UNIVERSITY OF CAMBRIDGE

M2: Deep Learning - Coursework Assignment

at2128

Department of Physics, University of Cambridge

April 6, 2025

Word Count: 2,927

Abstract

LoRA fine-tuning was implemented on the Qwen2.5 0.5B LLM to improve its performance on time-series regression of predator-prey systems. The hyperparameter search and training were performed under an artificially imposed compute budget of 10^{17} FLOPS. A minor 3.7% improvement was displayed on mean absolute error when fine-tuned with a default LoRA configuration, while a further 78% improvement was achieved after training with the best parameters as determined by the hyperparameter search. Drawbacks of using categorical cross-entropy loss for this type of problem were discussed and a modification to the transformer architecture was suggested to facilitate the use of the mean-squared error loss function.

1 Introduction

Large language models (LLMs) have recently experienced significant success and attention, particularly through OpenAI's ChatGPT and open-source models such as Meta's Llama [1]. Most LLMs leverage the transformer architecture, which introduces causal self-attention layers, facilitating variable context lengths and improving training efficiency [2]. LLMs have displayed strong zero-shot performance in predicting future points in time-series data [3].

LLMs are often fine-tuned to specialise in certain tasks. The fine-tuning process can be computationally expensive and require substantial storage to save the tuned models. Low-rank adaptation (LoRA) mitigates some of these issues by freezing the original model parameters and injecting low-rank matrices into each self-attention layer [4]. This results in a much smaller fraction of parameters that need to be updated during each optimisation step. In this project, LoRA is used to fine-tune the Qwen2.5 0.5B LLM on simulated predator-prey time series data under a tight computational constraint (10^{17} FLOPS). The fine-tuning process resulted in significant performance improvements in mean absolute error (MAE) when evaluated on the validation set compared to the untuned Qwen2.5 model.

2 Qwen2.5 Model

2.1 Architecture

The Qwen2.5 0.5B model consists of a traditional transformer decoder architecture with some modifications compared to the original design. The key features of Qwen2.5 that distinguish it from the original transformer architecture include:

- **Grouped Query Attention (GQA)**: Self-attention heads have unique query heads but share key-value heads [5].
- **Rotary Positional Embeddings (RoPE)**: A type of positional embedding that is applied to every self-attention layer rather than once at the start [6].

- **PreNorm & RMSNorm**: RMSNorm is applied before the self-attention and feed-forward network (FFN) in each block [7].
- **SwiGLU Activation**: The SwiGLU activation function is used in each FFN [8].

A block diagram of the model is displayed in Figure 1. Quantitative hyperparameter details of the Qwen2.5 0.5B model are presented in Table 1.

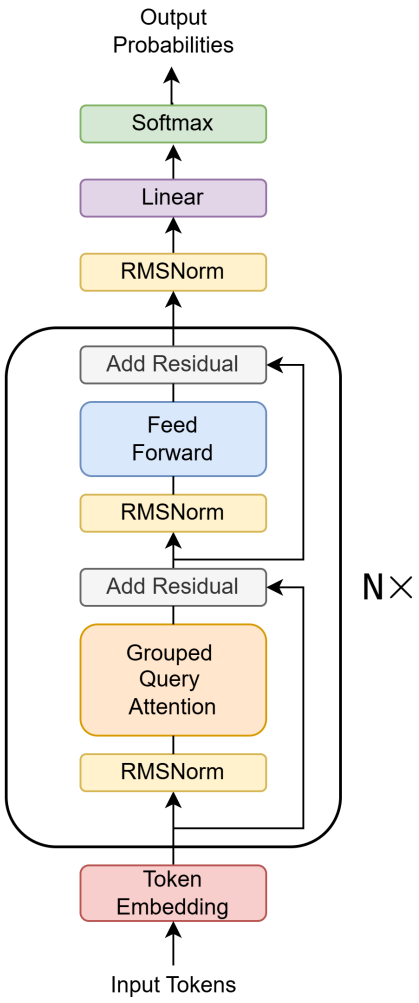

Figure 1: High-level block diagram of the key components of the Qwen2.5 model architecture. The self-attention block repeats N times.

2.2 FLOPS Accountings

The number of floating points operations (FLOPS) for a forward pass of the Qwen2.5 0.5B model was calculated to impose an artificial computational constraint during training. A few simplifications were made during the FLOPS calculation:

1. The number of FLOPS in GQA was assumed to be the same as in standard multi-head self-attention (MHSA).
2. Positional embeddings were added just once to the token embeddings before the self-attention blocks. This is in contrast to RoPE, which is added at every self-attention layer.
3. The number of FLOPS in the backward pass is assumed to be twice the number in the forward pass.

The mapping that was used to determine the number of FLOPS in each operation is broken down in Table 2. The total FLOPS for each component of Qwen2.5 0.5B given a context length of 512 tokens is presented in Table 3.

Attribute	Value
Hidden Size	896
# Self-Attention Blocks	24
# Query Heads	14
# KV Heads	2
Head Size	64
Intermediate Size	4,864
Vocabulary Size	151,646

Table 1: Architecture of the Qwen2.5 0.5B model. Hidden size is the length of an embedding vector, head size is the size of the query/key/value vectors in a self-attention head, and intermediate size is the size of the hidden layer in the FFN.

Operation	FLOPS
Addition/Subtraction/Negation	1
Multiplication/Division/Inverse	1
ReLU/Absolute Value	1
Exponentiation/Logarithm	10
Sine/Cosine/Square Root	10

Table 2: Map of mathematical operations to their corresponding number of FLOPS.

Component	FLOPS
Positional Embeddings	4.95×10^5
RMSNorm	5.97×10^6
Add Residual	4.95×10^5
Multi-Head Self-Attention	4.27×10^9
Feed Forward	1.34×10^{10}
Self-Attention Block $\times 1$	1.77×10^{10}
Self-Attention Blocks $\times 24$	4.25×10^{11}
Final RMSNorm	5.97×10^6
Linear	1.39×10^{11}
Softmax	9.33×10^8
Forward Pass	5.65×10^{11}
Forward + Backward Pass	1.70×10^{12}

Table 3: Breakdown of the FLOPS usage in a single forward pass of the Qwen2.5 0.5B model with a context length of 512. Italicized entries denote intermediate computations reused across multiple components. The FLOPS for each component is computed over all the 512 input tokens during training.

3 Dataset and Preprocessing

The dataset contained 1,000 predator/prey systems. Each system contained 100 time steps of predator/prey populations, giving a dataset shape of $(1000, 100, 2)$. The 1,000 systems were split into a training set of 850 and a validation set of 150. The subsequent preprocessing steps of each dataset were performed independently to avoid data leakage. The preprocessing of each system was performed similarly to that in LLMTIME [3].

3.1 Scaling

For each dataset, the max population value was determined, considering each predator/prey population. The max population value, $\max(x)$, was then used to compute the scale factor:

$$\alpha = \frac{\max(x)}{9.9}, \tag{1}$$

which was used to compute the scaled values: $x' = \frac{x}{\alpha}$. The denominator in Equation 1 was set to 9.9 to ensure that each of the scaled population values would be normalised to strictly less than 10. The scaling was performed so that the fine-tuned model would not need to generalise to inputs of greater magnitude than it had seen during training.

3.2 String Formatting and Tokenisation

After scaling, the population values were rounded to two decimal places to prevent arbitrarily long input sequences. Then, the predator/prey points in each system were formatted as a string, with semicolons used to separate time steps. Commas were used to separate the predator and prey population values at each time step.

The Qwen2.5 tokeniser was then used to convert from string to token ids. Qwen's tokeniser splits numerical strings into individual digits by default. An example of the preprocessing steps taken after scaling is demonstrated for two time steps in Figure 2.

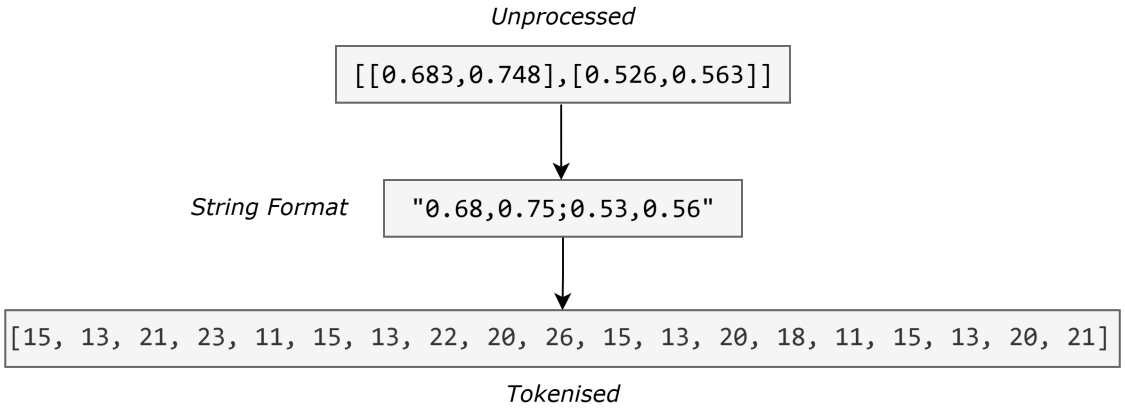

Figure 2: Block diagram illustrating the steps taken to obtain the token IDs from a predator/prey system with two time steps. The first step involves rounding to two decimal places and transforming to the string format. The second step involves using the Qwen2.5 tokeniser to obtain the token IDs from the string.

4 Baseline Performance

To establish a baseline, the untuned Qwen2.5 model was tested on the validation set of 150 systems. Mean absolute error (MAE) was used as the primary metric to evaluate the final model's performance. While categorical cross-entropy loss measures the model's ability to predict the individual token, it does not directly reflect the numerical accuracy of the prediction, which is directly addressed by MAE. This is an important consideration: a model might perform well under cross-entropy by correctly predicting less significant digits but still produce numerically inaccurate

forecasts, resulting in a higher MAE. Nonetheless, cross-entropy loss was used to guide hyperparameter selection. This was justified as no changes were made to tokenisation or numeric precision (e.g. number of decimal places) that would have fundamentally changed the token distribution.

Before inference, an output mask was applied so that the model only output tokens associated with the integers 0 through 9, dots, commas, and semicolons. This prevented the model from generating malformed sequences resulting from non-numeric tokens. The model autoregressively generated 20 time steps into the future over context lengths of 40 and 80. The combined MAE metric was computed by averaging the MAE over the 20 predicted future points, across both predator and prey variables, and over context lengths of 40 and 80. The model performed with a categorical cross-entropy loss of 0.57 and a combined MAE of 0.53, evaluated on the validation set. The MAE at each future time is shown in Figure 3. As expected, the errors increase with the number of time steps from the first prediction, and accuracy improves with a larger context length.

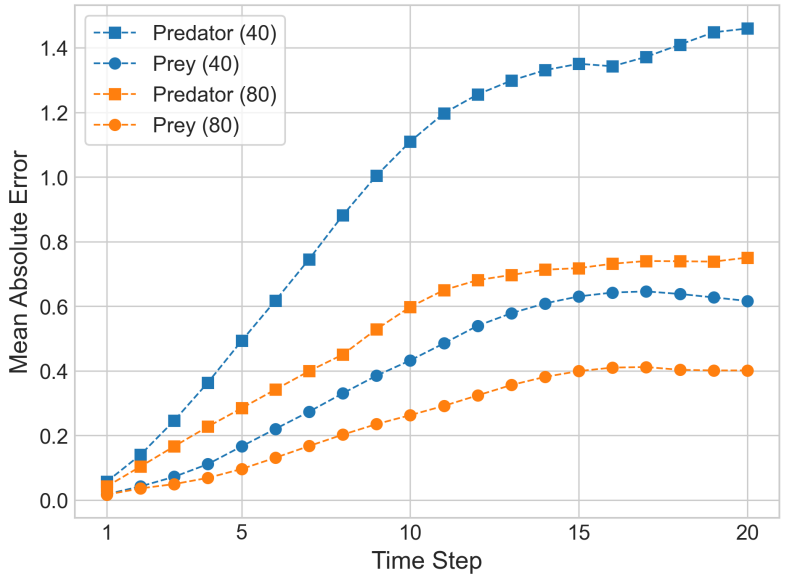

Figure 3: MAEs of the untuned Qwen2.5 model in predicting 20 future time steps for both predator and prey populations over 40 and 80 context lengths. As expected, the model performs better with a larger context length, and accuracy decreases with increasing time steps from the start.

5 LoRA Finetuning

5.1 LoRA

LoRA is a method of fine-tuning where the parameters of the pre-trained model are frozen, and trainable matrices are injected into the model. In this case, the matrices were applied to the query and value projections in the self-attention layers. This involves adding some matrix ΔW_q to the query matrix (and similarly to the value matrix):

$$\boldsymbol{q} = (W_q + \Delta W_q) \cdot \boldsymbol{x} + \boldsymbol{\beta}_q, \tag{2}$$

where $\boldsymbol{q}$ is the query vector, W_q is the original query matrix, and $\boldsymbol{\beta}_q$ is the bias term. ΔW_q is decomposed as

$$\Delta W_q = A_q B_q, \tag{3}$$

where $A_q \in \mathbb{R}^{d_h \times r}$, and $B_q \in \mathbb{R}^{r \times d_h}$, with d_h being the head size, and r being the LoRA rank. Typically, $r \ll d_h$, meaning that the model needs to train and store a much smaller number of parameters compared to directly optimising the parameters in W_q. The LoRA contribution to FLOPS was neglected.

5.2 FLOPS Budget

A key requirement of this project was to perform hyperparameter search and tuning under a FLOPS budget of 10^{17}. It was decided that approximately 10% of the budget should be used for the initial training run to establish a baseline and 20% to train the final model. This left 70% to perform hyperparameter tuning, in which 12 experiments were planned: 10 experiments at a chunk size of 512 and two additional runs at chunk sizes of 128 and 768. This permitted approximately 880 optimiser steps per experiment.

5.3 Initial Training Run

Before training, chunking was performed on the training set with a chunk size of 512 and stride of 268. Padding was added to any chunks that fell short of 512 elements. This quadrupled the original number of training samples (although many of the samples were now correlated). Chunking was not performed on the validation set.

The initial training run was performed with the following hyperparameters:

- Learning rate: 10^{-5}
- LoRA rank: 4
- Batch size: 4

The training loss was tracked every optimiser step and validation loss every 200 steps using the `wandb` package. After 1085 training steps, training was stopped to conserve the FLOPS budget. The fine-tuned model achieved a combined MAE equal to 0.51 and a validation loss of 0.53. This represented a minor improvement over the untuned Qwen2.5 performance.

5.4 Hyperparameter Tuning

Hyperparameter search was carried out for learning rates in $\left(10^{-5}, 5 \times 10^{-5}, 10^{-4}\right)$ and LoRA ranks in $(2, 4, 8)$. The batch size was held at 4 and the chunk size at 512. Since there were only nine possible combinations to trial, grid search was used to search the space. 170 training systems (20% of the total training set) were used so that each experiment could train for more epochs given the FLOPS budget. To conserve FLOPS, the trained model was only evaluated once on the validation set at the end of the training process. The grid search results are presented in Table 4, which displays the validation loss for each hyperparameter combination. A learning rate of 10^{-4} and LoRA rank of 8 resulted in the best performance on the validation set after 880 optimiser steps.

	10^{-5}	5×10^{-4}	10^{-4}
2	0.56	0.46	0.41
4	0.54	0.42	0.38
8	0.51	0.39	0.34

Table 4: Summary of the validation loss (categorical cross-entropy) for each point in the hyperparameter grid. The rows correspond to LoRA rank and columns to learning rate. The hyperparameter combination with the best validation loss is highlighted in green (learning rate of 10^{-4} and LoRA rank of 8).

The best learning rate and LoRA rank were used to run two further experiments with chunk sizes of 128 and 768, resulting in validation losses of 0.40 and 0.41, respectively. As a result, the best hyperparameter combination was chosen to be a learning rate of 10^{-4}, LoRA rank of 8, and chunk size of 512.

5.5 Final LoRA Performance

A final training run was performed on the whole training set of 850 systems, using the best hyperparameter combination as determined by the hyperparameter search. Training was run for 2499 optimiser steps, with the validation loss computed every 200 steps. After training, when evaluated on the validation set, the fine-tuned model achieved a combined MAE of 0.11 and a

validation loss of 0.31. This demonstrates a significant performance improvement over the untuned model and fine-tuned model with default hyperparameters. The MAE at each future time step is displayed in Figure 4.

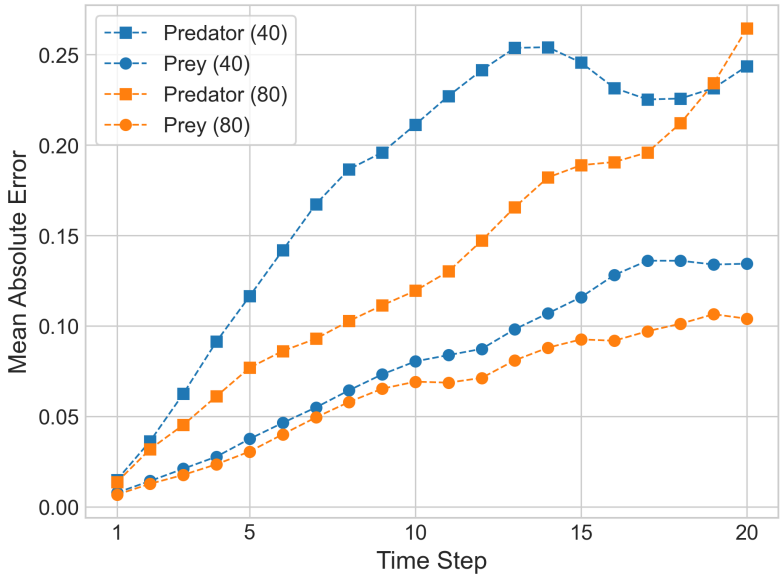

Figure 4: Validation MAEs of the optimised LoRA model in predicting 20 future time steps for both predator and prey populations over 40 and 80 context lengths.

A comparison of the MAEs of the unturned model, LoRA tuned with default hyperparameters, and LoRA tuned with the optimised hyperparameters is presented in Figure 5. There is a minor improvement in the performance of the default LoRA model compared to the untuned model but a significant improvement exhibited by the LoRA model with optimised hyperparameters. The improvements of the optimised hyperparameter model is seen over different context lengths and for both predator and prey populations. Examples of the three model configurations in predicting future steps are shown in Figure 6 for three systems in the validation set.

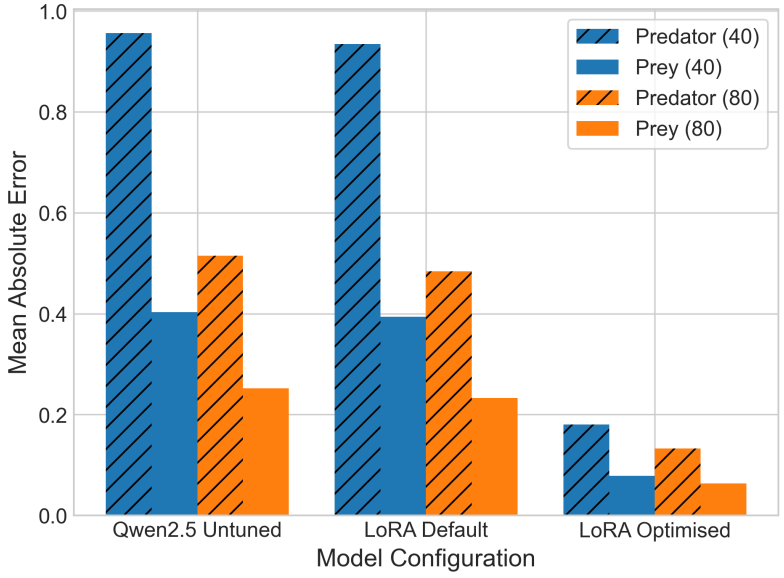

Figure 5: MAE of predator and prey population forecasts across three model configurations: Qwen2.5 Untuned, LoRA Default, and LoRA Optimised. Results are shown on the validation set for context lengths of 40 and 80.

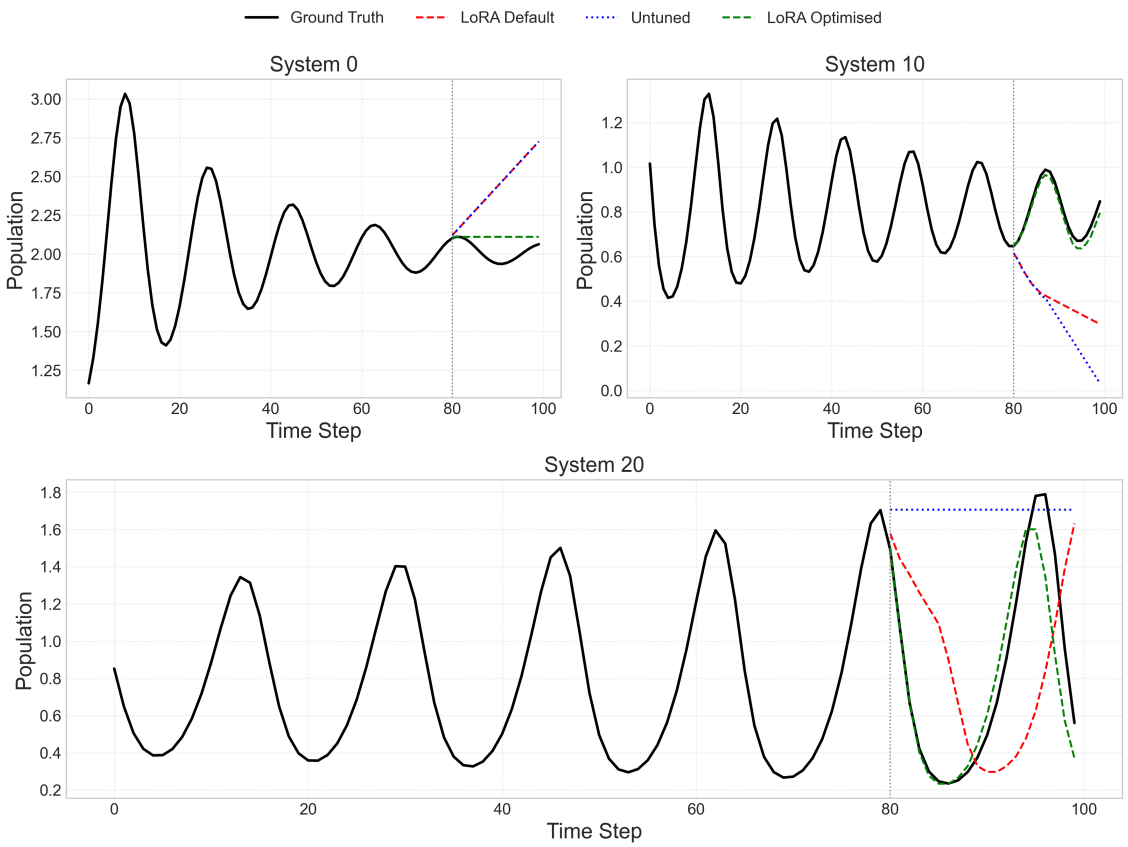

Figure 6: Plot of autoregressive inference of 20 future time steps given a context length of 80 for the Qwen2.5 model, default LoRA model, and optimised LoRA model. The systems displayed are the 0th, 10th, and 20th indices of the validation set. System 20 has a wider horizontal scale than system 0 and system 10 so that the future prediction paths can be more easily distinguished. These examples demonstrate a minor improvement in the predictive power of the default LoRA model and a significant improvement for the optimised LoRA model, consistent with the MAE metrics.

5.6 FLOPS Breakdown

The FLOPS usage during each experiment (training and final model evaluation) is broken down in Table 5.

	Total FLOPS
Initial Training Run	8.19×10^{15}
512 Chunk Size Experiment ×9	5.83×10^{16}
128 Chunk Size Experiment	1.95×10^{15}
768 Chunk Size Experiment	9.15×10^{15}
Final Training Run	1.94×10^{16}
Total	9.70×10^{16}

Table 5: Summary of the FLOPS used during each experiment/training run. Each entry contains both the FLOPS used for training as well as FLOPS used for evaluation of the model.

5.7 Discussion

The highest-performing model used the largest learning rate during the training process. It was noted that the validation loss in the lower learning rates had not begun plateauing before training had stopped. This meant that there was a possibility that a lower learning rate may have performed better if training had been allowed to run for more steps. But for hyperparameter tuning within a tight compute budget, it may be beneficial to focus on higher learning rates as they can result in greater improvement after a smaller number of steps.

In addition, running small-scale experiments (e.g. with 20% of the total training set) can be advantageous as it allows for the model to see a greater number of epochs of data for the same number of optimiser steps. The results of the small-scale experiments can then be used as a proxy for how the model might perform when trained on the full dataset.

One important aspect is that categorical cross-entropy loss, which is minimised during training, does not align well with key performance metrics of time-series regression, such as mean-squared error (MSE). Accordingly, better performances may be achieved if the transformer architecture were modified so that inputs were an array of continuous values rather than discrete tokens. This would permit the transformer to output continuous values, which could be used to compute the MSE loss. This should address the issue discussed in Section 4, where cross-entropy loss may result in less significant figures being optimised at the expense of more significant ones.

6 Summary

An estimate of the total number of FLOPS used in the forward pass and backward pass of the Qwen2.5 0.5B model was computed based off of the model architecture. The zero-shot performance of Qwen2.5 was evaluated predicting future data points of a set of predator-prey time-series systems. The performance was then compared to Qwen2.5 after it had been fine-tuned on predator-prey systems with LoRA on a default set of hyperparameters. The default LoRA model experienced a minor performance improvement compared to the untuned one. A hyperparameter grid search was subsequently carried out using small scale experiments, and the most optimal hyperparameters were used to fine-tune the model on the full training set. All of the experiments and training runs were performed within a 10^{17} FLOPS budget. The optimised LoRA model achieved a combined MAE of 0.11 which represented a significant improvement over the performance of the untuned model (0.53) and default LoRA model (0.51). Recommendations for time-series fine-tuning under a tight compute constraint were then discussed, including using a higher learning rate, using subsets of the training set, and potentially adjusting the model architecture to facilitate the use of MSE as a loss function.

References

[1] Hugo Touvron et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[2] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[3] Nate Gruver et al. "Large language models are zero-shot time series forecasters". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 19622–19635.

[4] Edward J Hu et al. "Lora: Low-rank adaptation of large language models." In: *ICLR* 1.2 (2022), p. 3.

[5] Joshua Ainslie et al. "Gqa: Training generalized multi-query transformer models from multi-head checkpoints". In: *arXiv preprint arXiv:2305.13245* (2023).

[6] Jianlin Su et al. "Roformer: Enhanced transformer with rotary position embedding". In: *Neurocomputing* 568 (2024), p. 127063.

[7] Zixuan Jiang et al. "Pre-RMSNorm and Pre-CRMSNorm transformers: Equivalent and efficient pre-LN transformers. CoRR, abs/2305.14858, 2023. doi: 10.48550". In: *arXiv preprint arXiv.2305.14858* ().

[8] Noam Shazeer. "Glu variants improve transformer". In: *arXiv preprint arXiv:2002.05202* (2020).

A Autogeneration Tools Declaration

ChatGPT was used to rephrase certain portions of the report for clarity. The autogeneration tools declaration for the code is located in the README.md file situated in the root directory of the repository.