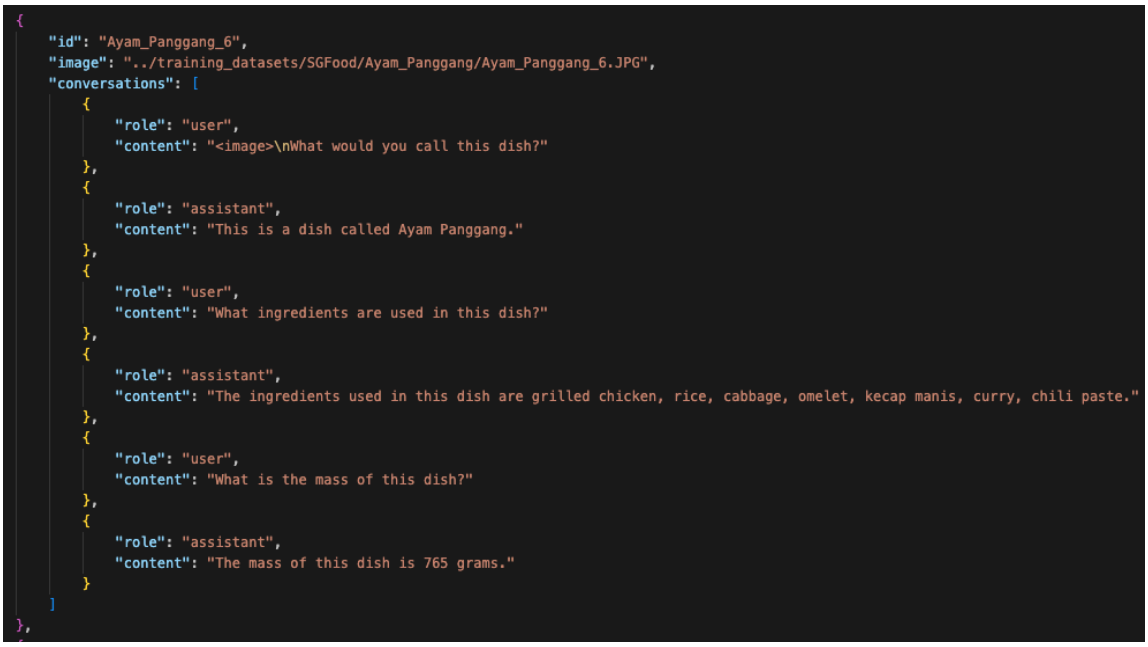
Steps for Fine-Tuning MiniCPM-V2 for Local Food Classification

1. Prepare the Dataset

- **Create JSON File:**
  - Structure the dataset in a JSON format where each entry contains the image path, questions, and answers. Example format:



```
{
    "id": "Ayam_Panggang_6",
    "image": "../training_datasets/SGFood/Ayam_Panggang/Ayam_Panggang_6.JPG",
    "conversations": [
        {
            "role": "user",
            "content": "<image>\nWhat would you call this dish?"
        },
        {
            "role": "assistant",
            "content": "This is a dish called Ayam Panggang."
        },
        {
            "role": "user",
            "content": "What ingredients are used in this dish?"
        },
        {
            "role": "assistant",
            "content": "The ingredients used in this dish are grilled chicken, rice, cabbage, omelet, kecap manis, curry, chili paste."
        },
        {
            "role": "user",
            "content": "What is the mass of this dish?"
        },
        {
            "role": "assistant",
            "content": "The mass of this dish is 765 grams."
        }
    ]
},
```

- **Collate Data:**
  - Gather and organize all food images and their respective metadata into this format.

2. Split Dataset into Train and Test Sets

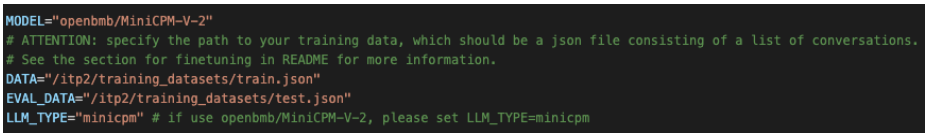
- **Run split.py (/minicpmv2/split.py):**
  - **Use the provided split.py script to divide the dataset into 80% for training and 20% for testing, ensuring each category is represented proportionally in both sets.**

3. Set Up Fine-Tuning

- **Ensure the following files (/minicpmv2/finetuning_scripts) are prepared and in place:**

1. **dataset.py: Handles dataset loading and preprocessing to convert the provided JSON data into a format compatible with training.**
2. **ds_config_zero2.json: Configuration file for DeepSpeed, optimized for memory-efficient training with stage 2 zero optimization.**
3. **ds_config_zero3.json: Advanced DeepSpeed configuration enabling stage 3 zero optimization for improved scalability.**
4. **finetune_ds.sh: Main script for distributed fine-tuning using full model parameter updates. Configures learning rate, batch sizes, and training strategy.**
5. **finetune_lora.sh: Optional script for LoRA (Low-Rank Adaptation) fine-tuning, focusing on efficient attention layer updates.**
6. **finetune.py: Core fine-tuning script orchestrating model training, evaluation, and logging.**
7. **trainer.py: Handles the training and evaluation logic, integrating the model, optimizer, and training data.**

- **Update File Paths and Training Parameters**
    - **Before editing hyperparameters, update the paths in the finetune_ds.sh file:**
        1. **Set Paths for Training and Evaluation Data:**



- **Edit Hyperparameters:**
    - **Adjust the learning rate and number of epochs directly in the torchrun command inside finetune_ds.sh. Example:**

```
torchrun $DISTRIBUTED_ARGS finetune.py  \
    --model_name_or_path $MODEL \
    --llm_type $LLM_TYPE \
    --data_path $DATA \
    --eval_data_path $EVAL_DATA \
    --remove_unused_columns false \
    --label_names "labels" \
    --prediction_loss_only false \
    --bf16 false \
    --bf16_full_eval false \
    --fp16 true \
    --fp16_full_eval true \
    --do_train \
    --do_eval \
    --tune_vision true \
    --tune_llm true \
    --model_max_length 2048 \
    --max_slice_nums 9 \
    --num_train_epochs 3 \
    --eval_steps 1000 \
    --output_dir output/output_minicpmv2 \
    --logging_dir output/output_minicpmv2 \
    --logging_strategy "steps" \
    --per_device_train_batch_size 1 \
    --per_device_eval_batch_size 1 \
    --gradient_accumulation_steps 1 \
    --evaluation_strategy "steps" \
    --save_strategy "steps" \
    --save_steps 1000 \
    --save_total_limit 10 \
    --learning_rate 1e-6 \
    --weight_decay 0.1 \
    --adam_beta2 0.95 \
    --warmup_ratio 0.01 \
    --lr_scheduler_type "cosine" \
    --logging_steps 1 \
    --gradient_checkpointing true \
    --deepspeed ds_config_zero2.json \
    --report_to "tensorboard"
```

4. Run Fine-Tuning

- **Execute the fine-tuning script by running:**
  sh finetune_ds.sh

- **The script will:**
  - **Load the base MiniCPM-V2 model.**
  - **Fine-tune it using the provided training data.**

- **Evaluate it periodically using the test data to monitor progress.**

5. Evaluate the Fine-Tuned Model

- **Use models_evaluation.py (/minicpmv2/models_evaluation.py) to compare the base MiniCPM-V2 model with the fine-tuned model:**
  - **This will provide metrics like accuracy by each food category and overall accuracy, allowing you to assess the improvements achieved through fine-tuning.**

*** For more details on fine-tuning MiniCPM, you can visit the official MiniCPM-V Fine-Tuning Documentation.**