



S2: Advanced Statistical Methods

Yuchen Mao (ym429)

Department of Physics, University of Cambridge

April 6, 2025

Word count: 2998

Contents

1	Introduction	2
2	Dataset	2
3	LLMTime Preprocessor	3
4	FLOPS Calculation	4
4.1	Model Components and Assumptions	4
4.2	FLOPs Estimation Approach	5
4.3	Total FLOPS Estimation	9
4.4	Experiment Planning within FLOPS Budget	9
5	Experimental Setup	10
6	Results	13
6.1	Untrained Baseline Performance (Q2b)	13
6.2	Initial LoRA Training (Q3a)	13
6.3	Hyper-parameter Search (Q3b)	14
6.4	Final Training with Optimal Configuration (Q3c)	15
6.5	FLOPS Usage Summary	15
7	Discussion	16
8	Conclusion	20
A	AI Usage	21
B	FLOPS Breakdown	22
C	Test Set Prediction Examples	22
D	Additional LLMTime Samples	24

1 Introduction

This report describes the fine-tuning of the **Qwen-2.5-0.5B-Instruct** large language model (LLM) — originally trained for natural language tasks — for forecasting predator-prey time series. This adaptation was undertaken within a challenging constraint: a strict computational budget of 1×10^{17} Floating Point Operations (FLOPS) covering all experimental stages. To achieve this, we employed the LLM-Time preprocessing scheme [4] for efficient numerical sequence representation and Low-Rank Adaptation (LoRA) [7] for parameter-efficient fine-tuning. Through systematic experimentation, including hyper-parameter optimization guided by careful FLOPS calculation and planning, we demonstrate a significant improvement in the model’s forecasting accuracy compared to its untrained performance. Overall, this work documents the methodology, experimental progression, key findings, and showcased the overall effectiveness of adapting a tiny LLM for this quantitative task under significant resource constraints. The code is available at our [GitLab repository](#), with a tutorial and documentation at [GitHub Pages](#).

2 Dataset

This study uses a dataset of 1000 distinct Lotka-Volterra systems, simulating predator-prey population dynamics. Each system is a time series data of 100 time-steps, with both predator and prey population levels. Examples are visualized in Figure 1. The primary task is to fine-tune the pre-trained **Qwen-2.5-0.5B-Instruct** model using this dataset, such that the model is able to learn the underlying temporal patterns governing these interactions and accurately forecast future population levels based on sequences of past observations.

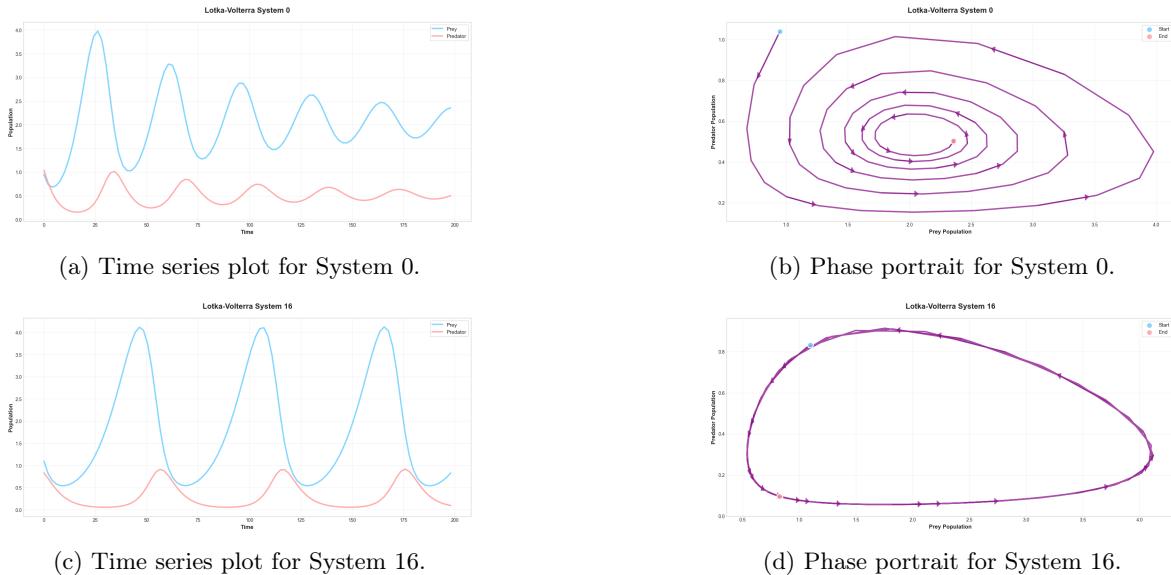


Figure 1: Visualization of predator-prey dynamics for two example systems (0 and 16) from the dataset. Subfigures (a) and (c) show population levels (Prey in blue, Predator in orange) over 100 time steps. Subfigures (b) and (d) display the corresponding phase plots, illustrating the cyclical relationship between prey and predator populations.

3 LLMTTime Preprocessor

Fine-tuning LLMs like **Qwen-2.5-0.5B-instruct** on numerical time-series data presents a challenge: standard tokenization schemes are inefficient for floating-point numbers. For instance, Qwen’s tokenizer represent 0.12345 as a sequence of individual digit and punctuation tokens [0, ., 1, 2, 3, 4, 5], inflating sequence length and computational cost. To address this, we implemented the preprocessing method proposed by LLMTTime [4], designed to create a more compact and LLM-friendly representation.



Figure 2: Visual workflow of the LLMTTime preprocessing: raw data undergoes scaling (to [0, 10]), rounding (2 decimal places), and formatting into a structured string suitable for LLM input.

The process, illustrated in Figure 2, involves three main steps:

1. **Scaling:** Values are scaled to map the majority (up to the 99.7th percentile) into the range [0, 10] using a calculated scaling factor (Equation 1). While this aims to achieve a compact representation, it unfortunately imposes an unnatural upper limit on the range of predictable values, as discussed further in Section 7.

$$\text{scaler} = \frac{10}{\text{percentile}_{99.7}(\text{data})}. \quad (1)$$

-
2. **Rounding:** Scaled values are rounded to two decimal places (e.g., X.XX) to standardize the format and reduce token usage. Note that this operation results in information loss, which will be assessed later in this section.
3. **Formatting:** The processed numerical pairs (prey, predator) are formatted into a single string. Values at the same timestamp are comma-separated, and different timestamps are semicolon-separated (e.g., $p_t, h_t; p_{t+1}, h_{t+1}; \dots$). This preserves temporal structure while being easily tokenizable.

To quantify the impact of scaling and rounding, a round-trip evaluation (scaling, rounding, formatting and decoded back to time-series data) was performed. As shown in Table 1, the Mean Absolute Error (MAE) between the original and recovered values is only 0.002167. This confirms that the pre-processing introduces minimal information loss while significantly reducing sequence length, enabling more efficient training and inference. Implementation details can be found in preprocessor python file¹ and the accompanying notebook² for tutorial.

Table 1: Round-trip evaluation assessing information loss from the LLMTTime preprocessing method (scaling, rounding, formatting and decoded back to time-series data). The Mean Absolute Error between the original and recovered time-series data is 0.002167. This minimal difference between original and recovered values hints that the preprocessing preserves essential signal characteristics despite significant data condensation required for Large Language Model compatibility.

Prey Values					Predator Values				
Time	Original	Processed	Recovered	Difference	Time	Original	Processed	Recovered	Difference
0	0.949917	1.20	0.951563	0.001646	0	1.040624	1.31	1.038790	-0.001834
1	0.740551	0.93	0.737461	-0.003090	1	0.779542	0.98	0.777110	-0.002432
2	0.682246	0.86	0.681954	-0.000292	2	0.564390	0.71	0.563008	-0.001382
3	0.716674	0.90	0.713672	-0.003002	3	0.407644	0.51	0.404414	-0.003230
4	0.824511	1.04	0.824688	0.000177	4	0.300283	0.38	0.301328	0.001045
5	1.004565	1.27	1.007071	0.002506	5	0.229832	0.29	0.229961	0.000129
6	1.262928	1.59	1.260821	-0.002107	6	0.186162	0.23	0.182383	-0.003779
7	1.605509	2.02	1.601798	-0.003711	7	0.162235	0.20	0.158594	-0.003642
8	2.031105	2.56	2.030001	-0.001104	8	0.154365	0.19	0.150664	-0.003700
9	2.522882	3.18	2.521642	-0.001240	9	0.161886	0.20	0.158594	-0.003292

4 FLOPS Calculation

Training modern LLMs requires substantial computational resources. However, due to the strict budget of 1×10^{17} FLOPS imposed by this coursework for all reported experiments, careful planning and efficient experimentation are required. This section details the algorithm used to estimate the FLOPS cost for fine-tuning the **Qwen-2.5-0.5B-Instruct** model with LoRA, allowing us to manage experiments within the budget.

4.1 Model Components and Assumptions

The **Qwen-2.5-0.5B-Instruct** model is composed of stacked decoder layers. Each layer primarily contains a Grouped Query Attention block and a Multilayer Perceptron block, both utilizing RM-SNorm [13] and residual connections [5]. The MLP employs SwiGLU activation [11]. For fine-tuning,

¹see [src/preprocessor.py](#) file

²see [notebooks/1_dataset_preprocess.ipynb](#) file

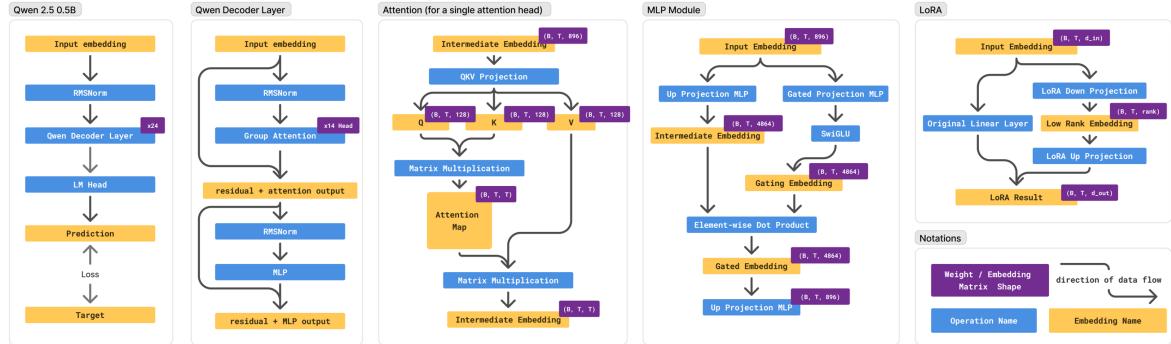


Figure 3: Architecture of Qwen 2.5 0.5B Model. The diagram details the overall model structure, including the Qwen Decoder Layer (stacked 24 times), single-head attention mechanism, MLP module, and LoRA (Low-Rank Adaptation) integration. Each component is annotated with data flow directions, embedding names, and tensor shapes to illustrate how input embeddings are transformed into predictions through normalization, multi-head attention, MLP gating, and low-rank updates.

LoRA [7] are applied to the query and key projection matrices within the attention mechanism. See Figure 3.

For FLOPS calculation, we followed the coursework guidelines and used Table 3 for the flops costs on each primitive operations. The key hyper-parameters influencing FLOPS are batch size (B), sequence length (S), model dimensions (Table 2), and LoRA rank (r). The specific values used are listed in Table 2.

Symbol	Description	Value
B	Batch Size	4
S	Sequence Length	variable (128, 512, 768)
H	Hidden/Model Dimension	896
H_{mlp}	MLP Intermediate Hidden Dimension	4864
N_{heads}	Number of Attention Heads	14
H_{head}	Dimension per Attention Head (H/N_{heads})	128 ³
V_{size}	Vocabulary Size	151936
N_{layers}	Number of Qwen Decoder Layers	24
r	LoRA Rank	variable (2, 4, 8)
α	LoRA Scaling Factor	r (common practice)

Table 2: Notation for FLOPS Calculation.

4.2 FLOPs Estimation Approach

We adopted a component-based approach, calculating the forward pass FLOPS for each distinct operation within a decoder layer: RMSNorm, Attention, MLP, LoRA, and Residual connections. The final LM head projection and loss computation were also included.

³We assume the attention head dimension to be 128 (896 / 7), based on the original Qwen architecture, which uses a grouped query attention block with 2 groups and 14 heads. This results in 7 heads per group, with each group splitting the embedding dimension of 896 evenly, yielding a head dimension of 128.

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

Table 3: FLOPS Accounting for Primitives (per coursework Table 1).

The detailed breakdown and formulas for each component are implemented in the script⁴ and explained in the accompanying notebook⁵.

Standard Multi-headed Attention (*FlopsAttention*) The attention module is the fundamental component of the Transformer architecture and modern LLMs. Given an input embedding $X \in \mathbb{R}^{B \times S \times H}$, it is first projected to Q, K, V embeddings using weight matrices $W_q, W_k, W_v \in \mathbb{R}^{H \times (N_{\text{heads}} H_{\text{head}})}$.

$$Q = XW_q, \quad K = XW_k, \quad V = XW_v \quad (2)$$

These matrices $Q, K, V \in \mathbb{R}^{B \times S \times (N_{\text{heads}} H_{\text{head}})}$ are then typically reshaped to N_{heads} individual heads, yielding $Q_i, K_i, V_i \in \mathbb{R}^{B \times S \times H_{\text{head}}}$ for $i = 1, \dots, N_{\text{heads}}$. Rotary Positional Embeddings (RoPE) [12] are applied to Q_i and K_i . For the FLOPS of RoPE, we approximate it as simple element-wise addition operations.

The standard scaled dot-product attention is then computed per head:

$$\text{Attention}(Q_i, K_i, V_i) = \text{Softmax}\left(\frac{Q_i K_i^T}{\sqrt{H_{\text{head}}}}\right) V_i, \quad (3)$$

where a causal mask is applied on $Q_i K_i^T$. The outputs from all heads are then concatenated, $O = \text{Concat}(\text{Attention}_1, \dots, \text{Attention}_{N_{\text{heads}}}) \in \mathbb{R}^{B \times S \times (N_{\text{heads}} H_{\text{head}})}$, and projected back to the hidden dimension using $W_o \in \mathbb{R}^{(N_{\text{heads}} H_{\text{head}}) \times H}$:

$$X_{\text{Attn}} = O W_o \quad (4)$$

FLOPS are detailed in Table 4.

RMSNorm (*FlopsRMSNorm*) normalizes input $x \in \mathbb{R}^{B \times S \times H}$ across the hidden dimension H with the root mean square of the embedding, scaled by a learnable parameter γ . Mathematically, it can be represented as:

$$\text{RMS}(x) = \sqrt{\frac{1}{H} \sum_{i=1}^H x_i^2 + \epsilon} \quad (5)$$

$$y_i = \frac{x_i}{\text{RMS}(x)} \gamma_i \quad (6)$$

⁴[src/get_flops.py](#)

⁵[notebooks/2_flops_calculation.ipynb](#)

Operation	Flops
To Q, K, V	$Flops_{Mul} \cdot BSH \cdot 3N_h H_{hd} + Flops_{Add} \cdot BS(H - 1) \cdot 3N_h H_{hd}$
Add ROPE	$Flops_{Add} \cdot BS \cdot 2N_h H_{hd}$
QK^T	$Flops_{Mul} \cdot BN_h SH_{hd} S + Flops_{Add} \cdot BN_h S(H_{hd} - 1)S$
Scale by $1/\sqrt{H_{head}}$	$Flops_{SQRT} + Flops_{DIV} \cdot BN_h SS$
Add Mask	$Flops_{Add} \cdot BN_h SS$
Softmax	$(Flops_{EXP} + Flops_{Add} \times (S - 1) + Flops_{DIV}) \cdot BN_h SS$
Multiply by V	$Flops_{MUL} \cdot BN_h SSH_{hd} + Flops_{ADD} \cdot BN_h S(S - 1)H_{hd}$
Output Projection (W_o)	$Flops_{MUL} \cdot BS(N_h H_{hd})H + Flops_{ADD} \cdot BS(N_h H_{hd} - 1)H$

Table 4: FLOPS Breakdown for Attention Mechanism Layer (Standard MHA). $N_h = N_{\text{heads}}$, $H_{hd} = H_{\text{head}}$.

Operation	Flops
Square elements	$Flops_{Mul} \cdot BSH$
Mean over H	$Flops_{Add} \cdot BS(H - 1) + Flops_{Div} \cdot BS$
Add epsilon	$Flops_{Add} \cdot BS$
Square root	$Flops_{Sqrt} \cdot BS$
Reciprocal (rsqrt)	$Flops_{Div} \cdot BS$
Normalize input	$Flops_{Mul} \cdot BSH$
Scale by γ	$Flops_{Mul} \cdot BSH$

Table 5: FLOPS Breakdown for RMSNorm Layer.

MLP (Flops_{MLP}) Qwen’s MLP module use a SwiGLU activation with two linear layers [11]. For an input $x \in \mathbb{R}^{B \times S \times H}$, it’s first projected to an intermediate dimension H_{mlp} using two weights $W_{up}, W_{gate} \in \mathbb{R}^{H \times H_{MLP}}$. The gated projection is activated with SiLU:

$$\text{SiLU}(z) = z \cdot \text{sigmoid}(z),$$

and element-wise multiplied with the up projected embedding. These operation can be seen as the SwiGLU activation. A down-projection with $W_{down} \in \mathbb{R}^{H_{MLP} \times H}$ restores the embedding to the hidden dimension:

$$X_{MLP} = [(XW_{up}) \odot \text{SiLU}(XW_{gate})] W_{down} \quad (7)$$

FLOPS are in Table 6.

LoRA [7] (Flops_{LoRA}) is a parameter-efficient fine-tuning technique that freezes the full LLM model while injecting two low-rank, trainable weight matrices into each q and k projection of the multi-headed attention layers. These matrices effectively mimic the weight updates from full fine-tuning, allowing efficient adaptation by tuning only a fraction of the parameters. For an input x , the

Operation	Flops
Gate Projection (W_{gate})	$Flops_{Mul} \cdot BSH_{mlp} + Flops_{Add} \cdot BS(H - 1)H_{mlp}$
SiLU Activation	$(Flops_{Neg} + Flops_{Exp} + Flops_{Add} + Flops_{Div} + Flops_{Mul}) \cdot BSH_{mlp}$
Up Projection (W_{up})	$Flops_{Mul} \cdot BSH_{mlp} + Flops_{Add} \cdot BS(H - 1)H_{mlp}$
Element-wise Mult	$Flops_{Mul} \cdot BSH_{mlp}$
Down Projection (W_{down})	$Flops_{Mul} \cdot BSH_{mlp}H + Flops_{Add} \cdot BS(H_{mlp} - 1)H$

Table 6: FLOPS Breakdown for MLP Layer with SwiGLU.

output of a LoRA-adapted linear layer is given by:

$$y = xW + xAB(\alpha/r), \quad (8)$$

where $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{r \times d_{out}}$ are the trainable low-rank matrices, r denotes the rank, α is a scaling factor, d_{in} is the input dimension, and d_{out} is the output dimension. Table 7 details the FLOPS for the LoRA path ($xAB(\alpha/r)$), representing the additional computational cost to the Q/K projection layer. For Q/K projections, we have $d_{in} = H$ and $d_{out} = H_{head}N_{heads}$.

Operation	Flops
Down Projection (A)	$Flops_{Mul} \cdot BSHr + Flops_{Add} \cdot BS(H - 1)r$
Up Projection (B)	$Flops_{Mul} \cdot BSr(N_hH_{hd}) + Flops_{Add} \cdot BS(r - 1)(N_hH_{hd})$
Scaling (α/r)	$Flops_{Mul} \cdot BS(N_hH_{hd})$
Addition to Original	$Flops_{Add} \cdot BS(N_hH_{hd})$

Table 7: FLOPS Breakdown for LoRA

Residual Connection ($Flops_{\text{Residual}}$) Adds the input of a block to its output:

$$X_{out} = X_{in} + \text{Block}(X_{in}).$$

FLOPS are in Table 8.

Operation	Flops
Element-wise Addition	$Flops_{Add} \cdot BSH$

Table 8: FLOPS Breakdown for Residual Connection.

LM Head and Loss ($Flops_{\text{LM Head}} + Flops_{\text{Loss}}$) represents the last layer of Qwen model, which projects the final decoder layer's output $X_{final} \in \mathbb{R}^{B \times S \times H}$ to vocabulary size V_{size} , applies Softmax, and computes Cross-Entropy loss. FLOPS are in Table 9.

Operation	Flops
LM Head Projection	$Flops_{Mul} \cdot BSHV_{size} + Flops_{Add} \cdot BS(H - 1)V_{size}$
LM Head Bias	$Flops_{Add} \cdot BSV_{size}$
Softmax	$(Flops_{Exp} + Flops_{Add} \times (V_{size} - 1) + Flops_{Div}) \cdot BSV_{size}$
Log Loss (per token)	$(Flops_{Log} + Flops_{Mul}) \cdot BS$

Table 9: FLOPS Breakdown for LM Head and Final Operations.

4.3 Total FLOPS Estimation

To conclude, the FLOPS for one forward pass through a single Qwen decoder layer is:

$$\begin{aligned} Flops_{Layer} = & Flops_{RMSNorm} + (Flops_{Attention} + 2 \times Flops_{LoRA}) \\ & + Flops_{Residual} + Flops_{RMSNorm} + Flops_{MLP} \\ & + Flops_{Residual} \end{aligned} \quad (9)$$

Total inference FLOPS for the model is then:

$$Flops_{Inference} = N_{layers} \times Flops_{Layer} + Flops_{LM\ Head\ +\ Loss} \quad (10)$$

Total training step FLOPS (forward + backward):

$$Flops_{Training\ Step} \approx 3 \times Flops_{Inference} \quad (11)$$

This estimate (Equation 11) guided our experimental planning.

4.4 Experiment Planning within FLOPS Budget

Equation 11 allowed calculation of FLOPS per step for various configurations (Table 10).

Context Length (S)	Rank (r)	FLOPS per Step	Max Steps (Budget)
128	2	1.90×10^{12}	52,743
128	4	1.90×10^{12}	52,738
128	8	1.90×10^{12}	52,728
512	2	8.00×10^{12}	12,500
512	4	8.00×10^{12}	12,499
512	8	8.00×10^{12}	12,496
768	2	1.24×10^{13}	8,054
768	4	1.24×10^{13}	8,053
768	8	1.24×10^{13}	8,052

Table 10: Calculated FLOPS per training step (Batch Size $B = 4$) and theoretical maximum optimizer steps within the 1×10^{17} FLOPS budget.

Overall, the total budget allows us to run around 10,000 runs for $B = 4$, and needs to accommodate initial tests, grid searches (12 runs), and final training. Therefore, we allocated steps per phase (Table 11), using short runs for exploration and reserving budget for the final model. This FLOPS-aware planning before training was essential to the success of the final resulting model.

Training Phase	Planned Optimizer Steps
Initial Training / Smoke Test	1,000
Grid Search Runs (per run) ^a	500 ^b
Final Model Training	$\approx 3,000^c$

Table 11: Planned optimizer step allocation per training phase.

^aApplies to each grid search experiment (learning rate, rank) and search over context length.

^bThe design of 500 steps allows the model to see the entire dataset at least 1 time (≈ 1.2 epoch).

^cFlexible, utilizing remaining budget after searches (estimated 2k-4k steps).

5 Experimental Setup

Our experimental process followed four main phases, designed for systematic exploration within the FLOPS budget: 1) Initial baseline training, 2) Grid search over LoRA rank (r) and learning rate (η), 3) Search over context length (S), and 4) Final training with the optimized configuration.

Data Handling The Lotka-Volterra dataset (Section 2) was split into training (80%), validation (10%), and test (10%) sets. Each time series was processed using the LLMTTime method (Section 3) into a token sequence. Due to model input limits, these sequences (≈ 1000 tokens/system) were chunked into smaller segments of context length $S \in \{128, 512, 768\}$. We used a sliding window with a stride of 256 tokens to generate overlapping chunks, maximizing data use.

A critical issue encountered was input padding. The Qwen model requires left-padding, conflicting with the right-padding provided in the `lora_skeleton` code. Right-padding disrupted model behaviour (affecting the causal attention mask and positional encoding interpretation), leading to non-numerical outputs (Figure 4). As this was only discovered after the full training, a workaround was implemented: only full-length sequence chunks (exactly matching S and requiring no padding) were used for training and evaluation. Shorter chunks, mainly from the end of time series, were discarded. This ensured correct model operation without padding but potentially excluded some data, a limitation discussed in Section 7.

Training and Evaluation All models were trained using the AdamW optimizer [9]. During training, performance was evaluated on the validation set every 50 optimizer steps recording the validation loss. The model checkpoint with the lowest validation loss was saved and used for final evaluation. At all cases, the validation set was used exclusively for model selection, while only the untrained 6.1, initial 6.2, and final 6.4 models were evaluated on the test set, which was never used for parameter tuning.

Final performance was measured using Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the model’s predicted numerical sequences and the ground truth sequences. Note that the MSE and MAE are calculated on decoded time-series data, not on the tokens. Notably, the MSE and MAE show little difference at the first prediction step. However, as prediction

Effect of Tokenization Padding Methods on Qwen Model

Right Padded

Input:

16 13 16 23 11 16 13 17 23 26 ... 151643 151643 151643

Generated text:

Natural language instead of numerical tokens!

Human can not only generate the correct answer but also ensure that the answer is correct. This is a very powerful tool for learning and improving.

In addition to this, it is also important to understand the underlying principles of the problem and how to apply them to solve it. This requires a lot of practice and repetition...

✗ Error

No Padding

Original input length: 128

Input:

16 13 16 23 11 16 13 23 ... 26 16 13 15 15 11 16 13

Generated text:

1;5.00,0.44;5.68,0.47;6.36,0.50;6.94,0.53;7.52,0.56;8.10,0.59;8.68,0.62;9.26,0.65;9.84,0.68;10.42,0.

✓ Good

Left Padded

Input:

151643 151643 151643 ... 16 13 16 23 11 16 13 17 23 26

Generated text:

1;4.47,0.46;5.43,0.45;5.43;6.45,0.44;6.45;7.45,0.43;7.45;8.45,0.42;8.42;9.42,0.39;9.42;10.42,0.38;10

✓ Good

Figure 4: Effect of padding on Qwen output. Right-padding (Top) causes failure. Using only full sequences (Middle) or correct left-padding (Bottom) yields expected numerical format.

errors accumulate over time in autoregressive models, evaluating performance over multiple steps provides a more sensitive measure of the model's capability. That said, if too many prediction steps are considered, even well-performing models can degrade significantly due to accumulated errors, making them appear indistinguishable from poorer models. To strike a balance, we conducted an initial analysis (Figure 5) to determine a sensible number of future steps for evaluation. Based on this, we found that using five prediction steps ($\approx 50 - 60$ tokens)—averaging the MSE and MAE across them—offers a meaningful and stable metric for comparing model performance.

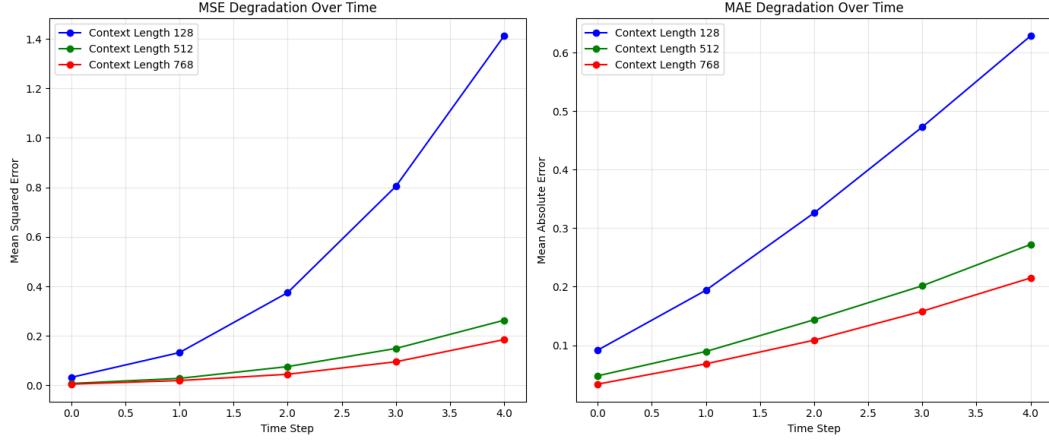


Figure 5: Example of increasing prediction error (e.g., MAE) over successive prediction time steps for different configurations. Note that at the fifth time-stamp, there is already a significant enough gap between the result of different context-length evaluation, thus motivating the 5-step average evaluation approach.

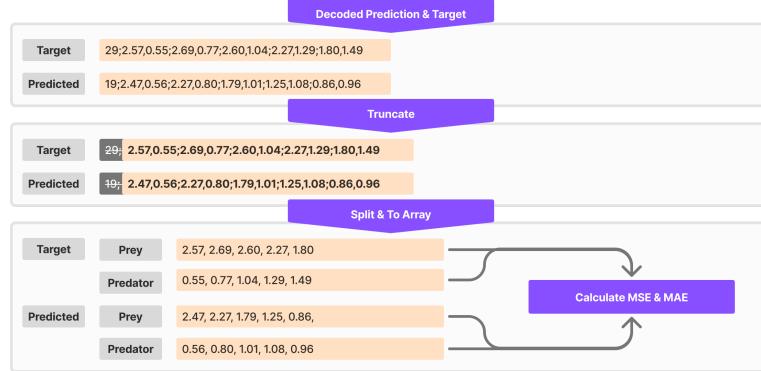


Figure 6: Model Evaluation Process: Given the model's decoded predictions and target data, only the first five complete time steps are retained by truncating from the first to the sixth semicolon. The resulting text is then converted into numerical arrays and compared against the target values. Finally, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are computed to assess prediction accuracy.

6 Results

This section presents the performance evaluation of the `Qwen` model at different stages. Performance is primarily assessed using Mean Squared Error (MSE) and Mean Absolute Error (MAE) on the test set, averaged over a 5-step prediction as described in Section 5. Table 12 summarizes the key metrics across stages.

6.1 Untrained Baseline Performance (Q2b)

Initially, the pre-trained `Qwen` model was evaluated without any fine-tuning (zero-shot) to establish a baseline. As shown in Table 12 (Untrained), the performance was poor, with high MSE and MAE values. This is expected, given the model’s pre-training on natural language. A noticeable trend was better performance (lower errors) with longer context lengths (S), suggesting some inherent ability to leverage longer history, even if poorly. Visualizations (Figure 17 in Appendix) confirm the lack of task understanding, with predictions often being constant or near-constant values, failing entirely to capture the cyclical dynamics of the Lotka-Volterra systems.

Context (S)	Untrained		Initial Training		Final Training	
	MSE	MAE	MSE	MAE	MSE	MAE
128	0.5510	0.3427	0.1356	0.1746	0.0263	0.0726
512	0.1048	0.1509	0.0362	0.0834	0.0028	0.0240
768	0.0700	0.1167	0.0297	0.0651	0.0020	0.0191

Table 12: Test set performance (averaged over 5 prediction steps) across training stages. Initial training: 1000 steps ($S = 512, \eta = 10^{-4}, r = 4$). Final training: Optimized ($\eta = 10^{-4}, r = 8, S = 768$).

6.2 Initial LoRA Training (Q3a)

The model was then fine-tuned for 1000 steps using default hyper-parameters ($S = 512, \eta = 10^{-4}, r = 4$). LoRA is applied to all q and k layers of attention module. Therefore, the parameters that are been tuned are all the `model.model.layers[i].self_attn.q_proj.A` and `model.model.layers[i].self_attn.q_proj.B`, for $i \in \{0, \dots, 23\}$ and `model.model.lm_head.bias`.

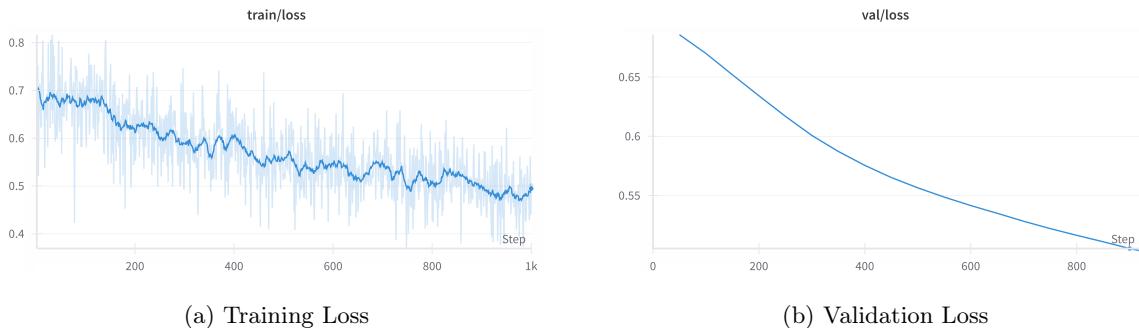


Figure 7: Loss curves during initial 1000 steps of fine-tuning (default hyperparameters).

Both the training and validation losses decreased consistently (Figure 7), indicating successful learning. Test set performance improved dramatically compared to the baseline (Table 12, Initial Training), with MSE and MAE reducing significantly for the trained context length ($S = 512$) and also showing improvement when evaluated on other context lengths ($S = 128, S = 768$). Predictions (Figure 18 in Appendix) now attempted to follow the cyclical patterns, producing curved trajectories instead of flat lines, although deviations in phase/amplitude remained, especially over longer horizons. This demonstrated the potential of LoRA fine-tuning.

6.3 Hyper-parameter Search (Q3b)

To optimize performance, we conducted searches over key hyper-parameters, training each configuration for 500 steps.

Learning Rate and LoRA Rank A grid search explored learning rates $\eta \in \{10^{-5}, 5 \times 10^{-5}, 10^{-4}\}$ and LoRA ranks $r \in \{2, 4, 8\}$ at fixed $S = 512$. Loss curves generally showed decreasing and good learning dynamics (Figure 8). Comparing validation MAE (Figure 9), the best performance was achieved with the highest learning rate ($\eta = 10^{-4}$) and highest rank ($r = 8$) tested. This suggests that adapting from language to time-series, a significant change in domain, benefits from larger parameter updates (higher η) and increased adaptation capacity (higher r).

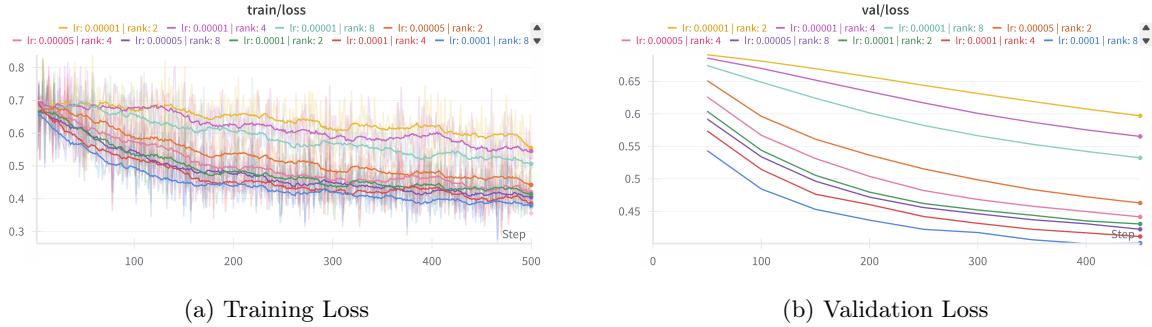


Figure 8: Loss curves from grid search over learning rate (η) and LoRA rank (r).

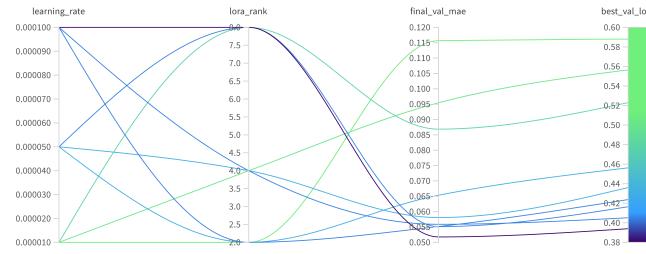


Figure 9: Validation MAE after 500 steps for varying LoRA rank (r) and learning rate (η) at $S = 512$. Lower is better. Best: $r = 8, \eta = 10^{-4}$.

Context Length Using the best $\eta = 10^{-4}$ and $r = 8$, we evaluated context lengths $S \in \{128, 512, 768\}$. Loss curves are shown in Figure 10. Comparing validation MAE (Figure 11), performance consistently

improved with longer context lengths. The lowest MAE was achieved with $S = 768$, confirming that providing more historical information aids prediction accuracy.

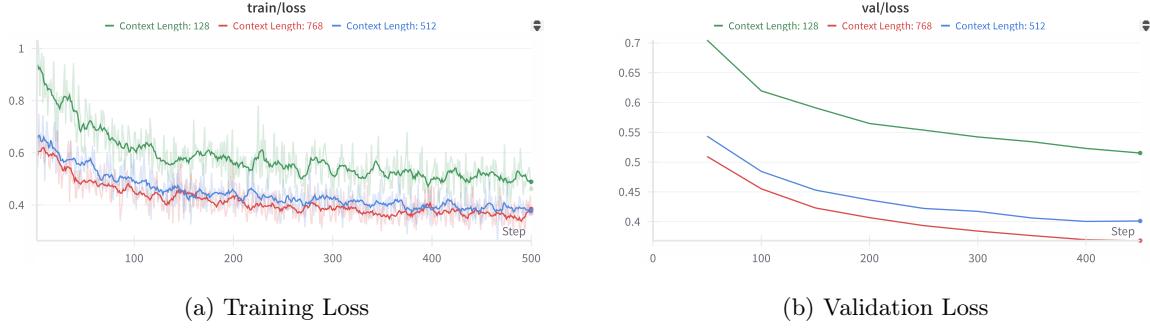


Figure 10: Loss curves for different context lengths (S) using best η, r .

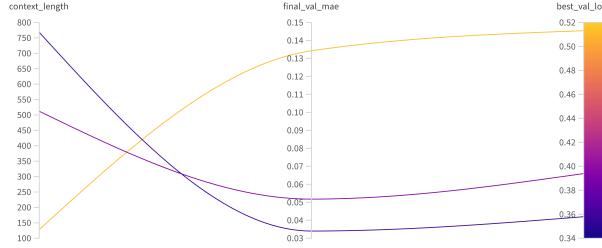


Figure 11: Validation MAE after 500 steps for different context lengths (S) using best η, r . Lower is better. Best: $S = 768$.

6.4 Final Training with Optimal Configuration (Q3c)

The final model was trained using the optimized hyper-parameters: $\eta = 10^{-4}$, $r = 8$, $S = 768$. Training utilized the remaining FLOPS budget 40%, running for approximately 3000 steps. The loss curves (Figure 12) show steady initial improvement, with the validation loss starting to plateau towards the end, suggesting convergence near the allocated budget limit.

Test set evaluation (Table 12, Final Training) reveals substantial improvement over the initial training phase. For the optimal $S = 768$, MSE improved by over an order of magnitude, reaching 0.0020, and MAE reached 0.0191. This final MAE is somewhat close to the estimated information loss from the LLMTTime preprocessing (Table 1), indicating the model learned the patterns nearly as well as possible given the data representation. Final predictions (Figure 19 in Appendix) are highly accurate for $S = 512$ and $S = 768$ over the short-term horizon and capture the dynamics reasonably well even when evaluated on the shorter $S = 128$ context. Although the predicted trajectory for $S = 128$ deviates from the ground truth, it still reasonably represents the oscillatory behaviour of both prey and predator populations.

6.5 FLOPS Usage Summary

A detailed breakdown of the FLOPS consumed by each experimental run (initial training, grid searches, final training) is provided in Appendix B.

Sweep Type	Context Length	Learning Rate	Rank	Validation MAE
Rank and LR Sweep	512	1e-5	2	0.11567
Rank and LR Sweep	512	1e-5	4	0.095438
Rank and LR Sweep	512	1e-5	8	0.086855
Rank and LR Sweep	512	5e-5	2	0.065344
Rank and LR Sweep	512	5e-5	4	0.058066
Rank and LR Sweep	512	5e-5	8	0.055118
Rank and LR Sweep	512	1e-4	2	0.055142
Rank and LR Sweep	512	1e-4	4	0.055869
Rank and LR Sweep	512	1e-4	8	0.051754
Context Length Sweep	128	1e-4	8	0.13428
Context Length Sweep	512	1e-4	8	0.051754
Context Length Sweep	768	1e-4	8	0.034004

Table 13: Hyperparameter Tuning Results: Final Validation MAE for Different LoRA Ranks, Learning Rates, and Context Lengths, Organized by Sweep Strategy.

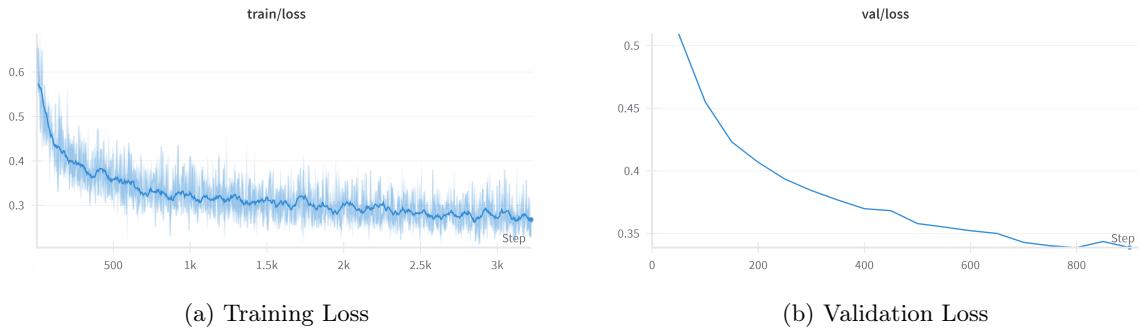


Figure 12: Loss curves during final training phase ($\eta = 10^{-4}$, $r = 8$, $S = 768$).

7 Discussion

The experiments demonstrated that **Qwen-2.5-0.5B-Instruct** can be adapted for time-series forecasting using LoRA with LLMTIME. Analysis of model components and limitations reveals further insights.

LM Head Bias Adaptation In addition to the LoRA matrices, the final LM Head layer’s bias parameters (`model.model.lm_head.bias`) were also trained. Analysing these adapted biases (Figure 13) reveals an interesting pattern related to the task structure. While the vast majority of the 151,936 bias values remained negative or zero (Median Bias: -0.0005), precisely seven developed a small positive bias. These correspond to the tokens essential for constructing the LLMTIME numerical format ($X.XX, Y.YY, \dots$; ..., Section 3): the structural punctuation ., , ;, and intriguingly, only the digits ‘0’, ‘2’, ‘7’, and ‘9’. Other digit tokens did not develop a positive bias.

This selective positive biasing of only four digits, which are somewhat evenly spread across the

0-9 range, is noteworthy as an efficient adaptation strategy. Instead of boosting all digits, the model focuses only on these four. One hypothesis for why this is sufficient involves two complementary aspects: Firstly, these boosted digits might serve as readily available ‘anchors’ or even direct proxies for nearby digits (e.g., approximating ‘6’ with the positively biased ‘7’ with minimal loss) when generating numerical output. Secondly, by ensuring these anchors are likely, the model may be able to generate other non-boosted digit tokens accurately when needed through learned context and internal state adjustments, without requiring a specific positive bias for every single digit.

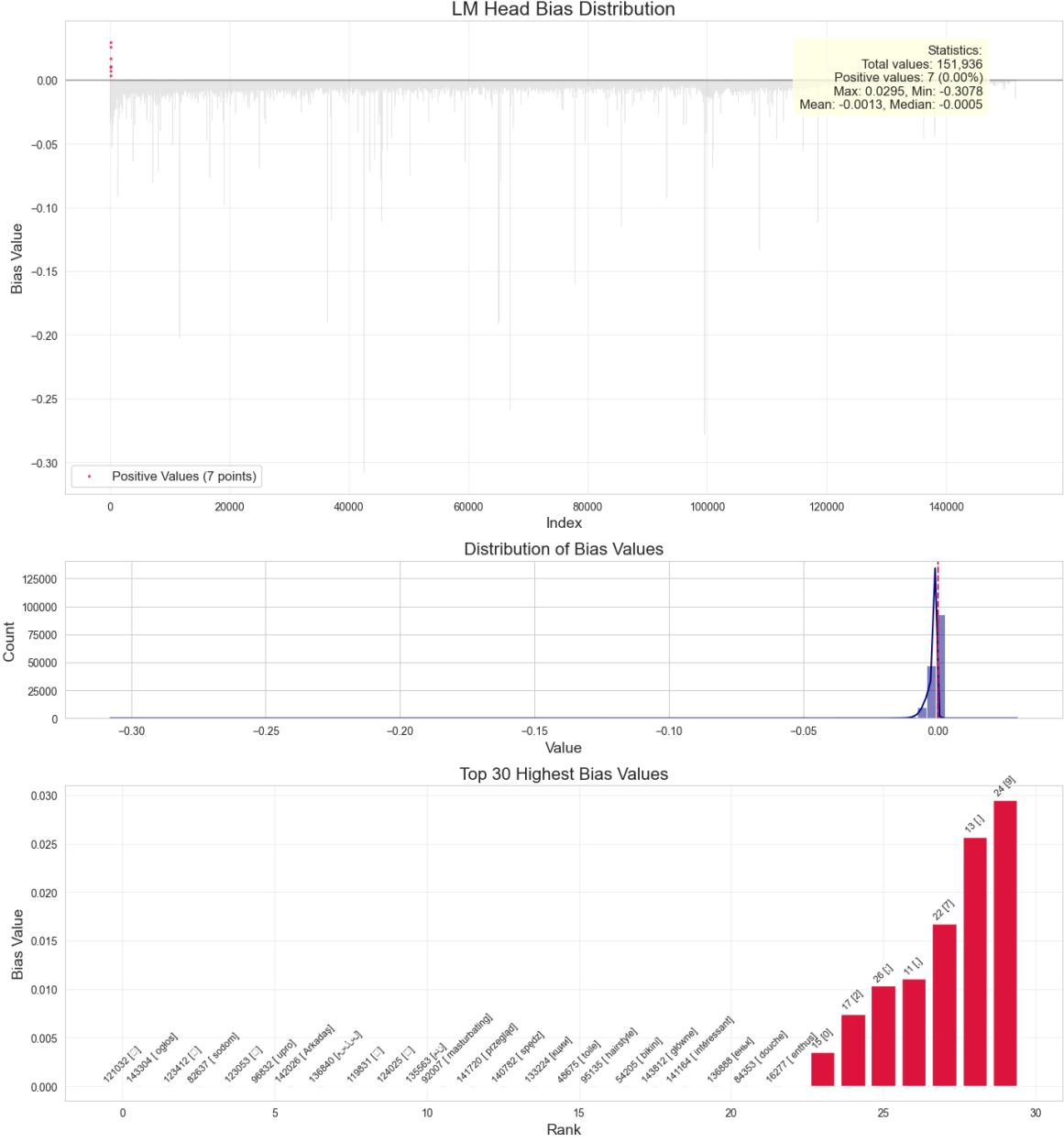


Figure 13: Analysis of the trainable LM Head bias weights after fine-tuning. Top: Bias values across vocabulary index, highlighting 7 positive values. Middle: Distribution histogram showing concentration below zero. Bottom: Top 30 highest bias values, including the positive ones.

Limitations Three main limitations impacted this study:

- **Scaling-Induced Prediction Ceiling:** The LLMTime preprocessing scaled data to $[0, 10]$ (Section 3). While crucial for token efficiency, this discouraged the model from predicting values exceeding 10, artificially capping or reversing trends during long-term forecasting where true dynamics might surpass this bound (As seen in the third plot of row 1 in Figure 14, and in detail in Figure 15). This is an artifact of the how we chose to represent the time-series, not necessarily model learning failure within the scaled range.
- **Data Exclusion from Padding Workaround:** The late discovery of Qwen’s left-padding requirement (Section 5) led us to adopt a workaround by discarding shorter sequence chunks that would have required padding. While this approach ensured correct model behaviour, it resulted in the exclusion of potentially valuable data from the ends of time series during training and evaluation. Fortunately, the lack of tuning for the padding token did not pose a significant drawback when inferring on a single time series.
- **Potential Catastrophic Forgetting:** Intensive fine-tuning for time-series forecasting may have impaired some general abilities. Qualitative tests (Figure 16) revealed that the final model misidentified numerical patterns—such as failing to recognize the Fibonacci sequence—and produced flawed explanations of decimal addition, tasks that earlier versions handled correctly. Although basic arithmetic and creative skills remained intact, this degradation suggests that specialization may have led to forgetting, warranting further investigation with larger benchmark datasets like GSM8K [1] and MATH [6].

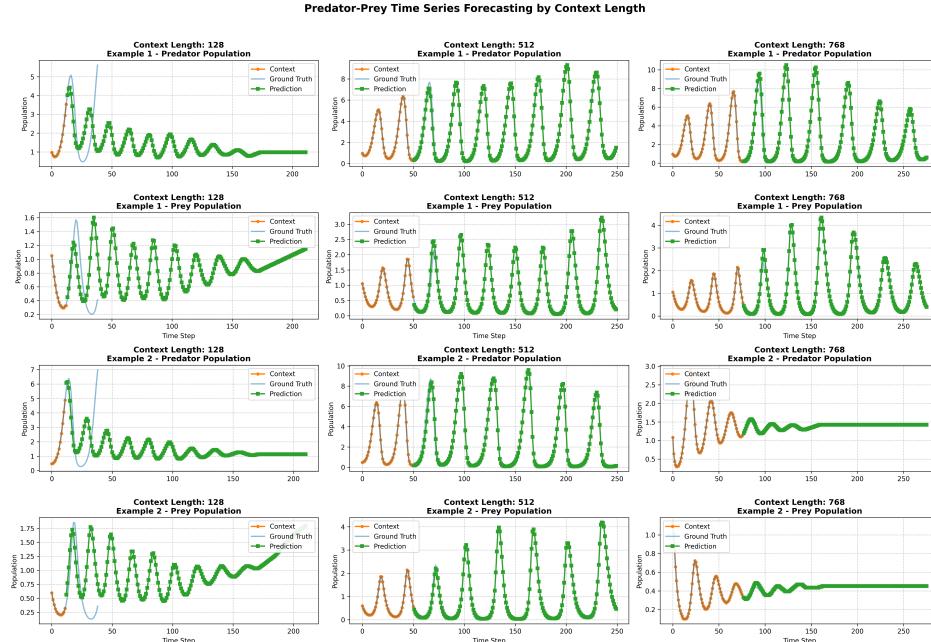


Figure 14: Example of prediction failure during long-term extrapolation (200 time steps). The model (green line) correctly captures the increasing trend initially (see third plot, first row. Figure 15) but unnaturally reverses direction as the predicted value approaches the normalization ceiling of 10, failing to match the ground truth (blue) which exceeds this bound.

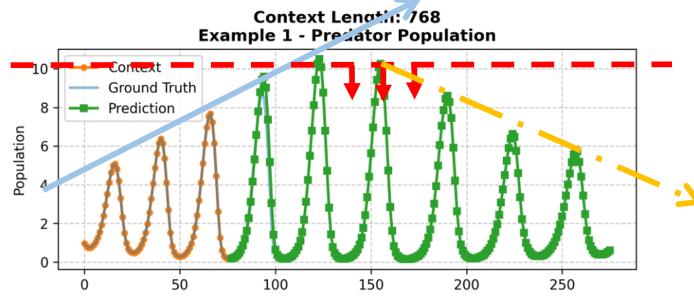


Figure 15: A zoomed-in view of Figure 14. The predicted trajectory (green) initially grows, as indicated by the blue arrow. However, once it reaches a population magnitude of 10 (red dashed line), it is suppressed and starts declining, as shown by the red arrows. This suppression leads to an overall downward trend in subsequent oscillations, marked by the yellow dashed arrow.

Analysis of Catastrophic Forgetting in Time-Series Data Fine-Tuning			
Question	Untrained Behavior	Initial Trained Behavior	Final Trained Behavior
How many r in strawberry?	The number "r" stands for "red" in the context of strawberries. So when you ask how many "r"s there are... <i>Incorrect: Provides a misleading explanation about "r" representing "red" rather than correctly counting the letters.</i>	In the Chinese language, "strawberry" is pronounced as "sǎo bāng", which means "red strawberry". The number... <i>Incorrect: Provides irrelevant information about Chinese pronunciation instead of counting letters.</i>	In the Chinese language, "strawberry" is written as "qiānér", which consists of two characters: "qiān" (straw)... <i>Incorrect: Again provides irrelevant information about Chinese rather than answering the simple counting question.</i>
what is the result of 1 + 1	The result of adding 1 and 1 is: 2 <i>Correct: Simple and accurate answer.</i>	The result of adding 1 and 1 is 2. <i>Correct: Concise and accurate.</i>	The result of 1 + 1 is 2. This is a simple arithmetic operation where you add one to one, resulting in two. <i>Correct: Provides the right answer with a brief explanation.</i>
what is the result of 0.12 + 3.24	The result of adding 0.12 and 3.24 is 3.36. <i>Correct: Concise and accurate.</i>	The result of $(0.12 + 3.24)$ is: (3.36) This calculation involves adding two numbers together... <i>Correct: Provides detailed step-by-step explanation with the right answer.</i>	The result of adding 0.12 and 3.24 is 3.36. This calculation involves basic addition of two numbers... <i>Mostly correct, but has some errors in the intermediate steps ($0.12 + 0.2 \neq 0.32$, etc.).</i>
0, 1, 1, 2, 3, 5, 8,	The sequence you've provided is known as the Fibonacci sequence. Here's a brief explanation of how it works... <i>Correct: Accurately identifies the Fibonacci sequence and explains the pattern.</i>	The sequence you've provided is the Fibonacci sequence, which starts with two one-digit numbers (0 and 1)... <i>Correct: Accurately identifies and explains the Fibonacci sequence.</i>	The sequence you've provided is an arithmetic sequence where each term increases by a constant difference... <i>Incorrect: Misidentifies the Fibonacci sequence as an arithmetic sequence and derives wrong formula.</i>
Write a poem that uses numbers as metaphors...	Verse 1: In the world of numbers, there's no room for error, A journey through the realms of numbers... <i>Partial: Creates a numerical poem but doesn't specifically use the requested numbers as metaphors.</i>	Verse 1: In the realm of numbers, where one is measured, One can see how they hold weight, and their worth... <i>Partial: References the specific numbers but doesn't fully explore them as metaphors.</i>	In the vast expanse of our existence, Numbers dance, like shadows on the page... <i>Partial: Creates a poetic response about numbers but doesn't specifically use the requested numbers.</i>

Figure 16: Qualitative analysis of model behavior across training stages on diverse prompts. The final model shows degraded performance on Fibonacci sequence recognition and errors in explaining decimal addition, indicating potential catastrophic forgetting due to specialization.

Recommendations for Budgeted Time-Series Fine-tuning (Q5) Based on our experience under the FLOPS constraint:

- **Prioritize Planning:** Accurate FLOPS calculation (Section 4) is essential for allocating budget across necessary experimental phases (exploration vs. final training).
- **Small-Scale Exploration:** Use short runs for hyper-parameter searches before committing significant budget.
- **Small-Scale Data Training:** When data is abundant, using only a fraction of it can be beneficial. However, for this dataset, I experimented with using only 1/10th of the data, but it led to over-fitting too quickly to yield meaningful insights. Therefore, I chose to use the entire dataset.
- **Expect Higher LR/Rank for Domain Shift:** Adapting from language domain to time-series domain might require higher learning rates and LoRA ranks (as seen in Section 6) than typical LLM fine-tuning to facilitate adaptation.
- **Maximize Context Length:** Longer context lengths consistently improved performance (Section 6);

Future Improvements (Q5) Potential next steps to enhance performance and efficiency include:

- **Improved Data Handling:** Implement correct left-padding to utilize all data chunks. Investigate alternative numerical representations less prone to clipping, such as xVal which allows continuous numerical tokenization [3].
- **Advanced Training Techniques:** use learning rate scheduling (e.g., warmup and decay) for potentially faster adaptation and finer convergence.
- **Efficient LoRA Variants:** Explore methods like dynamic LoRA ranks (e.g., ALoRA [8], SeLoRA [10]) which optimize rank allocation during training. These methods start with a small rank and learn to increase it strategically across different layers, enabling more efficient training, especially in early stages under resource constraints.
- **Quantization:** Apply techniques like QLoRA [2], Quantized LoRA, to reduce memory usage and potentially speed up training. Although this doesn't lower FLOPs, it enables training larger models within the same memory budget.

8 Conclusion

This work demonstrated the adaptation of the Qwen-2.5-0.5B-Instruct LLM for time-series forecasting using LoRA, achieving substantial performance gains within a strict 1×10^{17} FLOPS budget. Careful FLOPS planning enabled systematic hyper-parameter optimization, revealing that longer context lengths, higher LoRA ranks, and relatively high learning rates were beneficial for adapting the model from its original language domain. We identified the critical sensitivity of this model to input padding direction (requiring left-padding), a crucial factor for practical application. The optimized model achieved a final MAE of 0.0191 ($S = 768$), representing a sixfold improvement over the untrained model. Despite limitations related to data scaling and a padding workaround, this study confirms the feasibility and effectiveness of fine-tuning tiny LLMs for specialized quantitative tasks under given computational constraints.

References

- [1] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.
- [2] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.
- [3] Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, Bruno Régaldo-Saint Blancard, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. xval: A continuous numerical tokenization for scientific language models, 2024.
- [4] Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters, 2024.
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [6] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.
- [7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.
- [8] Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. Alora: Allocating low-rank adaptation for fine-tuning large language models, 2024.
- [9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
- [10] Yuchen Mao, Hongwei Li, Wei Pang, Giorgos Papanastasiou, Guang Yang, and Chengjia Wang. Selora: Self-expanding low-rank adaptation of latent diffusion model for medical image synthesis, 2024.
- [11] Noam Shazeer. Glu variants improve transformer, 2020.
- [12] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.
- [13] Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019.

A AI Usage

For this report, I declare that Gemini was used to improve grammar, while Claude assisted with coding, specifically in generating more visually appealing plots, partially generating the test suite and README, and providing auto-completion features.

Training Phase	Budget Usage (%)
Small-scale test (training pipeline validation)	0.623
Initial Training	8.000
Grid Search (LR, Rank)	36.000
Sweep over Context Length	11.158
Final Training	40.010
Total Used	95.791

Table 14: FLOPS Budget Usage Breakdown by Training Phase, as a percentage of the total 1×10^{17} FLOPS budget.

B FLOPS Breakdown

C Test Set Prediction Examples

This section provides visual examples of the model’s prediction capabilities on test set samples at different training stages.

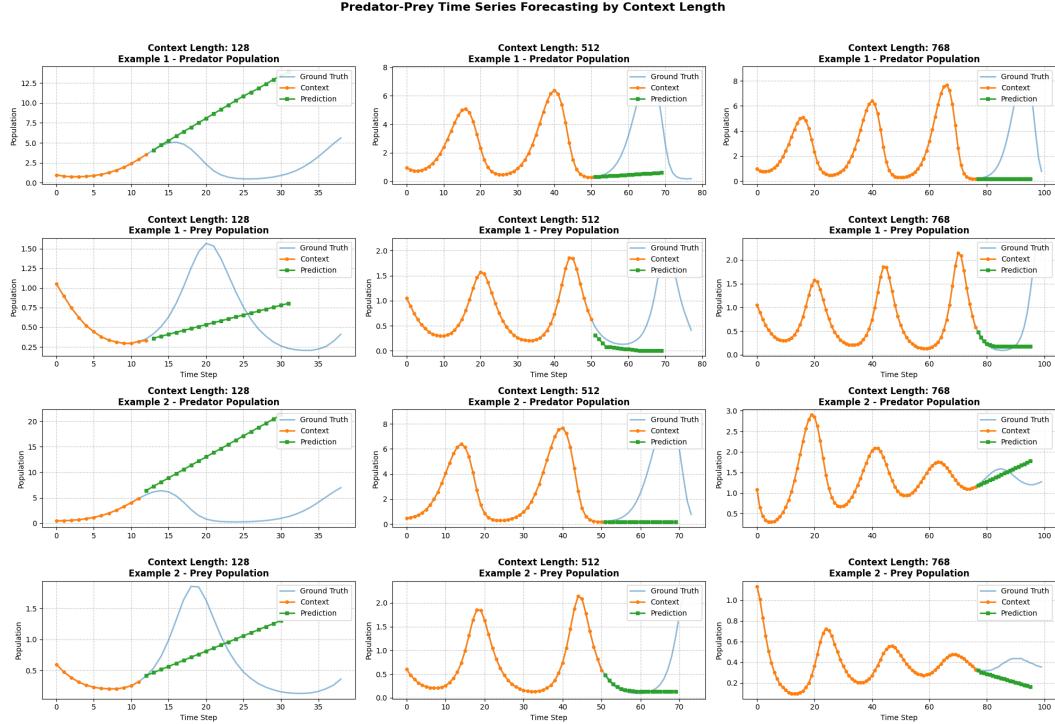


Figure 17: Untrained Qwen predictions (green) vs. ground truth (blue) for 20 steps. The model fails to capture dynamics, predicting near-constant values.

Predator-Prey Time Series Forecasting by Context Length

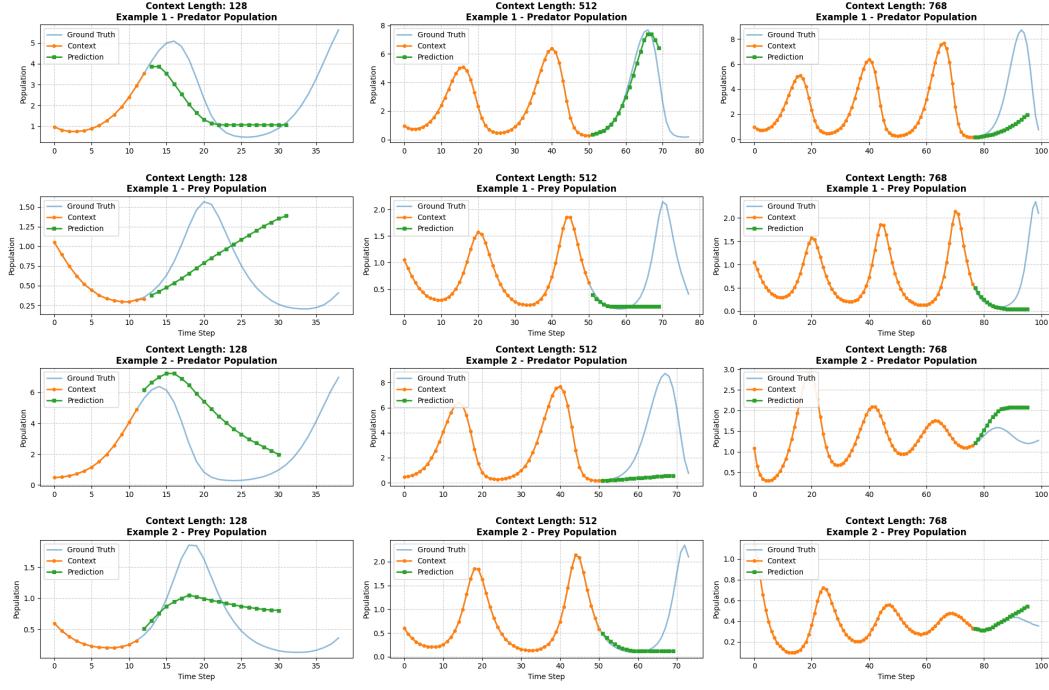


Figure 18: Predictions after initial 1000 training steps. The model now attempts to capture cyclical dynamics, a clear improvement over the untrained state.

Predator-Prey Time Series Forecasting by Context Length

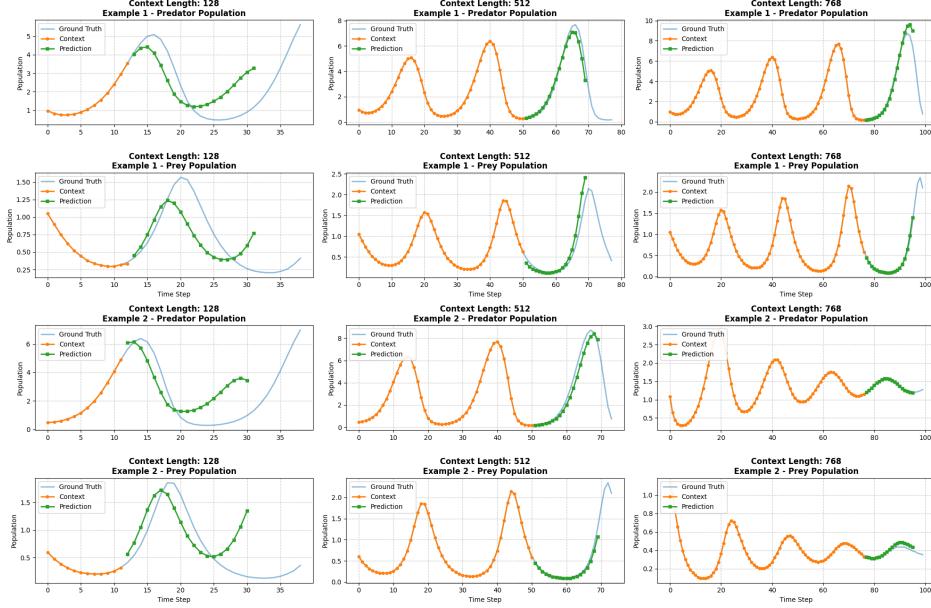


Figure 19: Final model predictions. Highly accurate short-term forecasts, especially with longer context ($S = 512, 768$).

D Additional LLMTIME Samples

Table 15: Sample 1 Preprocessing Results

Description	Data
Raw prey data	[0.9714744, 1.0787003, 1.260828, 1.5218158, 1.8605354]
Raw predator data	[1.0054137, 0.82180643, 0.6863802, 0.59312147, 0.53635156]
Tokens	[16, 13, 17, 18, 11, 16, 13, 17, 22, 26, 16, 13, 18, 21, 11, 16, 13, 15, 19, 26, 16, 13, 20, 24, 11, 15, 13, 23, 22, 26, 16, 13, 24, 17, 11, 15, 13, 22, 20, 26, 17, 13, 18, 20, 11, 15, 13, 21, 23, 26]

Table 16: Sample 2 Preprocessing Results

Description	Data
Raw prey data	[1.0732226, 0.8540631, 0.7681343, 0.7687197, 0.8349962]
Raw predator data	[1.112855, 0.9001321, 0.70823574, 0.55306584, 0.434924]
Tokens	[16, 13, 18, 20, 11, 16, 13, 19, 15, 26, 16, 13, 15, 23, 11, 16, 13, 16, 19, 26, 15, 13, 24, 22, 11, 15, 13, 23, 24, 26, 15, 13, 24, 22, 11, 15, 13, 22, 15, 26, 16, 13, 15, 20, 11, 15, 13, 20, 20, 26]