.5, which will use symbolic representations rather than numerical calculation. In section 3, the implementation of baseline experiment with unfine-tuned model on the preprocessed prey-predator dataset will be presented, followed by the FLOPS evaluation using the experimental parameters. Then in section 4, the training experiments of fine-tuned model with LoRA will be displayed, along with the performance of final model and also the numerical evaluation of associated FLOPS. Finally, section 5 will showcase a table summarising the FLOPS used for every experiment, as well as the recommendations for high efficient fine-tuning methods for time-series.

2) FLOPS Analysis

In order not to exceed the compute limitation, a maximum of 10^{17} floating point operations (FLOPS), this section provides a detailed breakdown of FLOPS calculation across different components of the Qwen2.5-Instruct model, following the accounting rules provided.

2.1 Model Architecture

The Qwen2.5-Instruct model preserves the transformer-based decoder architecture [2] while making some improvements from the standard version. Specifically, it consists of several key components: RMSNorm instead of LayerNorm, Grouped Query Attention (GQA), Rotary Positional Embeddings (RoPE), SwiGLU activation function and so on [3].

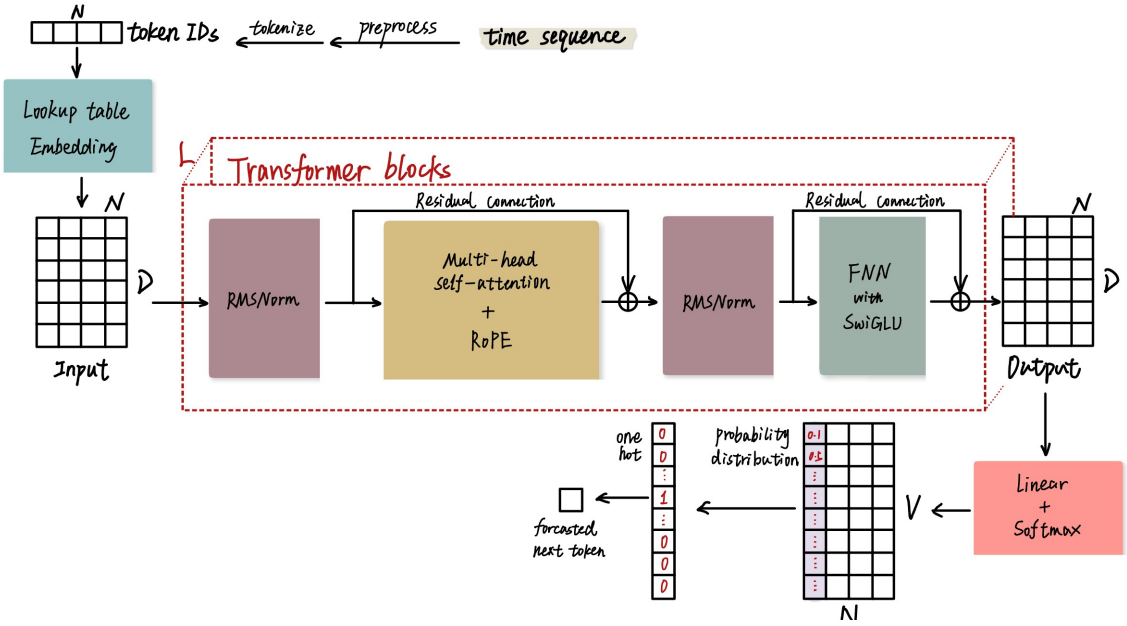


Figure 1: Very simplified sketch of the model structure and time series forecasting

Ignoring FLOPS associated with array indexing, it is assumed that the lookup table-based token embedding layer does not consume FLOPS.

2.2 Transformer Blocks

I use the standard multi-head attention mechanism to estimate the FLOPS of the GQA method in realistic scenarios, taking the number of query heads in GQA as the number of heads in the standard multi-head approach, (which is 14 for 0.5B model [3]). For clarity in presenting the estimation process of FLOPS, the notation conventions given in Table 1 are assumed, the values are from the configuration file of the model [4] and my following experiments (the context length of 499 for inference is equivalent to 50 time steps in the prey-predator dataset).

Table 1: Notation Conventions

Notation	Description	Value
N	Context Length	[128, 512, 768] for training 499 for inference
D	Hidden dimension size	896
H	Number of attention heads	14
d	Hidden dimension size of each head ($D = d \times H$)	64
L	Number of hidden layers	24
V	Vocabulary size	151936

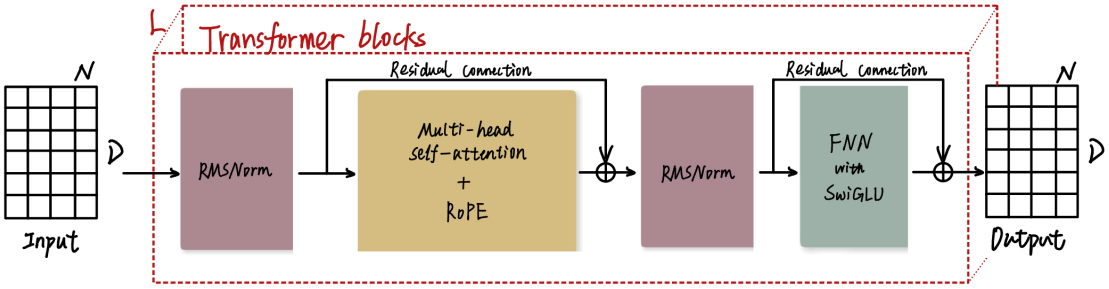


Figure 2: Very simplified sketch of the transformer layer structure

2.2.1 Rotary Position Embedding (RoPE)

Unlike other position encoding methods, RoPE is integrated into the self-attention layer in the transformer block. After the $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{d \times N}$ are obtained, RoPE multiplies a matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ with Q and K before the attention scores are calculated using the new Q', K' [5]:

$$\begin{aligned} \mathbf{Q}' &= \boldsymbol{R}\mathbf{Q} \\ \mathbf{K}' &= \boldsymbol{R}\mathbf{K} \end{aligned} \tag{2.1}$$

Where $\boldsymbol{R}$ is concatenated by d/2 2×2 rotation matrices:

$$\boldsymbol{R}^{d}_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{2.2}$$

Ignoring FLOPS associated with computing the positional encodings, the FLOPS for adding the positional information to the token embeddings is:

$$\begin{aligned} \text{FLOPS}_{\text{RoPE/head}} &= 2 \times [\text{Matrix Multiplication of } (d, d) \times (d, N)] \\ &= 2 \times [d \times N \times (2d - 1)] \\ &= 4d^2N - 2dN \end{aligned} \tag{2.3}$$

Considering the multi-head and applying $d = D/H$:

$$\begin{aligned} \text{FLOPS}_{\text{RoPE}} &= H \times \text{FLOPS}_{\text{RoPE/head}} \\ &= \frac{4D^2N}{H} - 2DN \end{aligned} \tag{2.4}$$

2.2.2 RMSNorm

Root Mean Square Normalisation (RMSNorm) is used instead of LayerNorm in the Qwen2.5 architecture, which is defined in [6]:

$$\text{RMSNorm}(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\sqrt{\|\boldsymbol{x}\|_2^2/d + \epsilon}}, \text{ where } \epsilon > 0 \tag{2.5}$$

So the FLOPS for each RMSNorm can be broken down as:

- Compute the root mean square of the elements from N input vectors, each of length D.

$$\begin{aligned} \text{FLOPS}_{\text{RMS}} &= [\text{D Square} + \text{ D-1 Addition} + 1 \text{ Division} + 1 \text{ Square Root}] \times N \\ &= [(D-1) + D + 1 + 10] \times N \end{aligned} \tag{2.6}$$

- Divide by RMS to normalise these N vectors (D divisions each).

$$\text{FLOPS}_{\text{Norm}} = D \times N \tag{2.7}$$

Summing the above two items yields the FLOPS for each RMSNorm layer. As each transformer layer contains two RMSNorm layers, the total FLOPS consumed by all RMSNorm layers within each transformer layer is as follows:

$$\text{FLOPS}_{\text{RMSNorm}} = 2 \times (3DN + 10N) \tag{2.8}$$

2.2.3 Self-Attention Mechanism

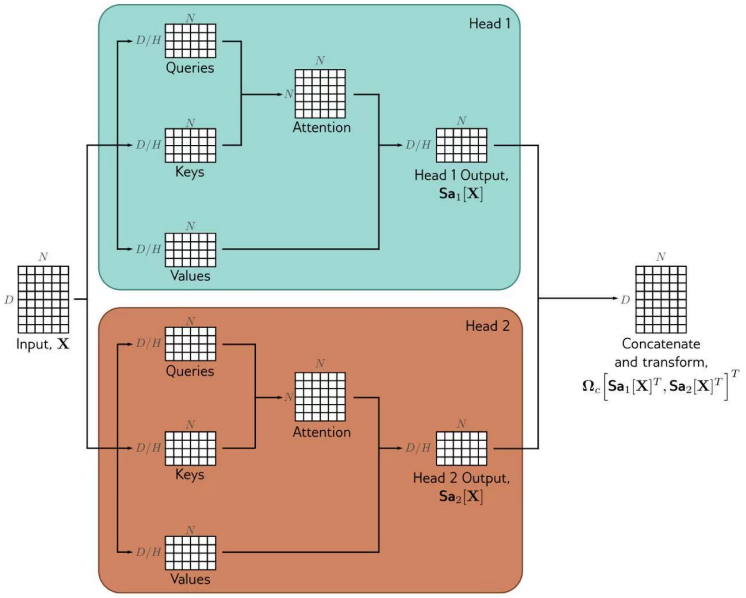


Figure 3: Sketch of Multi-head self-attention, from [7]

The 2-head self-attention is depicted in figure 3 from [7]. The FLOPS for **each head** can be broken down as:

- Calculate the $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d \times N}$ from the linear projections of input matrix $\mathbf{X} \in \mathbb{R}^{d \times N}$ with the bias:

$$\begin{aligned} \mathbf{V}[\mathbf{X}] &= \boldsymbol{\beta}_v \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_v \mathbf{X} \\ \mathbf{Q}[\mathbf{X}] &= \boldsymbol{\beta}_q \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_q \mathbf{X} \\ \mathbf{K}[\mathbf{X}] &= \boldsymbol{\beta}_k \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_k \mathbf{X} \end{aligned} \tag{2.9}$$

  Where $\boldsymbol{\Omega} \in \mathbb{R}^{d \times D},\ \boldsymbol{\beta}\mathbf{1}^{\mathbf{T}} \in \mathbb{R}^{d \times N}$.

  Hence, the FLOPS here is:

$$\begin{aligned} \text{FLOPS}_{\text{QKV}} &= [(d \times N) \text{ Additions} + \text{Matrix Multiplication of } (d, D) \times (D, N)] \times 3 \\ &= [d \times N + d \times N \times (2D - 1)] \times 3 \\ &= 6DdN \end{aligned} \tag{2.10}$$

- Compute the attention scores $M = \frac{\mathbf{K}^T \mathbf{Q}}{\sqrt{\mathbf{d}}} \in \mathbb{R}^{N \times N}$:

$$\begin{aligned} \text{FLOPS}_{\text{M}} &= (N \times N) \text{ Divisions} + \text{Matrix Multiplication of } (N, d) \times (d, N) + \text{only one square root} \\ &= N^2 \times 1 + N^2 \times (2d - 1) + 10 \\ &= 2dN^2 + 10 \end{aligned} \tag{2.11}$$

- Yield attention matrix $A = Softmax(M) \in \mathbb{R}^{N \times N}$:

  For each elements in $\mathbf{A}$, $A_{i,j} = \frac{exp(M_{ij})}{\Sigma_{j'=1}^{N} exp(M_{ij'})}$. Since the elements in each row share the same denominator, it is sufficient to compute it only once per row: For i^{th} row:

$$\begin{aligned} \text{FLOPS}_i &= N \text{ Divisions} + N \text{ Exponentiations} + (N - 1) \text{ Additions} + N \text{ Exponentiations} \\ &= N \times 1 + N \times 10 + (N - 1) \times 1 + N \times 10 \\ &= 22N - 1 \end{aligned} \tag{2.12}$$

  Where the first two terms are for N numerators in this row while the last two terms are for the shared denominator.

  There are N rows in matrix A so that the FLOPS for Softmax is:

$$\text{FLOPS}_{Softmax} = 22N^2 - N \tag{2.13}$$

- Calculate the output matrix $Sa = V \cdot A \in \mathbb{R}^{d \times N}$:

$$\begin{aligned} \text{FLOPS}_{\text{Sa}} &= \text{Matrix Multiplication of } (d, N) \times (N, N) \\ &= d \times N \times (2N - 1) \\ &= 2dN^2 - dN \end{aligned} \tag{2.14}$$

Summing the above four items yields the FLOPS for each head in one transformer layer:

$$\text{FLOPS}_{\text{one head}} = 6DdN + 2dN^2 + 10 + 22N^2 - N + 2dN^2 - dN \tag{2.15}$$

Applying $D = d \times H$, the FLOPS used by the total multi-heads self-attention mechanism can be obtained:

$$\begin{aligned} \text{FLOPS}_{\text{self-attention}} &= H \times \text{FLOPS}_{\text{one head}} \\ &= 6D^2N + 2DN^2 + 10H + 22HN^2 - HN + 2DN^2 - DN \\ &= 6D^2N + 4DN^2 + (22N^2 - N + 10)H - DN \end{aligned} \tag{2.16}$$

2.2.4 FNN with SwiGLU

Qwen2.5 uses SwiGLU activation in the feed-forward networks. In fact, they extended their FNN to MoE architecture by replacing standard FFN layers with MoE layers [3]. However, I'm still assuming it is a traditional two-layer FNN structure with a SwiGLU activation layer and a linear layer:

$$\overrightarrow{FFN(\vec{x})} = (W_2 \cdot (SwiGLU(W_1 \cdot \vec{x} + \vec{\beta_1}))) + \vec{\beta_2} \tag{2.17}$$

Where $\vec{x}$ is the D-dimensional vector that forms the input matrix $\mathbf{X} \in \mathbb{R}^{D \times N}$, with a total of N. $W_1 \in \mathbb{R}^{4D \times D}, W_2 \in \mathbb{R}^{D \times 4D}$. Therefore, for each $\vec{x}$, the FLOPS calculation can be divided into three parts:

- $\vec{\alpha} = W_1\vec{x} + \vec{\beta_1}$.

$$\begin{aligned} \text{FLOPS}_{\text{linear1}} &= \text{Matrix Multiplication of } (4D, D) \times (D, 1) + 4D \text{ Additions} \\ &= 4D \times 1 \times (2D - 1) + 4D \times 1 \\ &= 8D^2 \end{aligned} \tag{2.18}$$

- $SwiGLU(\vec{\alpha}) = \vec{\alpha_1} \odot \sigma(\vec{\alpha_2})$.

  Where $\vec{\alpha_{1,2}}$ are the gating results of $\vec{\alpha} \in \mathbb{R}^{4D}$, and both have the shape 2D.

$$\begin{aligned} \text{FLOPS}_{\text{SwiGLU}} &= 2D \times (1 + 10 + 1 + 1) + 2D \times 1 \\ &= 28D \end{aligned} \tag{2.19}$$

  The first term above is from sigmoid function while the last term is from Hadamard product of two vectors of shape 2D.

- $\vec{y} = W_2 \cdot SwiGLU + \vec{\beta_2}$.

$$\begin{aligned} \text{FLOPS}_{\text{linear2}} &= \text{Matrix Multiplication of } (D, 4D) \times (4D, 1) + D \text{ Addition} \\ &= D \times 1 \times (8D - 1) + D \times 1 \\ &= 8D^2 \end{aligned} \tag{2.20}$$

Considering a total of N vectors, the whole FLOPS used by FNN is:

$$\begin{aligned} \text{FLOPS}_{\text{FNN}} &= (28D + 8D^2 + 8D^2) \times N \\ &= 28DN + 16D^2N \end{aligned} \tag{2.21}$$

2.2.5 Full Transformer Layer

Combining the components above, the FLOPS for a single transformer layer are:

$$\begin{aligned} \text{FLOPS}_{\text{layer}} &= \text{RoPE} + \text{RMSNorm} + \text{Attention} + \text{FNN} \\ &= \frac{4D^2N}{H} + (22N^2 - N + 10)H + 22D^2N + 4DN^2 + 31DN + 20N \end{aligned} \tag{2.22}$$

2.3 Output Projection

- $logits = W \cdot Y + \beta$, where $logits,\ \beta \in \mathbb{R}^{V \times N}$, $W \in \mathbb{R}^{V \times D}$, $Y \in \mathbb{R}^{D \times V}$.

$$\begin{aligned} \text{FLOPS}_{\text{forward}} &= \text{Matrix Multiplication of } (V, D) \times (D, N)\ + VN \text{ Additions} \\ &= 2DVN \end{aligned} \tag{2.23}$$

- Yield probability matrix $P = Softmax(logits) \in \mathbb{R}^{V \times N}$:

  For each elements in $\mathbf{P}$, $P_{i,j} = \frac{exp(l_{ij})}{\Sigma_{j'=1}^{N} exp(l_{ij'})}$. For i^{th} row:

$$\begin{aligned} \text{FLOPS}_i &= N \text{ Divisions} + N \text{ Exponentiations} + (N - 1) \text{ Additions} + N \text{ Exponentiations} \\ &= N \times 1 + N \times 10 + (N - 1) \times 1 + N \times 10 \\ &= 22N - 1 \end{aligned} \tag{2.24}$$

There are V rows in matrix P so that the FLOPS for Softmax is:

$$\text{FLOPS}_{Softmax} = 22VN - V \tag{2.25}$$

So that the FLOPS for the output layer is:

$$\text{FLOPS}_{Softmax} = (2D + 22)VN - V \tag{2.26}$$

2.4 Full Model Forward Pass

For the entire model with L transformer layers, the FLOPS for one forward pass is:

$$\text{FLOPS}_{\text{forward}} = L[\frac{4D^2N}{H} + (22N^2 - N + 10)H + 22D^2N + 4DN^2 + 31DN + 20N] + (2D + 22)VN - V \tag{2.27}$$

3) Baseline

3.1 Preprocessing Implementation

3.1.1 Methodology

Following the LLMTIME preprocessing scheme[1], the first thing is to scale and round the raw values in the dataset. In order to scale them to range 0-10 with a constant α, I explored the dataset (in `explore_dataset.ipynb`) and printed the maximum value, which is around 13, so that the α was chosen to be 10.

The rounding precision was set to two decimal places to balance between redundancy caused by excessive precision in token sequences and potential information loss resulting from insufficient precision. After the formatting process based on the extended convention for multivariate data, a string was generated, in which variables at the same time step are separated by commas while different time steps are separated by semicolons.

3.1.2 Results

Figure 4 below shows the preprocessed and tokenized results of two example series from `lotka_volterra_data.h5` dataset. For clarity of presentation, only the first ten time points from each of the two sequences are illustrated here. (See `notebooks/run_baseline.ipynb` for more details)

```
------------------------------------ Preprocessed examples ------------------------------------
Example 1:
0.11,0.09;0.10,0.08;0.10,0.06;0.11,0.05;0.13,0.05;0.15,0.04;0.18,0.05;0.21,0.05;0.23,0.06;0.23,0.07

Example 2:
0.11,0.09;0.08,0.08;0.06,0.07;0.05,0.06;0.05,0.04;0.05,0.04;0.06,0.03;0.07,0.02;0.09,0.02;0.11,0.02

------------------------------------ Tokenized examples ------------------------------------
Example 1:
tensor([15, 13, 16, 16, 11, 15, 13, 15, 24, 26, 15, 13, 16, 15, 11, 15, 13, 15,
        23, 26, 15, 13, 16, 15, 11, 15, 13, 15, 21, 26, 15, 13, 16, 16, 11, 15,
        13, 15, 20, 26, 15, 13, 16, 18, 11, 15, 13, 15, 20, 26, 15, 13, 16, 20,
        11, 15, 13, 15, 19, 26, 15, 13, 16, 23, 11, 15, 13, 15, 20, 26, 15, 13,
        17, 16, 11, 15, 13, 15, 20, 26, 15, 13, 17, 18, 11, 15, 13, 15, 21, 26,
        15, 13, 17, 18, 11, 15, 13, 15, 22])

Example 2:
tensor([15, 13, 16, 16, 11, 15, 13, 15, 24, 26, 15, 13, 15, 23, 11, 15, 13, 15,
        23, 26, 15, 13, 15, 21, 11, 15, 13, 15, 22, 26, 15, 13, 15, 20, 11, 15,
        13, 15, 21, 26, 15, 13, 15, 20, 11, 15, 13, 15, 19, 26, 15, 13, 15, 20,
        11, 15, 13, 15, 19, 26, 15, 13, 15, 21, 11, 15, 13, 15, 18, 26, 15, 13,
        15, 22, 11, 15, 13, 15, 17, 26, 15, 13, 15, 24, 11, 15, 13, 15, 17, 26,
        15, 13, 16, 16, 11, 15, 13, 15, 17])
```

Figure 4: The preprocessed and tokenized results of two examples

3.2 Untrained Model Evaluation

3.2.1 Methodology

Firstly, I observed and plotted the period length distribution of the dataset in the second part of `explore_dataset.ipynb`. Using autocorrelation to estimate the period, the result shows that the mean cycle time for these 1000 systems is 40.18 while the maximum is 82.83, which can guide the choice of context length when making inference.

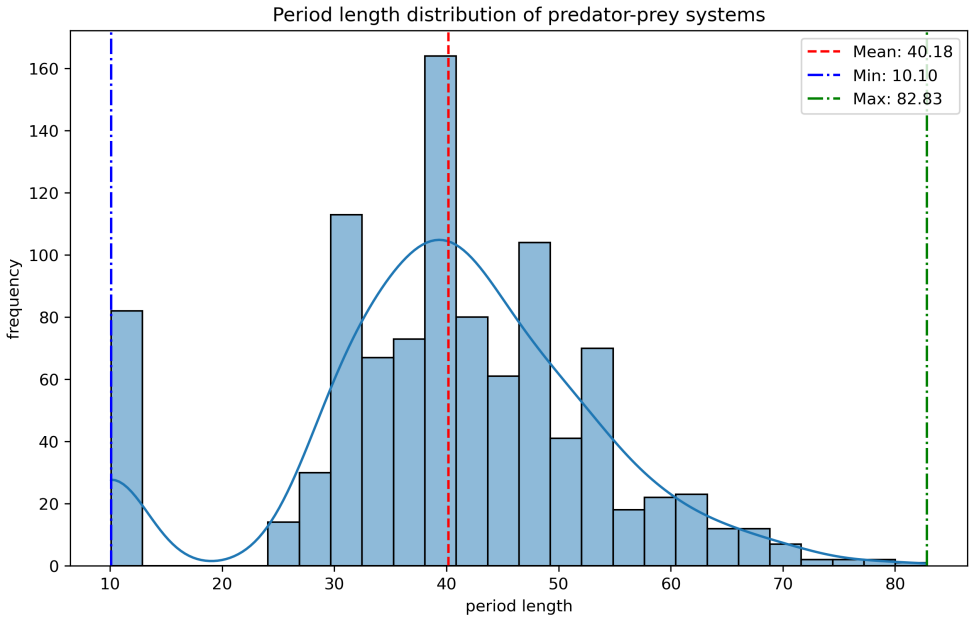


Figure 5: The period distribution for the 1000 prey-predator sequences

To evaluate the forecasting capability of the untrained Qwen2.5-Instruct model, I selected the first 50 time steps from each systems as context, ensuring coverage of at least one full cycle of most series. Then, a total of 99 forward pass predictions were executed to generate the next 10 time steps (99 tokens). To optimise computational efficiency, the forecasting performance was evaluated by computing metrics across 10 distinct trajectories, each producing a single prediction, rather than implementing the resampling method. The parameters used in forecasting and evaluation are list in table 2.

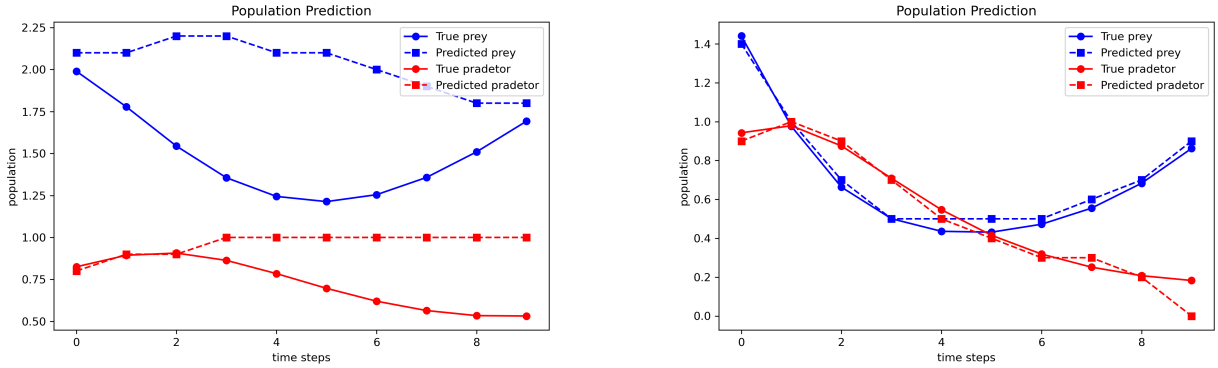
Table 2: Parameters of model evaluation

context length	target length	number of samples
50 times steps	10 times steps	10 systems

3.2.2 Results

The results presented in figure 6 and table 3 provide a clear indication of the untrained Qwen2.5-Instruct model's limited ability of prediction in this multivariate dataset. Despite having a reasonably low overall MSE of 0.2747 and MAE of 0.3066, the model demonstrated a relatively modest coefficient of determination ($R^2 = 0.2463$), suggesting it can only explain a small part of the underlying variance in the data. Notably, the model's accuracy differs for each variable: the MSE for the predator predictions is comparatively low (0.1279) while the MSE for the prey is much higher (0.4215), pointing towards inconsistencies in the model's capacity to capture different variant's behaviour.

Actually, the weak performance of the untrained model is anticipated, as the tricky multivariate dataset, even worse, may have complex interdependencies between the variants (e.g., trophic interactions between predator and prey). Without parameter updates informed by training data, the model can not effectively learn correlations or underlying system dynamics, which inevitably results in the unstable performance.



(a) Example 1

(b) Example 2

Figure 6: Predicted sequences generated by the untrained model for two example systems, with context length set to 50 (timewise) and target length set to 10 (timewise).

Table 3: Metrics for untrained model

MSE	MAE	R^2	MSE For Prey	MSE For Predator
0.2747	0.3066	0.2463	0.4215	0.1279

3.3 Baseline Budget

3.3.1 Methodology

Based on the results in section 2, I wrote a Python script `flops.py` which contains a function for FLOPS calculation. This function not only includes the hyperparameters specified previously in table 1 but also has an additional Boolean parameter to toggle between training and inference modes. Moreover, it incorporates training-specific parameters such as batch size and the number of training steps, as well as an inference-specific parameter, inference length, which is token-wise and indicates the number of forward pass when forecasting the target tokens. The core code is as below:

```
    # one transformer layer
    flops_transformer = flops_rope + flops_rmsnorm + flops_self_attention + flops_fnn

    # output projection
    flops_output = (2 * D + 22) * V * N - V

    # full model forward pass (only one time)
    flops= L * flops_transformer + flops_output

    if not inference:
        flops = 3 * flops * B * nsteps    # training mode, add backward pass, batch size and nsteps
    else:
        flops = flops * infer_length      # inference mode, add forcasted length

    return flops
```

Actually, employing the LoRA method for fine-tuning results in a slight increase in FLOPS during the forward pass due to the computation of an additional low-rank matrices (see Section 4). However, this increment is much smaller than the reduction caused by LoRA during the backpropagation. Thus, it is omitted from the present analysis.

3.3.2 Results

In the notebook `run_flops_calculation.ipynb`, the model parameters for Qwen2.5 (0.5B) were initially obtained from the open source configuration file. The symbolic conventions for these parameters are summarised in table 1. Then the compute budgets for every experiments were evaluated before the implementations, which helped with designing the parameters for subsequent experiments.

For baseline inference, I set the context length `N = 499` and the `infer_length=99`, which are equivalent to 50 and 10 time steps respectively. Activating inference mode, the FLOPS for the baseline experiment was calculated to be approximately 3.7×10^{13}.

Baseline Budget

```python
from src.flops import calculate_flops
baseline_flops = calculate_flops(499, H, D, L, B=1, inference=True, infer_length=99)
print(f"{baseline_flops:e}")
```

Python

```
3.716343e+13
```

Figure 7: The FLOPS for baseline inference.

4) LoRA

Introduced by Hu et al [8], Low-Rank Adaptation (LoRA) is an efficient fine-tuning technique that adapts large pre-trained models to specific tasks with fewer parameters being tuned instead of retraining all model parameters. The rationale of LoRA is decomposing weight updates into low-rank matrices:

$$\Omega = \Omega_0 + \Delta\Omega = \Omega_0 + BA \tag{4.1}$$

Where W_0 represents the frozen pre-trained weights, and the update $\Delta\Omega$ is parameterised as the product of two low-rank matrices $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times D}$, $r \ll \min(d, D)$ is the adaptation rank.

By wrapping the query and value projection layers with LoRALinear layers, the $\Delta\Omega_{q,v}$ are added to the original weight matrices $\Omega_{0;q,v}$ (equation 2.9) inside each heads of self-attention mechanism. The new weights $\Omega_{q,v}$ are exploited to do the inference, but only the decomposed matrices A and B undergo gradient updates during the backpropagation.

4.1 Training Preparation

First, I divided the dataset into training (80%) and validation (20%) sets, which consist of 800 and 200 sequences respectively. Then, the sliding windows were built in `train_lora.py` to cut the tokens into chunks of context length with stride of half context length for training set and context length for validation set. After preprocessing and tokenization, each sequence of length 100 time steps (999 tokens) in training set generated $\frac{999\times 2}{512} \approx 4$ chunks, resulted in totally 3200 chunks for training.

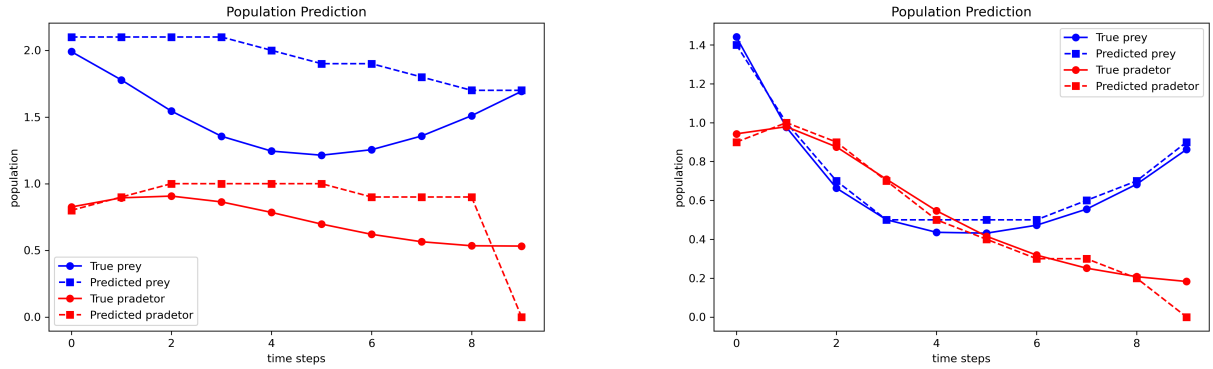
4.2 Training With Default Parameters

The default configuration are listed in table 4:

Table 4: Default parameters for LoRA training

Learning Rate	Rank	Context length	Batch Size	Max Steps	Actual Steps
1e-5	4	512	4	2000	2000

This initial LoRA fine-tuning yielded notable improvements over the untrained baseline model. As shown in figure 8, 9 and table 5, the model achieved an MSE of 0.0889, which represents a 67.7% reduction compared to the untrained model's MSE of 0.2747. The MAE similarly improved to 0.2018 (from 0.3066), and R^2 increased substantially to 0.6643 (from 0.2463).



(a) Example 1

(b) Example 2

Figure 8: Predicted sequences generated by the LoRA fine-tuned model with default parameters for the same two example systems shown in figure 6.

Table 5: Metrics for LoRA fine-tuned model with default parameters

	MSE	**MAE**	**R^2**	**MSE for Prey**	**MSE for Predator**
Model	0.0889	0.2018	0.6643	0.1230	0.0547
Improvement(%)	67.65	34.18	169.73	70.81	57.25

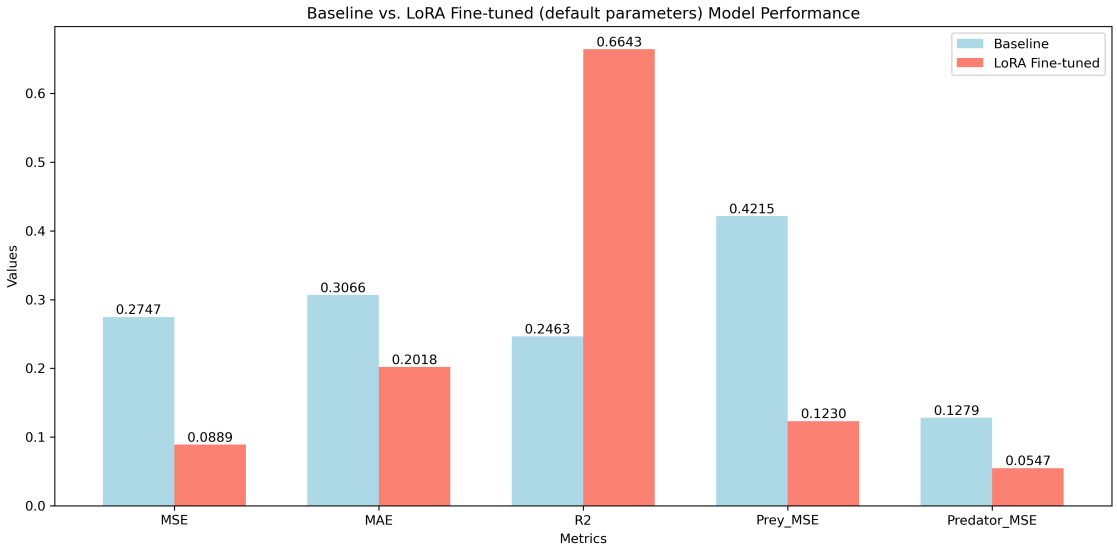


Figure 9: Default fine-tuning v.s. baseline metrics

The improvements demonstrate that even without hyperparameter-tuning, LoRA significantly enhances the model's ability to capture the underlying dynamics of the multivariate time series. Particularly notable is the substantial increase in R^2, indicating the model can now explain a much larger portion of the data variance.

4.3 Hyperparameter Search

To identify optimal hyperparameters for the LoRA adaptation, I searched across the learning rates, LoRA rank and context length. The search was conducted in two phases. In the first phase, I fixed the context length at 320 and varied the learning rate and LoRA rank, resulting in 9 different combinations. Each model was trained for up to 1500 steps, and performance was evaluated on the validation set. In the second phase, I fixed the best parameters found in phase 1 and varied the context length, training each configuration for 2000 steps.

Table 6: Experimental settings of hyperparameter search

Hyperparameter	**Values**	**Max Steps**	**Actual Steps**
Learning rate	$[10^{-5}, 5 \times 10^{-5}, 10^{-4}]$	1500×9	1500×7 + 900×2
LoRA rank	$[2, 4, 8]$		
Context length	$[128, 512, 768]$	2000×3	2000×3

The configurations are listed in table 6, where the training for two of them ((1e-05, 4) and (1e-05, 8)) early stopped at 900 steps due to the `patience=2` I set. (See `results.json` in each experiment's result folder for details and `hyperparamter_result.csv` for general overview).

4.3.1 Learning rate and rank

The outcomes for the learning rate and the LoRA rank search are presented in table 7. The MSE values are used as the indicator for the best configuration choosing.

Table 7: Search results (MSE values) in phase 1

Learning Rate	**Rank 2**	**Rank 4**	**Rank 8**
10^{-5}	0.2940	0.3116	0.2807
5×10^{-5}	0.0937	0.0888	0.0444
10^{-4}	0.0662	0.0445	0.0514

Overall, the lowest MSE (0.0444) is achieved by combining a learning rate of 5×10^{-5} with LoRA rank 8, closely followed by a learning rate of 10^{-4} at rank 4 (0.0445).

In addition, the results reveal several important trends. Firstly, increasing the rank from 2 to 8 generally improved performance. This suggests that the higher rank, which means more trainable parameters, provides more capacity to capture the dynamics of the new data. Secondly, at lower ranks (2 and 4), the faster learning rate of 10^{-4} tends to yield lower MSE values than 5×10^{-5}, suggesting that when fewer parameters are trainable, a larger learning rate can speed up convergence. By contrast, at the higher rank of 8, the mid-range learning rate 5×10^{-5} outperforms 10^{-4}, implying that the larger parameter spaces benefit from the slower updates. In all cases, the slowest learning rate (10^{-5}) produces comparatively larger errors, hinting at potential underfitting or insufficient gradient steps under this setting.

4.3.2 Context length

Based on the findings above, the top-performing configuration (learning rate = 5×10^{-5}, LoRA rank = 8) were selected and proceeded to investigate the impact of context length. The results from this second phase are presented in table 8.

Table 8: Impact of context length on performance

Context Length	MSE	MAE	R^2
128	0.1804	0.2767	0.3192
512	0.0231	0.0879	0.9063
768	0.0312	0.1084	0.8648

Clearly, the result demonstrate that the best performed length is 512 tokens. This is consistent with the nature of the predator-prey dataset, which has the mean period length of around 40 time steps (399 tokens), see figure 5.

4.4 Final Model

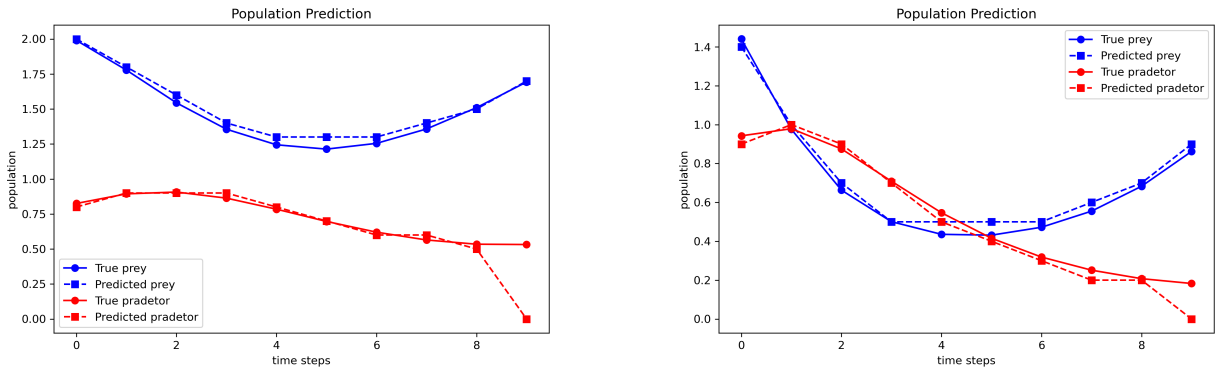
Based on the hyperparameter search results, the following configuration in table 9 was selected for the final model training.

Notably, I initially set the patience for the early stopping mechanism incorrectly to 5, which made it useless because the experiment was set up to evaluate and save checkpoints every 1/5 of the max steps (3000 steps here). To avoid overfitting, I interrupted training at step 9000, where the metrics were no longer improving, and then changed the configurations to evaluate on the validation set every 100 steps with an early stopping patience of 3. After resuming from the checkpoint saved at step 9000, the training finally terminated at step 9700 with a total of 10 evaluations.

Table 9: Configuration for the final model training

Learning Rate	Rank	Context Length	Batch Size	Max Steps	(Actual Steps)
5×10^{-5}	8	512	4	15000	9700

Choosing the same two example systems shown in figure 6 and 8, the predictions showed a great improvement (figure 10). The final model captured both the amplitude and phase of the oscillations with high accuracy, closely matching the true trajectories for both prey and predator populations.



(a) Example 1

(b) Example 2

Figure 10: Predicted sequences generated by the final model for the same two example systems shown in figure 6 and 8.

For the quantitative part, the final model was evaluated on the same metrics as previous experiments (MSE, MAE, R^2) and compared against the untrained baseline, which achieved significant performance improvements. The results are summarised in table 10 and visualised in figure 11.

Table 10: Metrics for the final model

	MSE	MAE	R^2	Prey MSE	Predator MSE
Final Model	0.0169	0.0663	0.9310	0.0057	0.0281
Improvement(%)	93.85	78.38	278.00	98.65	78.03

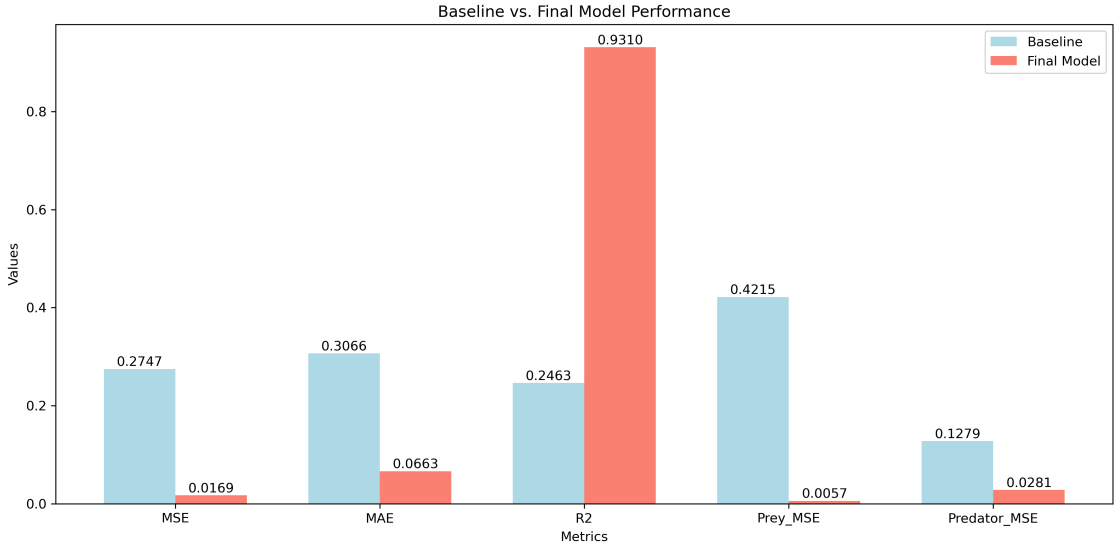


Figure 11: Final model v.s. baseline metrics

The final model achieved an MSE of 0.0169, representing a 93.85% reduction from the untrained baseline. The MAE similarly improved to 0.0663, while the R^2 value increased significantly to 0.9310.

5) Numerical FLOPS Calculation

This section provides a comprehensive overview of the FLOPS calculations for all experiments above. The computations are based on the formulas derived in section 2, with specific values for each experiment's parameters (see `notebooks/run_flops_calculation.ipynb` for more details).

Table 11 summarises the FLOPS usage across all experiments and their proportion of the budget provided. In total, the experiments consumed approximately 72% of the available computational budget, demonstrating efficient utilisation while maintaining a reserve for potential additional experiments if needed.

Table 11: FLOPS Summary for All Experiments

Experiment	FLOPS	% of Budget
Baseline Inference	3.716343×10^{13}	0.0372%
Default LoRA Training	9.444766×10^{15}	9.4448%
Hyperparameter Search	1.751959×10^{16}	17.5196%
Final Model Training	4.527753×10^{16}	45.2775%
Total	$\mathbf{7.227905 \times 10^{16}}$	**72.2790%**

6) Conclusions and Recommendations

Based on the results, this work confirms that LoRA provides an efficient method for adapting large language models to time series forecasting tasks under tight computational constraints. The experiences were summarised and the following are some recommendations for time-series fine-tuning with limited computation:

1. **Exploration of the dataset matters for experiment design**: For instance, using a context length that covers at least one full cycle significantly improves forecasting performance for datasets with periodic patterns.
2. **Set up early stopping and scheduled evaluations**: During these trainings, I configured periodic model evaluations at specified step intervals and implemented early stopping mechanism. When under constrained budgets, reducing the patience of the early stopping can prevent unnecessary computational consumption.
3. **Prioritise hyperparameter search over extended training**: The improvements from hyperparameter tuning yielded greater benefits per FLOPS than simply extending training duration with default parameters.

For future research, I suggest examining the impact of different preprocessing and tokenization strategies on forecasting performance. For example, it can simply begin from tuning the number of decimal places in the method of LLMTIME and potentially develop specialised token embeddings for numerical sequences.

In addition, this study only adapted the LoRA to the query and value projection layers, but future work could explore the impact of applying LoRA to different combinations of layers to potentially achieve better performance with the same parameter count.

References

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

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2023.

[3] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, *Qwen2.5 technical report*, 2025.

[4] Q. Team, "Qwen2.5-0.5b model configuration." https://huggingface.co/Qwen/Qwen2.5-0.5B/blob/main/config.json, 2024.

[5] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, *Roformer: Enhanced transformer with rotary position embedding*, 2023.

[6] Z. Jiang, J. Gu, H. Zhu, and D. Z. Pan, *Pre-rmsnorm and pre-crmsnorm transformers: Equivalent and efficient pre-ln transformers*, 2023.

[7] S. J. Prince, *Understanding Deep Learning*. The MIT Press, 2023.

[8] 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.

Appendix

Declaration: No auto-generation tools were used in this coursework.