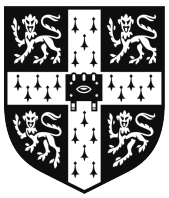


UNIVERSITY OF CAMBRIDGE

M2 Deep Learning - Coursework Report

Jacob Tutt (JLT67)

Department of Physics, University of Cambridge

Coursework Documentation

April 6, 2025

Word Count: 2997

1 Introduction

Large Language Models' (LLMs) ability to capture multi-modal distributions alongside their tendencies toward pattern simplicity and repetition was suggested by Gruver et al. [1] to be responsible for their success as 'zero-shot time-series forecasters'. This paper builds upon those observations by applying Low-Rank Adaptation (LoRA) [2] to the Qwen2.5-Instruct model [3] to investigate the potential performance increase that can be achieved by downstream training. This is performed under a tight computational budget of 10^{17} floating point operations (FLOPS).

The model is evaluated on a dataset generated from the Lotka–Volterra (Predator–Prey) equations, a set of first-order, nonlinear ODEs, which offer a benchmark of coupled dynamics, common in domains like astrophysics, economics and finance.

$$\frac{dx}{dt} = \alpha x - \beta xy \tag{1}$$

$$\frac{dy}{dt} = \delta xy - \gamma y \tag{2}$$

where $x(t)$ and $y(t)$ represent prey and predator populations, and α, β, δ, and γ govern growth, interaction, and decay.

2 Prepossessing of Data

There is huge variability in the approaches that can be used to encode 2D time-series data into 1D strings for input into Qwen's tokenizer. Due to computational constraints, hyperparameter tuning across these permutations was not feasible; thus, decisions were guided by the scheme presented by Gruver et al. [1] and simple baseline tests of model generation performance.

The resulting scheme scales the data so that 90% of values fall between 0-10, truncates values to two decimal places, separates time points with semicolons (;) and predator–prey pairs with commas (,). Two representative examples from the first five points of the first two (unshuffled) sequences are shown below, from which Qwen's vocabulary mapping can be determined (Table 2).

Finally, the preprocessing pipeline returns three shuffled datasets (with a fixed seed) for training (70%), validation (15%), and testing (15%), ensuring independent evaluation and adherence to good machine learning practice. These are passed to a custom PyTorch Dataset module that handles tokenisation, batching, and padding. This produces token sequences compatible with a DataLoader for efficient training and evaluation.

Example 1

Prey (raw): `['2.734', '2.132', '1.964', '2.063', '2.373']`

Predator (raw): `['2.996', '2.244', '1.625', '1.173', '0.864']`

String: `2.734,2.996;2.132,2.244;1.964,1.625;2.063,1.173;2.373,0.864`

Tokenised: `[17, 13, 22, 18, 19, 11, 17, 13, 24, 24, 21, 26, 17, 13, 16, 18, 17, 11, 17, 13, 17, 19, 19, 26, 16, 13, 24, 21, 19, 11, 16, 13, 21, 17, 20, 26, 17, 13, 15, 21, 18, 11, 16, 13, 16, 22, 18, 26, 17, 13, 18, 22, 18, 11, 15, 13, 23, 21, 19]`

Example 2

Prey (raw): `['2.797', '3.105', '3.629', '4.381', '5.356']`

Predator (raw): `['2.894', '2.366', '1.976', '1.707', '1.544']`

String: `2.797,2.894;3.105,2.366;3.629,1.976;4.381,1.707;5.356,1.544`

Tokenised: `[17, 13, 22, 24, 22, 11, 17, 13, 23, 24, 19, 26, 18, 13, 16, 15, 20, 11, 17, 13, 18, 21, 21, 26, 18, 13, 21, 17, 24, 11, 16, 13, 24, 22, 21, 26, 19, 13, 18, 23, 16, 11, 16, 13, 22, 15, 22, 26, 20, 13, 18, 20, 21, 11, 16, 13, 20, 19, 19]`

Table 1: Raw and tokenised time series data examples

Character	0	1	2	3	4	5	6	7	8	9	.	,	;
Token ID	15	16	17	18	19	20	21	22	23	24	13	11	26

Table 2: Qwen's token ID mapping for the digits and separators

2.1 Justification of Decimal Places and Scaling Scheme

While decimal places are arguably 'redundant given a fixed precision' [1], baseline tests with Qwen suggest performance improves when they are retained. Although a full investigation is beyond the scope, we hypothesise that decimal markers help the model in distinguishing relative magnitudes. The choice of two decimal places is justified later in subsection 7.1.

To avoid wasting tokens on outliers, data is scaled so a majority of values fall within a normalised range. Gruver et al. [1] recommends percentile-based scaling over max-based scaling to help models generalise to larger systems through exposing it to a subset of data points with a larger number of digits. Following this, we scale to 0-10 rather than 0-1, as it provides a number of examples with an extra digit (ie 9.99-10.00 vs 0.99-1.00).

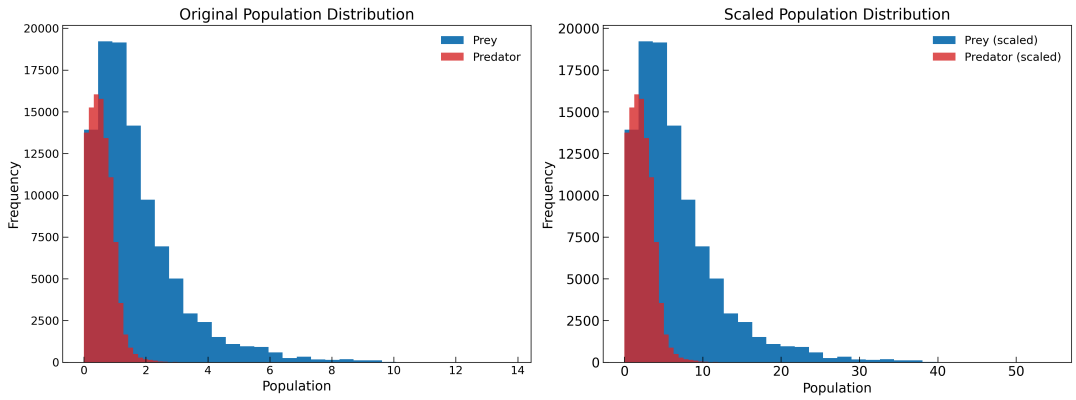


Figure 1: Distribution of data points before and after scaling

Since predator's counts are typically lower than the prey's, scaling the two populations or even each trajectory independently was considered. However, if scaling is applied independently, each trajectory effectively encodes a different set of ODEs (proved in Notebook 1), interfering with generalisation and training. Thus the scaling factor α must be applied globally.

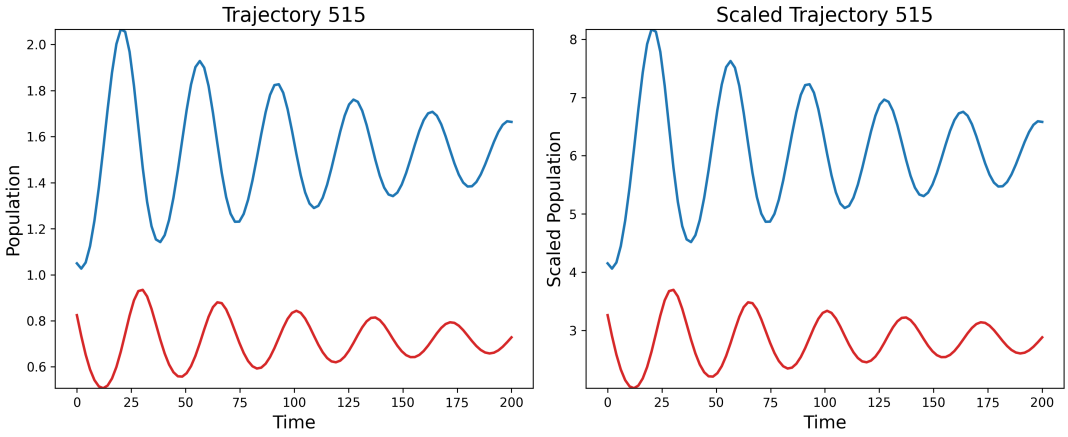


Figure 2: Trajectory 515 before and after scaling

3 Qwen's Baseline Evaluation

Qwen's initial performance was evaluated using the test set to establish a benchmark for assessing the training improvements achieved through Low-Rank Adaptation. The methods and metrics used are outlined below, with results compared across models in section 9.

3.1 Cross Entropy Loss

The first evaluation uses the cross-entropy loss across all tokens, quantifying the divergence from the model's output distribution to the target distribution. This provides a metric which reflects its ability to capture the underlying structure of the tokenised time series data.

While effective for training, it's short-term scope only evaluates the model's ability to predict the next immediate token (a single digit). However, most time-series forecasting applications desire longer-range consistency and an ability to extrapolate coherent patterns over extended periods.

3.2 Auto-regressive Metrics

To complement cross-entropy's stunted view, we assess the model's ability to generate longer-range trajectories using auto-regressive generation. We deemed greedy (deterministic) decoding advantageous, due to time-series forecasting requiring consistency and stability. However, future work could explore if any forecasting benefit is gained from probabilistic sampling or diverse outputs.

We report the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE). MAPE was chosen for its scale invariance, enabling fairer comparison across time steps and population magnitudes. We used an initial context of 50 data points and tasked the model to generate the next 50. This choice reflects the final model's training context length of 768 tokens, which corresponds to ≈ 77 data points when using 2 decimal places per value. This allows evaluation of the final model's behavior both within and beyond its trained context window. Results were averaged over all 150 trajectories in the test set.

4 Model Architecture

This section outlines Qwen's architecture with Low-Rank Adaptation (LoRA), focusing on the methodology and assumptions used to estimate FLOPs. Mathematical descriptions are detailed in the documentation. These components are composed into higher-level functions and integrated into the training, evaluation, and generation pipelines, enabling automatic FLOP computation, exporting to `.json` files, and tracking with Weights & Biases (wandb).

4.1 Qwen Skeleton

Qwen's 24-layer transformer is summarised in Figure 3, with key architectural components detailed below.

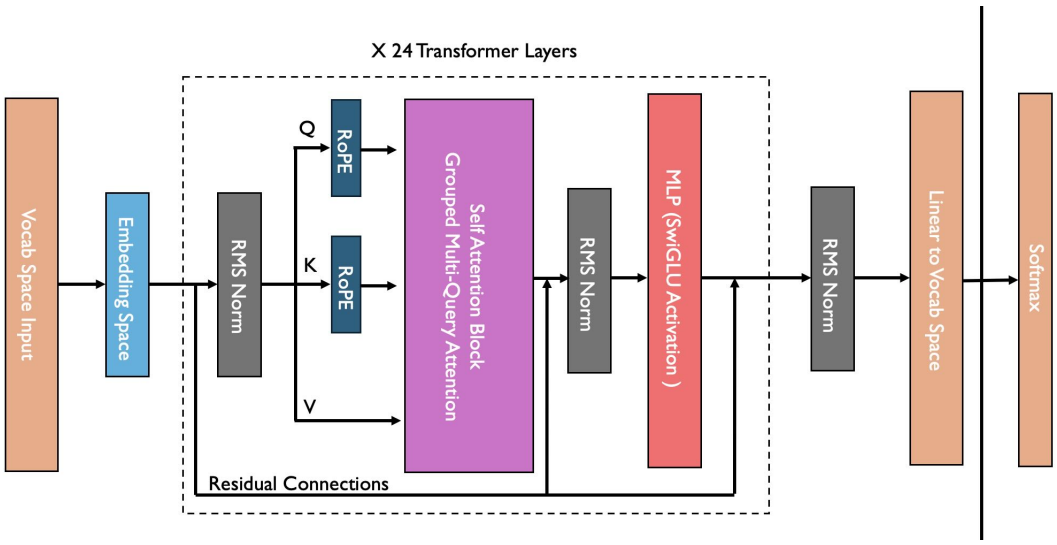


Figure 3: Qwen's Architecture

4.1.1 RMSNorm

RMSNorm is applied twice per transformer layer and once at the final output. It regularises the summed inputs to each neuron in a layer, aiming to provide learning rate adaptation ability and re-scaling invariance [6].

$$\text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2} \tag{3}$$

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} \tag{4}$$

4.1.2 Rotary Positional Embeddings (RoPE)

RoPE is applied to each query and key matrix in the self-attention block via a rotational matrix. This encoding captures relative positions while preserving absolute token distances [5]. The overall transform is defined as:

$$f_{\{q,k\}}(x_m, m) = R^d_{\Theta,m} W_{\{q,k\}} x_m \tag{5}$$

where the rotation matrix in d dimensions, $R^d_{\Theta,m}$ is:

$$R^d_{\Theta,m} = \begin{pmatrix} \cos(m\theta_1) & -\sin(m\theta_1) & 0 & 0 & \cdots & 0 & \\ \sin(m\theta_1) & \cos(m\theta_1) & 0 & 0 & \cdots & 0 & \\ 0 & 0 & \cos(m\theta_2) & -\sin(m\theta_2) & \cdots & 0 & \\ 0 & 0 & \sin(m\theta_2) & \cos(m\theta_2) & \cdots & 0 & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & \cos(m\theta_{d/2}) & -\sin(m\theta_{d/2}) \\ 0 & 0 & 0 & 0 & \cdots & \sin(m\theta_{d/2}) & \cos(m\theta_{d/2}) \end{pmatrix}$$

During FLOP estimation, RoPE is treated as a full matrix multiplication on the global query/key heads, omitting the internal cost. Although Su et al. [5] propose a more efficient computation by exploiting the sparsity of $R^d_{\Theta,m}$ (6), it is unclear whether this is applied in Qwen, and hence conservatively excluded.

$$R^d_{\Theta,m}x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix} \tag{6}$$

4.1.3 Group Query Attention

The Qwen model uses differing numbers of query/key and value heads; for simplicity, calculations assume a traditional multi-head attention with 14 unique heads. In masked self-attention, the upper triangular part of the attention matrix is masked to enforce causality. As it is unclear whether these masked multiplications are performed or treated as memory operations, we conservatively include their full computational cost. Finally, the linear projection after the concatenation of attention outputs is accounted for.

4.1.4 Multi-Layer Perceptron with SwiGLU

The feedforward block in Qwen consists of a Multi-Layer Perceptron (MLP) with a SwiGLU activation [4]. This can be interpreted as two parallel projections into the hidden dimension (4864), one of which is passed through the Swish activation, and then element-wise multiplied with the second. This result is then linearly projected back to the original dimension.

$$\text{SwiGLU}(x, W, V, \beta) = \text{Swish}_\beta(xW) \odot (xV) \tag{7}$$

$$\text{Swish}_\beta(z) = \frac{z}{1 + e^{-\beta z}} \tag{8}$$

4.2 Residual Connection

Residual connections are applied twice per transformer layer and allow information from earlier layers to be carried forward, improving gradient stability during training.

4.2.1 Vocabulary Projection

Finally, Qwen applies a linear projection that maps the model's output from the embedding space to the vocabulary space (151,936 tokens) to produce the output logits.

4.3 Low-Rank Adaptation (LoRA)

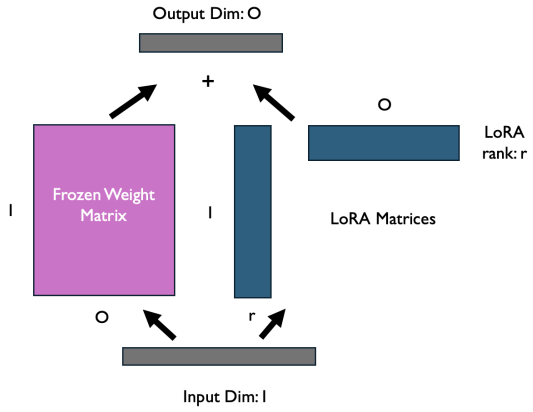


Figure 4: Overview of LoRA

LoRA [2] is applied to the Qwen model by injecting trainable low-rank matrices into the attention mechanism of all 24 transformer layers, specifically alongside the frozen query and value

projection paths (Figure 4). Instead of updating the full weight matrix, LoRA introduces a low-rank residual term to the original projection output, enabling efficient adaptation on this rather than the larger original weights.

$$\Delta W = AB, \quad A \in \mathbb{R}^{O \times r}, \quad B \in \mathbb{R}^{r \times I}, \quad \text{with } r \ll \min(O, I) \tag{9}$$

$$xW + \alpha ABx, \quad \text{where } x \in \mathbb{R}^I,\ W \in \mathbb{R}^{O \times I},\ \alpha \in \mathbb{R} \tag{10}$$

4.4 Values Achieved

The estimated FLOPs breakdown below uses a LoRA rank of 4 and a context length of 512 tokens.

4.4.1 Single Forward Pass

Operation Cost	**Add.**	**Mult.**	**Div.**	**Exp.**	**Sqrt**	**Total**
Single Attention Block	2.93×10^9	2.94×10^9	7.34×10^6	3.67×10^7	10	5.91×10^9
Single MLP Block	6.69×10^9	6.7×10^9	2.49×10^6	2.49×10^7	0	1.34×10^{10}
RMS, Residual etc	1.83×10^6	1.84×10^6	9.19×10^5	0	1.02×10^4	4.6×10^6
Single Transformer Layer	9.63×10^9	9.64×10^9	1.07×10^7	6.16×10^7	1.02×10^4	1.93×10^{10}
LoRA Cost in this Layer	7.34×10^6	8.26×10^6	0	0	0	1.56×10^7
Full Forward Pass	3.01×10^{11}	3.01×10^{11}	2.58×10^8	1.48×10^9	2.51×10^5	$\mathbf{6.04 \times 10^{11}}$
Percentage of Budget:			0.000604%			

Table 3: FLOPs for forward pass: LoRA Rank 4.

4.5 Evaluation

When evaluating, the cost of the cross-entropy calculation across logits is not considered internal to the model and is thus excluded. Overall, evaluation is simply a forward pass over the given number of batches.

4.5.1 Training (Forward + Back Pass)

We assume the backward pass has twice the cost of the forward pass, however in reality the LoRA adaptation means only the gradients with respect to the low-rank matrices (and the biases in the LM head) need calculation as the base model remains frozen, significantly reducing the computation.

Training Setup	**Total FLOPs**	**LoRA FLOPs**	**% of Total Budget**
Single Step ($N = 1$, $B = 4$)	7.2455×10^{12}	4.4909×10^9	0.0072%
Training Limit ($N = 13800$, $B = 4$)	9.9988×10^{16}	6.1975×10^{13}	99.988%

Table 4: FLOPs cost of training using LoRA rank 4.

4.5.2 Generation

Assuming no sliding context window, autoregressive generation requires successive forward passes of context: n, $n+1$, …, $n+y-1$, which makes it computationally demanding. While functions are provided that support both greedy and probabilistic (inc softmax) generation cost, it is excluded from the reported FLOPs budget.

4.6 FLOPs Forecast Budget

Mirroring the considerations typical of resource-limited research, a approximate FLOPs budget was established prior to training. Accounting for periodic evaluation, we are limited to 13,800 training steps.

Table 5: Planned FLOPs Budget

Stage	**Training**	**Evaluation**	**Total (% of Budget)**
Baseline Eval (Cross Entropy)	0	2.41×10^{14}	2.41×10^{14} (0.24%)
Baseline Eval (Autoregressive)	N/A	N/A	N/A
Default Model Training	7.25×10^{15}	5.74×10^{14}	7.81×10^{15} (7.82%)
Default Eval (Cross Entropy)	0	2.41×10^{14}	2.41×10^{14} (0.24%)
Default Eval (Autoregressive)	N/A	N/A	N/A
Hyperparameter Tuning	3.98×10^{16}	3.94×10^{15}	4.37×10^{16} (43.72%)
Final Model Training	4.43×10^{16}	3.88×10^{15}	4.82×10^{16} (48.20%)
Final Eval (Cross Entropy)	0	2.41×10^{14}	2.41×10^{14} (0.24%)
Final Eval (Autoregressive)	N/A	N/A	N/A
Total	$\mathbf{9.14 \times 10^{16}}$	$\mathbf{9.12 \times 10^{15}}$	$\mathbf{1.00 \times 10^{17}}$ (100%)

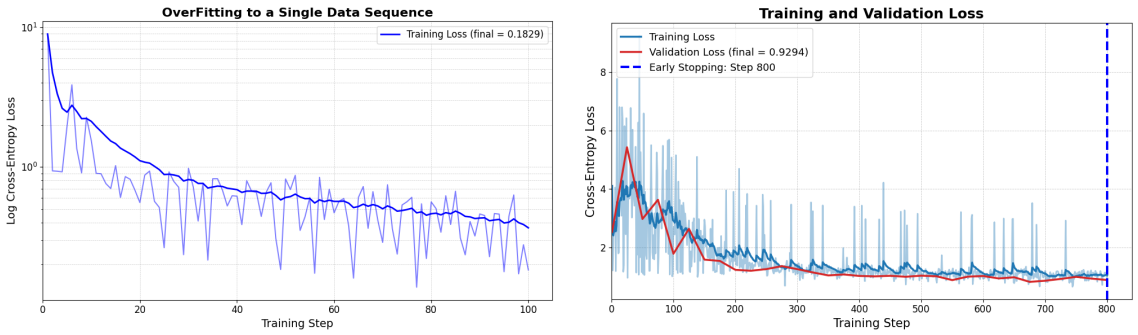
5 PyTorch Pipeline

The PyTorch pipeline was built from modular functions, aggregated into autonomous scripts (e.g., `Full_run_script.py`, `Full_hyper_wandbsweep.py`) for streamlined training, evaluation, and metric computation via single commands. Both performance and FLOPs metrics are logged to `.json` files and wandb.

To reduce cost, several flexibilities were introduced: early stopping was implemented based on stagnation in validation loss, and validation during training was performed at a user-defined frequency on a subset of the data (typically 25 batches every 25 steps). Full details are provided in the documentation.

6 Initial Training

We now turn to the downstream training of Qwen. As fully outlined in subsection 4.3 and Figure 4, LoRA enables efficient fine-tuning by freezing the base model and training only the low-rank residual terms: $A \in \mathbb{R}^{O \times r}$, $B \in \mathbb{R}^{r \times I}$ and the scalar α (9). This is specifically applied to the Query and Key projection paths. The final projection layer's bias terms also remain trainable, to allow the model to bias itself away from the natural text tokens and toward the relevant 12 (Table 2) from the 151,936 available. Future work could further improve efficiency by restricting the output projection to these 12 tokens, reducing projection compute.



(a) Overfitting a single trajectory

(b) Training with default hyperparameters

Figure 5: Intial training with performance (with EWMA)

Given the restricted compute, running extended training with unoptimised hyperparameters was deemed wasteful; thus, two smaller-scale investigations were conducted. To confirm the model could learn, we first overfit a single trajectory (learning rate: 1×10^{-3}, LoRA rank: 4, batch size: 2). This showed promising results, reaching a training loss of 0.182 in just 100 training steps, although an unsurprising much higher final validation loss of 1.519 (Figure 5a).

Secondly, we conducted a run of 1000 training steps on the training set to test the model's ability to generalise across examples (Learning rate: 1×10^{-5}, LoRA Rank: 4, Batch Size: 4), during which early stopping with a patience of 5 was used. Notably early stopping was triggered at 800 steps (val-loss: 0.929) with very slow learning beyond 400 further supporting the need for

hyperparameter tuning before extended runs are performed. Test set metrics from this trained model appear in section 9.

7 Hyperparameter Tuning

A full grid search over the proposed hyperparameters would require 11 complete runs. While allowing exhaustive evaluation of all permutations, it was determined too computationally inefficient. To conserve resources and accommodate additional experiments, each hyperparameter axis was investigated independently, reducing runs to a hypothetical 7. Early tests used a token length of 256, halving the default computational cost before treating it as a tunable parameter. While early stopping could have further reduced compute, a fixed training window of 800 steps was chosen to ensure fair comparisons across runs. Hyperparameter optimisation was performed using `wandb sweep`, with the supporting script and example configuration provided in `Full_hyper_wandbsweep.py` and `wandb_hyperparam.yaml`. A summary of all experiments' outcomes is presented below.

7.1 Investigation 1: Decimal Places

An initial experiment compared 2 and 3 decimal place precision to evaluate whether reduced precision (more values per context window) or higher precision (more informative) would be more effective at this training scale. Initial results showed 2 decimal places yielded lower validation loss and were adopted throughout. However, cross-entropy treats each token equally, so higher precision data is penalised more heavily for smaller errors, potentially inflating its loss despite better underlying accuracy.

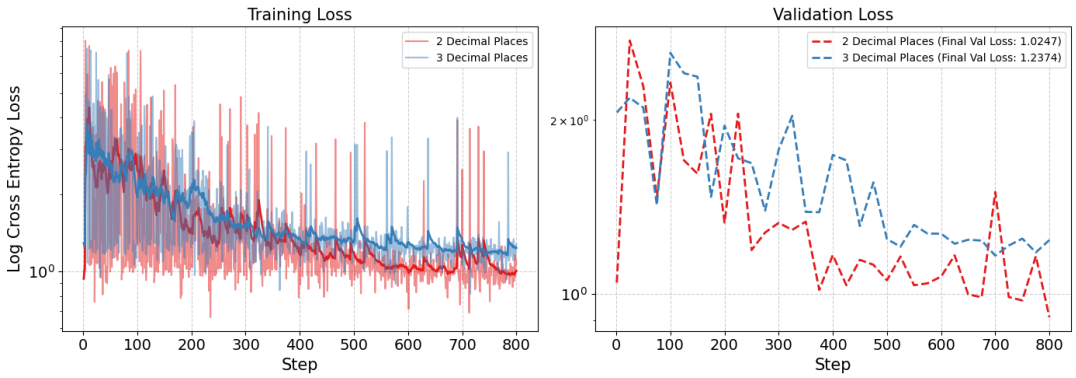


Figure 6: Tuning Decimal Places

7.2 Investigation 2: Learning Rate

While investigating the learning rate, 1×10^{-4} initially yielded the best performance, although it lay at the edge of the search space. To try to identify the true minimum along this hyperparameter axis, an additional run with learning rate 1×10^{-3} was performed, and yielded an even better result, leaving the true minimum unresolved. Due to computational constraints, further exploration was not possible. Similar limitations apply to the LoRA rank and context length results.

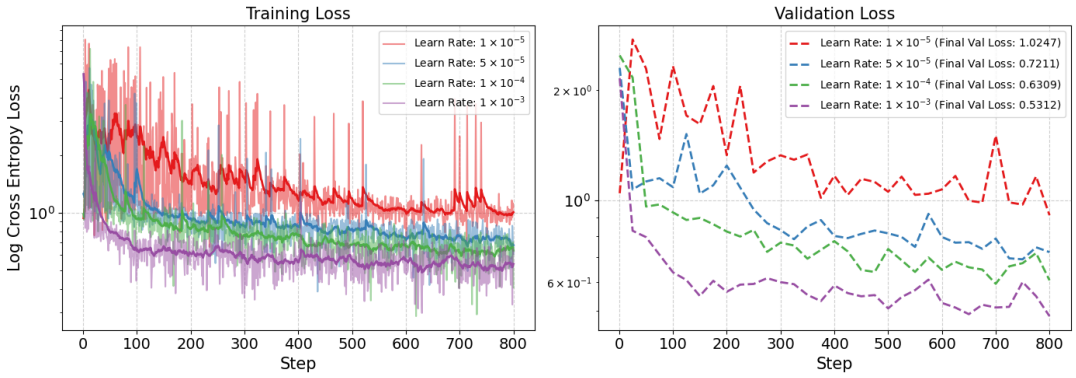


Figure 7: Tuning Learning Rate results: Context 256 and LoRA Rank 4

7.3 Investigation 3: LoRA Rank

Varying the LoRA rank across 2, 4, and 8 had minimal impact on validation loss, with a maximum deviation of just 0.03. This posed the question of whether improvements stemmed from the low-rank matrices or primarily from the unfrozen LM Head biases. To verify this, an additional run with LoRA disabled (updating only LM Head biases) was conducted, resulting in significantly worse performance-reaffirming LoRA's contribution. Ultimately, a rank of 8 yielded the best performance, albeit marginally, and the slight increase in computational cost was considered negligible.

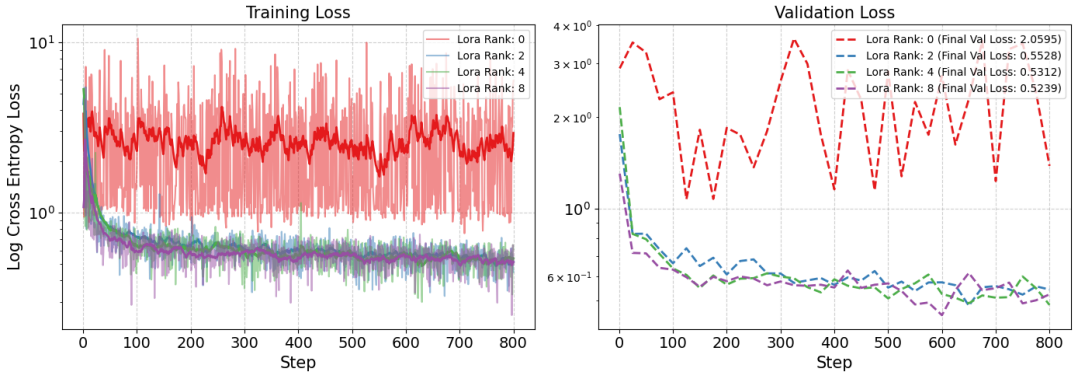


Figure 8: Tuning LoRA Rank: Context, 256 and Learning rate, 1×10^{-3}

7.4 Investigation 4: Context Length

Finally, the impact of context length on training performance was examined. Since this hyperparameter directly relates to the number of data points seen and the computational cost incurred per step, comparisons were made at equivalent compute positions: approximately 800 steps for context length 128, 533 for 512, and 133 for 768. Even after accounting for this, a context length of 768 consistently showed stronger performance and was therefore selected.

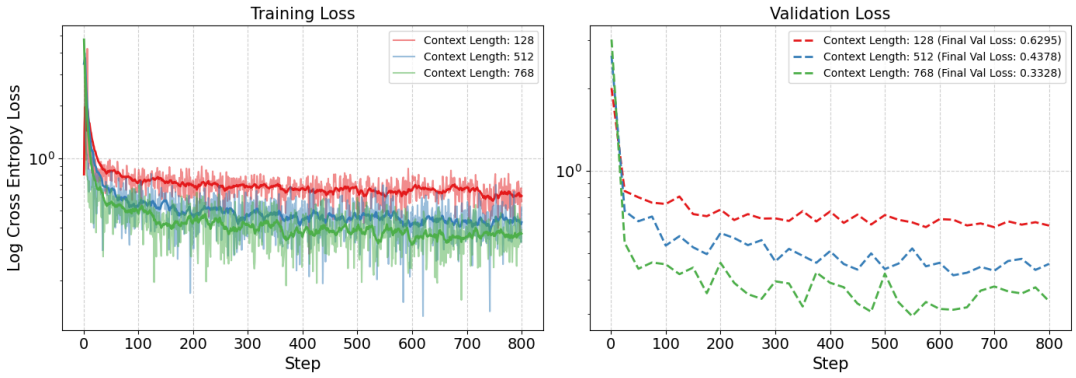


Figure 9: Tuning Context Length: Learning rate, 1×10^{-3} and LoRA Rank, 4

7.5 Tuning Results

The best hyperparameters determined within the constraints were a learning rate of 1×10^{-3}, LoRA rank 8, and context length 768. Overall these tests incurred a cost of 4.437×10^{16} FLOPs (44.37%).

8 Final Model

In section 7, models consistently showed rapid learning within the first 200–400 steps, followed by stagnation in the validation loss. Consequently, although the final model was trained for up to 4000 steps, early stopping (patience: 10) was used to avoid unnecessary computation. This halted training after just 1150 steps at a validation loss of 0.3138, with marginal improvement observed beyond step 600. This led to significantly reduced FLOPs usage compared to the budget

(1.439×10^{16} vs. 4.82×10^{16}). Finally, this model was evaluated using the metrics defined in section 3, which are presented in section 9.

Figure 10: Final Training Metrics

9 Comparison of Metrics

We now compare the performance of the untrained Qwen model, trained (untuned) model (section 6) and final model (section 8) on the test set to determine the success of LoRA's application to time series forecasting.

9.1 Cross Entropy Loss

Table 6 confirms that by training the model using masked self-attention and cross-entropy loss over the output logits, we have improved its ability to capture the underlying distribution of the next immediate token. Although reassuring, it remains a limited metric for evaluating time series forecasting over entire values, let alone extended time periods.

Table 6: Cross-Entropy Loss Comparison Across Model Variants

Model	**Cross-Entropy Loss**
Untrained Qwen	4.7794
Initial Trained (Untuned)	1.0687
Final Trained Model	0.3153

10 Autoregressive Metrics (MAE/MAPE)

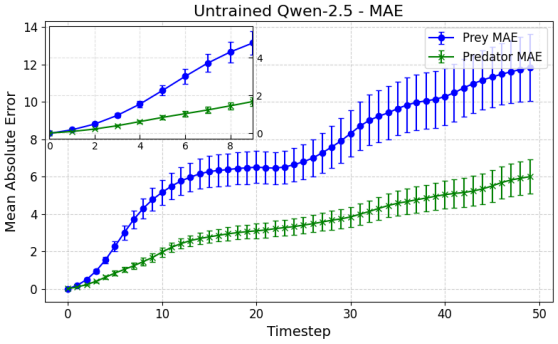
10.1 Untrained Qwen-2.5 Performance

The baseline performance demonstrates little promise for competing with purpose-built time series models. Within just 10 time-steps, the mean percentage errors for prey and predator populations reached $227.15\% \pm 47.44\%$ and $174.20\% \pm 72.82\%$, respectively.

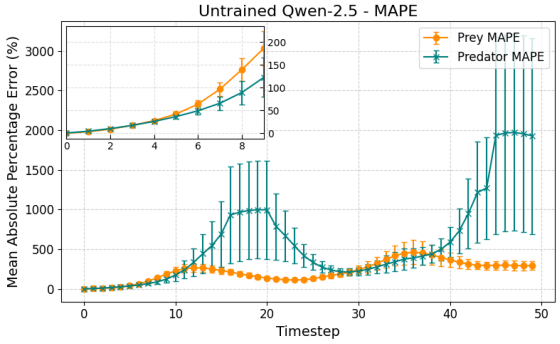
All graphs within Figure 11 show the expected global trend of error increasing with time, a result of the compounding nature of errors in auto-regressive generation. The noticeable oscillatory trends indicate the model's inability to fully capture the series' periodic nature, often favoring a linear prediction as shown in Figure 12c. Notably, the initial performance for the prey population exhibits more stable and less oscillatory percentage error, suggesting that its typically higher magnitude helps the model in preserving periodic structure, supported by differences in predators and prey predictions in Figure 12a.

Table 7: Forecasting metrics across time-steps using MAE and MAPE

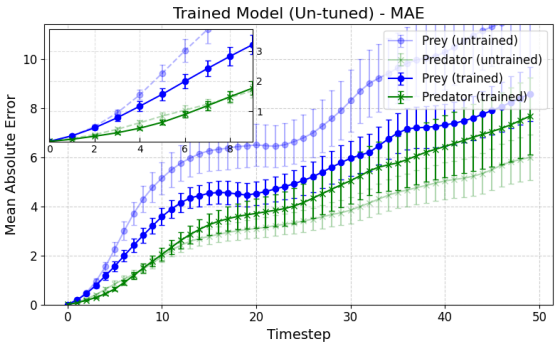
Step	Predator MAE	Prey MAE	Predator MAPE	Prey MAPE
1	0.10±0.01	0.19±0.03	4.86±0.40%	3.72±0.45%
5	0.85±0.12	2.27±0.27	36.8±4.9%	42.3±4.80%
10	1.97±0.25	5.16±0.65	174±72%	227±47%
20	3.11±0.39	6.51±0.86	996±616%	138±22%
30	3.85±0.54	8.32±1.25	224±52%	241±56%
40	5.05±0.75	10.29±1.53	590±191%	365±89%
49	6.01±0.92	11.85±1.78	1928±1234%	292±54%



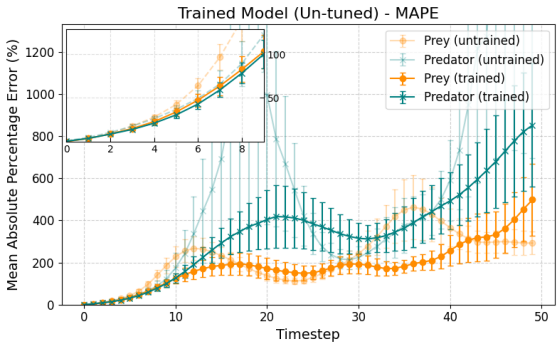
(a) MAE: Baseline Qwen2.5



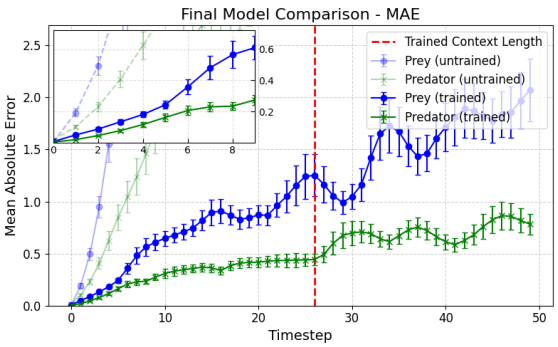
(b) MAPE: Baseline Qwen2.5



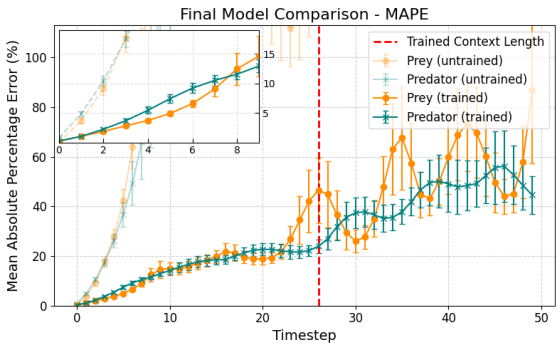
(c) MAE: Default Training



(d) MAPE: Default Training



(e) MAE: Optimised Training



(f) MAPE: Optimised Training

Figure 11: Model performance at all stages

10.2 Untuned, Default Training

The impact of the initial training (800 steps) varies notably between short and long term forecasts, as well as between predator and prey dynamics. For the prey population, forecasting accuracy improved consistently across the entire horizon, with MAE reduced by approximately 25–30%. In contrast, predator forecasts showed only marginal short-term improvement, with performance degrading relative to the baseline beyond 8 time steps.

Despite these differences, both time series show reduced oscillatory error patterns, most evident in Figure 11d, indicating the model's greater capacity to capture periodic structure. The consistent short-term improvements are due to cross-entropy's emphasis on predicting the immediate next token during training..

Furthermore, the greater improvements observed for the prey population further support the idea that exposure to larger fluctuations enables the model to learn more effectively and better constrain its extrapolated forecasts.

Table 8: Forecasting metrics: Untuned training (% improvement)

Step	Predator MAE	Prey MAE	Predator MAPE	Prey MAPE
1	0.08±0.01 (24%)	0.19±0.03 (0.20%)	3.63±0.31% (25%)	3.47±0.41% (6.9%)
5	0.64±0.08 (24%)	1.57±0.20 (31%)	30.48±3.52% (17%)	33.75±4.63% (20%)
10	2.05±0.26 (-4.2%)	3.59±0.35 (30%)	126.61±22.43% (27%)	123.87±24.90% (46%)
20	3.72±0.60 (-20%)	4.53±0.48 (30%)	403.71±138.42% (60%)	171.64±50.66% (-25%)
30	5.05±0.94 (-31%)	5.97±0.72 (28%)	314.57±72.40% (-41%)	192.88±46.18% (20%)
40	6.45±1.27 (-28%)	7.33±0.86 (29%)	492.19±135.10% (17%)	256.89±73.16% (30%)
49	7.69±1.57 (-28%)	8.59±1.10 (28%)	851.15±291.62% (56%)	497.54±171.19% (-70%)

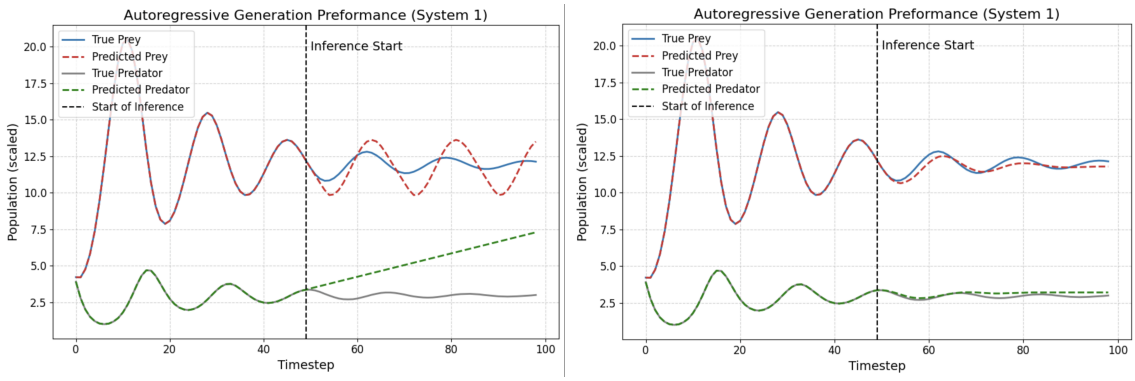
10.3 Final Model Performance

The final trained model demonstrates strong evidence for the effectiveness of LoRA in fine-tuning Qwen for time series forecasting. Improvements are observed consistently for all metrics across both predator and prey populations and short and long forecasts. Notably, all evaluated time points show at least a 70.3% improvement relative to the untrained baseline (Table 9). At a time step of 10, the model now achieves mean percentage errors of $14.93 \pm 2.63\%$ for predators and $14.15 \pm 1.50\%$ for prey, highlighting LLMs' future potential to rival dedicated time series models.

A notable observation is the re-emergence of oscillatory prediction errors once generation exceeds the model's trained context length (768 tokens). While the model demonstrates strong generalisation within its training window, its ability to extrapolate periodic patterns beyond this range is reduced, revealing a limitation in its capacity. This effect is subtly visible in the trained examples of Figure 12.

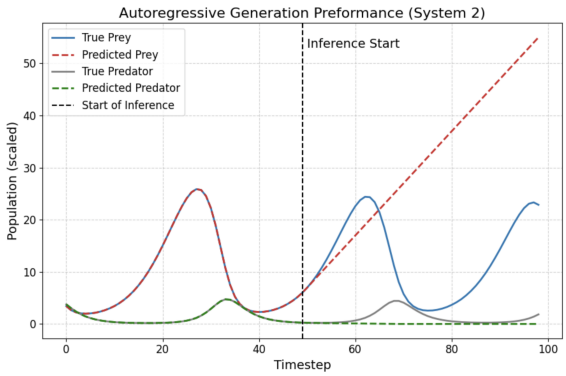
Table 9: Forecasting metrics: Tuned training (% improvement)

Step	Predator MAE	Prey MAE	Predator MAPE	Prey MAPE
1	0.05±0.01 (74%)	0.02±0.00 (81%)	1.07±0.24% (71%)	1.05±0.14% (79%)
5	0.24±0.03 (89%)	0.16±0.02 (81%)	4.92±0.43% (88%)	7.43±0.77% (80%)
10	0.65±0.07 (88%)	0.31±0.05 (84%)	14.93±2.63% (93%)	14.15±1.50% (92%)
20	0.87±0.09 (87%)	0.42±0.05 (87%)	18.89±2.32% (86%)	22.69±2.54% (98%)
30	1.05±0.12 (87%)	0.70±0.12 (82%)	25.96±4.62% (89%)	37.39±6.26% (83%)
40	1.72±0.24 (83%)	0.61±0.07 (88%)	59.83±14.88% (84%)	49.05±11.12 (92%)
49	2.07±0.30 (83%)	0.78±0.10 (87%)	86.81±29.38% (70%)	44.55±7.70% (98%)

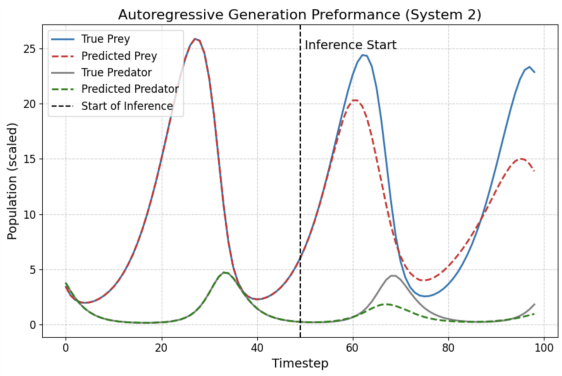


(a) Baseline 1

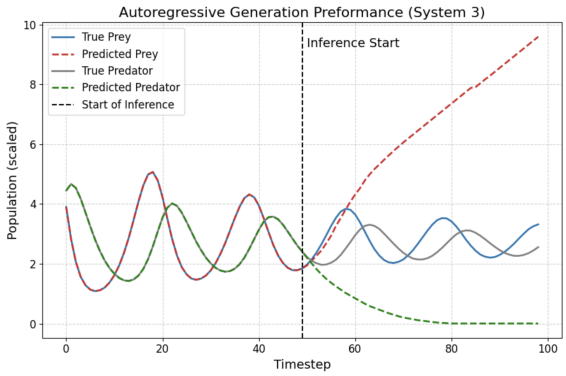
(b) Trained 1



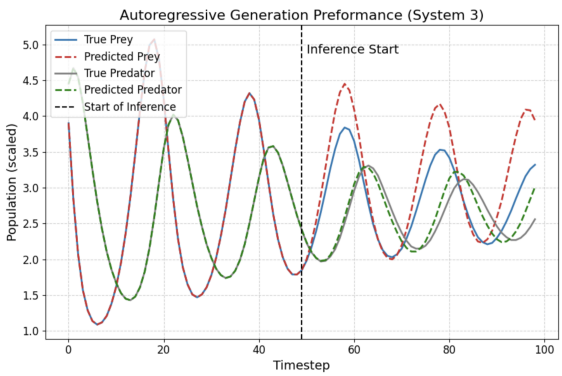
(c) Baseline 2



(d) Trained 2



(e) Baseline 3



(f) Trained 3

Figure 12: Forecasted series from baseline and fully trained models

11 Final Flops Budget

This section details the actual FLOPs used: 6.576×10^{16}, or 65.76% of the allocated budget. The reduction is largely due to early stopping, which lowered the final model's training cost by 70.14% compared to the forecast in subsection 4.6.

Table 10: Final FLOPs Breakdown Across Training and Evaluation

Stage	**Training**	**Evaluation**	**Total (% of Budget)**
Baseline Eval (Cross Entropy)	0	1.81×10^{14}	1.81×10^{14} (0.18%)
Baseline Eval (Autoregressive)	N/A	N/A	N/A
Overfitting Test	3.62×10^{14}	9.06×10^{13}	4.53×10^{14} (0.45%)
Default Model Training	5.79×10^{15}	3.59×10^{14}	6.16×10^{15} (6.16%)
Default Eval (Cross Entropy)	0	2.08×10^{14}	2.08×10^{14} (0.21%)
Default Eval (Autoregressive)	N/A	N/A	N/A
Hyperparameter Tuning	3.88×10^{16}	5.56×10^{15}	4.43×10^{16} (43.37%)
Final Model Training	1.28×10^{16}	1.36×10^{15}	1.41×10^{16} (14.39%)
Final Eval (Cross Entropy)	0	2.77×10^{14}	2.77×10^{14} (0.28%)
Final Eval (Autoregressive)	N/A	N/A	N/A
Total	$\mathbf{5.77 \times 10^{16}}$	$\mathbf{8.04 \times 10^{15}}$	$\mathbf{65.76 \times 10^{17}}$ (65.76%)

12 Summary

This work demonstrates the ability to significantly enhance the Qwen2.5-Instruct model for time series forecasting despite a strict computational budget of 1×10^{17} FLOPs. By step 10, the model achieves reductions in mean absolute percentage error of 88% for predators and 80% for prey. These improvements were enabled through targeted hyperparameter tuning and the use of strategies such as early stopping, allowing the full training process to complete within just 65% of the available budget.

Future work with greater computational allowance should explore finer and more extensive hyperparameter sweeps, particularly in underexplored areas such as the preprocessing strategy. Additional improvements could be achieved through custom loss functions designed to better capture long-range forecasting performance, defining a restricted output vocabulary to exclude text-based tokens, and incorporating learning rate schedulers to balance convergence with training stability and depth.

12.1 Declaration of Use of Autogeneration Tools

This report made use of Large Language Models (LLMs), to assist in the development of the project. These tools have been employed for assisting:

- Formatting plots to enhance presentation quality.
- Generating docstrings for the repository's documentation.
- Performing iterative changes to already defined code.
- Debugging code and identifying issues in implementation.
- Latex formatting for the report.
- Identifying spelling and punctuation inconsistencies within the report.
- Suggesting more concise phrasing to reduce the word count.

References

[1] Nate Gruver et al. *Large Language Models Are Zero-Shot Time Series Forecasters*. 2024. arXiv: 2310.07820 [cs.LG]. URL: https://arxiv.org/abs/2310.07820.

[2] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[3] Qwen et al. *Qwen2.5 Technical Report*. 2025. arXiv: 2412.15115 [cs.CL]. URL: https://arxiv.org/abs/2412.15115.

[4] Noam Shazeer. *GLU Variants Improve Transformer*. 2020. arXiv: 2002.05202 [cs.LG]. URL: https://arxiv.org/abs/2002.05202.

[5] Jianlin Su et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. arXiv: 2104.09864 [cs.CL]. URL: https://arxiv.org/abs/2104.09864.

[6] Biao Zhang and Rico Sennrich. *Root Mean Square Layer Normalization*. 2019. arXiv: 1910.07467 [cs.LG]. URL: https://arxiv.org/abs/1910.07467.