## Technical Report: Fine-Tuning a Large Language Model for Cantonese Instruction Following

**1. Introduction**

This report details the process of fine-tuning a pre-trained Large Language Model (LLM) to enhance its performance on Cantonese instruction-following tasks. The project aim to adapt a general-purpose LLM to a specialized domain (Cantonese language) using a specific dataset. The goal of fine-tuning in this case is to give the base model, Qwen 3b Instruct, the ability to process and generate in Cantonese language.

**2. Functional Requirements Implementation**

**2.1. Dataset Preparation**

* **Dataset Selection:** The project utilized the hon9kon9ize/yue-alpaca dataset available on Hugging Face. This dataset is specifically designed for Cantonese instruction-following tasks, making it appropriate for the project's goal.
* **Preprocessing and Formatting:** The data was formatted using the Alpaca instruction template, which structures the input with distinct sections for "Instruction," "Input," and "Response". A custom function formatting\_prompts\_func was implemented to apply this template to each example in the dataset, ensuring consistency. An End-of-Sequence (EOS) token was appended to each formatted text entry to signal the end of generation during training and inference.
* **Data Splitting:** The code loads the "train" split of the dataset. While explicit splitting into training, validation, and test sets within the notebook is not shown, the use of the SFTTrainer typically handles or assumes such splits if a separate validation set were provided. For evaluation, the last 10 samples from the training dataset were used in the Eval.ipynb notebook.

**2.2. Model Selection**

* **Pre-trained Model:** The Qwen/Qwen2.5-3B-Instruct model was chosen as the base LLM for fine-tuning. This model is part of the Qwen2.5 series, known for its strong multilingual capabilities and instruction-following performance, making it a suitable starting point for adapting to Cantonese. Its 3-billion parameter size offers a balance between performance and computational resource requirements.
* **Fine-Tuning Architecture:** Parameter-Efficient Fine-Tuning (PEFT) was employed using the LoRA (Low-Rank Adaptation) technique facilitated by the unsloth library. LoRA significantly reduces the number of trainable parameters by introducing low-rank matrices into specific layers, making fine-tuning more memory-efficient. The configuration involved:
  + Rank (r): 16
  + Target Modules: q\_proj, k\_proj, v\_proj, o\_proj, gate\_proj, up\_proj, down\_proj
  + LoRA Alpha: 16
  + Bias: "none"
  + Dropout: 0
* **Justification:** The selection of Qwen2.5-3B-Instruct is justified by its instruction-following capabilities and manageable size. Using LoRA via Unsloth allows for efficient fine-tuning on available hardware (like Google Colab GPUs) by training only a small fraction (approx. 1.00%) of the total parameters.

**2.3. Fine-Tuning Setup**

* **Environment:** The code utilizes the unsloth library, specifically designed to accelerate LLM fine-tuning and reduce memory usage. Installation commands suggest compatibility with Google Colab environments. Key optimizations include 4-bit quantization (load\_in\_4bit = True) and efficient checkpointing (use\_gradient\_checkpointing = "unsloth").
* **Training Loop:** The fine-tuning process is managed by the Hugging Face trl library's SFTTrainer. This trainer abstracts the training loop implementation.
* **Training Arguments:** The TrainingArguments class configured the training process with parameters including:
  + per\_device\_train\_batch\_size: 2
  + gradient\_accumulation\_steps: 4 (Effective batch size of 8)
  + warmup\_steps: 5
  + max\_steps: 300 (A full run would typically use num\_train\_epochs=1 and max\_steps=None)
  + learning\_rate: 2e-4
  + fp16 / bf16: Auto-detected based on hardware support
  + logging\_steps: 1
  + optim: "adamw\_8bit"
  + weight\_decay: 0.01
  + lr\_scheduler\_type: "linear"
  + seed: 3407
  + output\_dir: "outputs"
* **Logging & Checkpointing:** While SFTTrainer handles checkpointing to the output\_dir, logging was set to report\_to = "none" in this specific run, though integration with tools like WandB is possible. Training loss was logged every step.

**2.4. Hyperparameter Optimization**

* **Strategy:** The assignment required testing at least 3 different hyperparameter configurations. The provided notebook (Qwen2\_5\_(3B)\_Alpaca\_cantonese\_fine-tuned.ipynb) demonstrates a single fine-tuning run with a specific set of hyperparameters (learning rate = 2e-4, LoRA r = 16, etc.).
* **Tested Configurations:** Only one configuration is explicitly documented and run in the main fine-tuning notebook. To meet the assignment requirement fully, additional runs with varied parameters (e.g., different learning rates, LoRA ranks, batch sizes) would be necessary, followed by a comparison of their results (e.g., final training loss, evaluation metrics).
* **Documentation:** The parameters for the single run are documented within the TrainingArguments setup. A comparative analysis across different configurations is absent in the provided code.

**2.5. Model Evaluation**

* **Metrics:** The Eval.ipynb notebook implements a human evaluation strategy. It compares the responses of the original base model (Qwen/Qwen2.5-3B-Instruct) against the fine-tuned model (jackf857/canton\_fine-tuned\_qwen3b-instruct) for the last 10 samples of the training dataset. The primary metric is a user-assigned score from 1 to 10 based on the perceived quality of the fine-tuned model's response compared to the original.
* **Evaluation Set:** The last 10 samples from the training dataset were used as the evaluation set. Ideally, a separate, held-out test set should be used for a more robust evaluation.
* **Baseline Comparison:** The evaluation directly compares the fine-tuned model's output against the baseline (original pre-trained model) for the same prompts.
* **Results:** Based on the interactive scoring in Eval.ipynb, the average user score assigned across the 10 samples was approximately 6.56 out of 10. This suggests a perceived improvement over the baseline for these specific examples according to the evaluator, although the scoring shows variability.

**2.6. Error Analysis**

**2.6. Error Analysis (Requirement 6)**

* **Initial Observation:** During the initial fine-tuning attempts, the model struggled to demonstrate proficiency on the hon9kon9ize/yue-alpaca dataset with (Qwen/Qwen2.5-7B) base model, even after the planned training steps. The generated Cantonese responses were often inaccurate, lacked fluency, or failed to follow instructions correctly.
* **Hypothesis and First Attempt:** This poor performance suggested potential underfitting, where the model hadn't learned the dataset's patterns sufficiently. The first corrective action attempted was to significantly increase the number of training steps (max\_steps) to allow the model more exposure to the data. However, this approach led to computational constraints, specifically exceeding the available VRAM with batch size for manageable training time. Causing the training process to fail.
* **Second Attempt and Refinement:** Recognizing the resource limitations with the initial, larger base model and extended training, a second approach was adopted. This involved selecting a smaller, yet still capable, instruction-tuned base model (Qwen/Qwen2.5-3B-Instruct). The number of training steps was still increased compared to the initial baseline but kept within feasible limits (300 steps were used in the documented run) to avoid VRAM overload while allowing for sufficient learning. This revised strategy, combining a more appropriately sized base model with a moderate increase in training duration, proved successful outcome that perform better than the outcome with larger base model, leading to the fine-tuned model evaluated in the Eval.ipynb notebook.
* **Further Analysis (Potential):** While the second attempt yielded a functional model, a deeper analysis could still be performed on the final model's outputs (as demonstrated in Eval.ipynb). Examining specific instances where the model scores lower or makes mistakes would help identify residual weaknesses (e.g., handling specific types of complex instructions, nuances in Cantonese phrasing, or factual inaccuracies) and guide future iterations, such as targeted data augmentation or further hyperparameter adjustments.

**2.7. Inference Pipeline**

* **Interface:** The notebooks demonstrate a simple inference pipeline. The FastLanguageModel.for\_inference(model) function optimizes the model for faster generation. Input prompts are formatted using the defined alpaca\_prompt template.
* **Processing:** The tokenizer prepares the input tensors, and the model.generate function produces the output tokens. The TextStreamer class is used to display the output token by token, providing a continuous inference experience rather than waiting for the full response. This setup ensures reasonably efficient processing for single-turn inference.

**2.8. Documentation & Reproducibility**

* **Environment Setup:** The notebooks include pip install commands for necessary libraries, including unsloth, transformers, datasets, trl, and dependencies, aiding reproducibility. A formal requirements.txt file is not included but could be easily generated from these commands.
* **Instructions:** Markdown cells within the notebooks provide explanations for code sections, setup, training, inference, and saving. The code demonstrates saving the fine-tuned LoRA adapters locally (save\_pretrained) and pushing them to the Hugging Face Hub (push\_to\_hub), which is crucial for reproducibility and sharing. Loading the saved adapters is also shown.
* **Code Comments:** The Python code includes comments explaining specific parameters and steps, enhancing readability.

**3. Quality/Portfolio Score Considerations**

* **Methodology & Approach:** The approach of using a capable base model (Qwen2.5-3B-Instruct) and fine-tuning it on a domain-specific dataset (yue-alpaca) with an efficient technique (LoRA via Unsloth) is well-reasoned for the task. The dataset choice is appropriate.
* **Technical Implementation:** The code leverages established libraries (transformers, datasets, trl) and incorporates best practices like 4-bit quantization and gradient checkpointing for efficiency. The use of unsloth demonstrates awareness of tools for optimizing the fine-tuning process. Code is organized into logical cells within Jupyter notebooks.
* **Analysis & Interpretation:** The Eval.ipynb provides a basic framework for quantitative comparison via human scoring. The reported average score gives an initial insight into performance improvement. However, deeper qualitative error analysis and interpretation are missing.
* **Documentation & Presentation:** The notebooks contain good markdown explanations and code comments. Saving the model to the Hub enhances reproducibility. Visualizations (e.g., training loss curves, comparison graphs) are not explicitly generated in the provided code but would strengthen the presentation.
* **Overall "Polish":** The project successfully fine-tunes a model for Cantonese. Edge cases and limitations (like the evaluation set size and lack of hyperparameter search) are partially apparent but not explicitly discussed in depth within the code's documentation. Ethical considerations regarding potential biases in the yue-alpaca dataset are not addressed.

**4. Limitations and Future Improvements**

* **Hyperparameter Tuning:** Only one hyperparameter set was tested in the main notebook, falling short of the requirement to test at least three. A more systematic search (e.g., grid search, random search, Bayesian optimization) could yield better results.
* **Evaluation:** Evaluation relied on the last 10 samples of the training set and subjective human scoring. Using a dedicated, unseen test set and incorporating automated metrics (like BLEU, ROUGE, or model-based evaluations) alongside human judgment would provide a more robust assessment.
* **Error Analysis:** A detailed qualitative error analysis is needed to understand the model's weaknesses and guide further improvements.
* **Dataset:** While yue-alpaca is relevant, its size and diversity might be limitations. Augmenting it or using additional Cantonese datasets could improve generalization. Filtering for potential biases should also be considered.
* **Reproducibility:** Providing a requirements.txt file and potentially containerizing the environment (e.g., Docker) would further enhance reproducibility.

**5. Conclusion**

This project successfully demonstrated the process of fine-tuning the Qwen/Qwen2.5-3B-Instruct model for Cantonese instruction following using the hon9kon9ize/yue-alpaca dataset and the unsloth library for efficient LoRA tuning. The functional requirements were partially met, with clear implementation of dataset preparation, model selection, fine-tuning setup, inference pipeline, and documentation aspects. The evaluation showed perceived improvement based on limited human scoring. Key areas for improvement include conducting the required hyperparameter optimization, performing a thorough error analysis, and using a dedicated test set with more comprehensive evaluation metrics. Overall, the project establishes a solid foundation for a Cantonese-specific instruction-following model.