Introduction:
The overall approach involves training a model using a vision-based architecture, such as Vision Transformer, ResNet, Inception, or through Multimodal LLM fine-tuning. This typically leverages pre-trained weights from models trained on the ImageNet dataset.

I explored two main approaches: Multimodal LLM fine-tuning and training vision-based zero-shot image classification models.

Approach 1: Zero-Shot Image Classification Models
The model's inputs consist of product categories and their corresponding images linked to specific IDs. The outputs are the attributes, which can be encoded using a label encoder.

For vision-based image classification models, such as Open-CLIP's Vision Transformer (Base or Large), Google's SigLip, Microsoft's Swin Transformer, or Facebook's DEIT, the process is as follows:

1. Pass the image through the model's processor to convert it into tensors.
2. Use label-encoded attributes as labels and the category as a feature.
3. Extract pixel values from the image input to compute the image embeddings.
4. Concatenate the image embeddings with the category.
5. Feed the combined inputs to each attribute output, as this is a multi-label, multi-output problem.

To handle the outputs effectively, it is recommended to use 10 fully connected output layers, one for each attribute.

Approach 2: Fine-Tuning Multimodal LLMs

For fine-tuning a multimodal LLM, I used Qwen2-VL with 2 billion parameters. The steps are as follows:

1. Create the dataset in formats like Alpaca or ShareGPT.
2. Prepare the dataset and other required arguments for the **SFTTrainer**.
3. Used the Llama Factory model to fine-tune the multimodal LLM with the prepared dataset.

This approach leverages the pre-trained capabilities of Qwen2-VL to adapt the model to specific tasks effectively, ensuring high-quality outputs tailored to the dataset.

Data Preprocessing:

The first step in data preprocessing is to handle null values by replacing them with a dummy value, 'no'. This is done based on the specified length (len). For example:

- If len = 5 and there are fewer than five missing values, the initial missing values are filled with the most common value for the attribute within the given id.
- Once the required number of values (len) is filled, any remaining missing values are replaced with 'no'.

For products in the **Category = Saree**, I observed a unique pattern:

- Unlike other categories where *Attribute 1* represents the color, for Sarees, *Attribute 4* denotes the color.
- To address this, I replaced the values of *Attribute 1* with those of *Attribute 4*.

For Vision-Based Models (e.g., Open-CLIP ViT-Base):
Data augmentation techniques were applied to make the model more robust and generalized. This helps improve its performance on unseen data.

For Fine-Tuning with Llama-Factory:
I used the original images provided in the dataset without applying any data augmentation.

Modeling Approach:

For Approach 1:
I utilized various zero-shot image classification models from Hugging Face. The models used are as follows:

1. **OpenCLIP/ViT-Base-Patch32**
2. **OpenCLIP/ViT-Base-Patch14**
3. **Google/Siglip-Large**
4. **Google/Siglip-Base**
5. **Microsoft/Swin Transformer**

Reason for Choice:
The primary reason for selecting these zero-shot image classification models is their exceptional performance on the ImageNet dataset, which consists of over 1,000 classes. The problem of distinguishing products and accurately predicting their attributes shares similarities with the challenges addressed by these models.

Vision Transformers, in particular, have demonstrated outstanding performance on large datasets like LAION-2B. Given their robustness and ability to generalize across diverse categories, they were my first choice for this task.

Architecture:

The architecture of these Vision Transformer (ViT)-based models incorporates a **Transformer architecture for image encoding** and a **masked self-attention Transformer for text encoding**. The key components are as follows:

1. **Image Encoding with Vision Transformers (ViT):**
    - The input image is divided into fixed patches, with the size of each patch determined by the ViT variant. For instance, in ViT-Base-Patch32, each patch is of size 32×32 pixels.
    - Positional encoding is applied, which incorporates information about the position of each patch.
    - The patches are passed through a series of Transformer blocks. Each block consists of:
        - Multi-Head Self-Attention layers.
        - Feed-Forward layers.
        - Residual connections to facilitate skip-layer connections.
        - Layer normalization to stabilize training.
2. **Text Encoding:**
    - Text inputs (e.g., product categories) are encoded using a masked self-attention Transformer.
    - The encoded text features are concatenated with the image features.
3. **Feature Fusion:**
    - The concatenated features (image embeddings and category) are passed to the output layers.
    - For CLIP models, the concatenated features are first passed through a fully connected layer of size 512, followed by another fully connected layer of size (512, 256).
    - The number of layers can be adjusted; in my experiments, these dimensions provided the best results.
4. **Generalization to Other Models:**

- A similar architecture is applied to other models, such as Google's Siglip. The primary difference lies in the dimensions of the output layer and specific configurations of the underlying Transformer blocks.

This architecture ensures that the model learns the relationships between the product images and their corresponding attributes effectively.

Final Public Score for the approach 1 : 0.59

For Approach 2:
Using the LLM fine-tuning approach with **Qwen2-VL-2B-Instruct**, the process involved:

- Preparing the dataset in JSON format, specifically using the **ShareGPT** format, which is well-suited for this task.
- Fine-tuning the model with **LlamaFactory**, a platform chosen for its computational efficiency and cost-effectiveness.
- Employing **LoRA adapters** with the **qwen2_vl** template for fine-tuning.

LoRA (Low-Rank Adaptation):
LoRA fine-tunes specific layers of the model by adding low-rank matrix adaptations to the pre-trained weights. This method is both cost-effective and capable of maintaining high-quality performance, making it an ideal solution for large models like Qwen2-VL.

Model Configuration and Training Details:

Arguments Used for the Model:

- **stage**: 'sft' - Indicates the use of Supervised Fine-Tuning (SFT) Trainer.
- **template**: 'qwen2_vl' - Specifies the fine-tuning template.
- **finetuning_type**: 'lora' - Low-Rank Adaptation (LoRA) is used for efficient and cost-effective fine-tuning.
- **lora_target**: 'all' - Applies LoRA to all target layers.
- **per_device_train_batch_size**: 2 - Sets the batch size per device.
- **gradient_accumulation_steps**: 4 - Accumulates gradients over 4 steps for larger effective batch sizes.
- **lr_scheduler_type**: 'cosine' - A cosine learning rate scheduler for smooth decay.
- **warmup_ratio**: 0.1 - Warm-up ratio for the learning rate.
- **max_grad_norm**: 1.0 - Clips gradients to avoid exploding gradients.

- **loraplus_lr_ratio**: 16 - Learning rate ratio for LoRA layers.
- **fp16**: True - Enables mixed-precision training with FP16 to speed up training.
- **use_liger_kernel**: True - Utilizes Liger kernel for memory efficiency during training.

The parameters are saved in JSON format (train_qwen2vl.json) and passed to the **LlamaFactory CLI** for training.

Model Architecture:

- **Model Type**: Qwen2VLForConditionalGeneration
- **Hidden Size**: 1536
- **Intermediate Size**: 8960
- **Number of Attention Heads**: 12
- **Number of Hidden Layers**: 28
- **Max Position Embeddings**: 32768
- **Max Window Layers**: 28
- **Torch Dtype**: bfloat16 (optimized for performance and memory efficiency).

Vision Configuration:

- **Spatial Patch Size**: 14
- **Input Channels**: 3 (RGB).

Processor Details:

- **Processor Class**: Qwen2VLProcessor
- Includes image normalization, resizing, and rescaling.
- **Image Mean**: [0.48145466, 0.4578275, 0.40821073]
- **Image Std**: [0.26862954, 0.26130258, 0.27577711]
- **Patch Size**: 14

Training and Results:
After completing training, the **LoRA adapters** are saved and merged with the full model for optimal and efficient results.

Final Public Score:
The model achieved a public score of **0.61**, the highest in my experiments, making this approach my final submission.

Novelty and Innovation:

1. **Dynamic Attribute Filling:**
   - Dummy values were filled based on the **most common values** of a particular attribute until the total attributes matched the specified len value. After reaching this threshold, dummy values like 'no' were used for the remaining entries.
   - This approach significantly improved the public score, raising it from **0.51** to **0.59** and finally to **0.61**.
2. **Category-Specific Adjustments:**
   - For the category **'Saree'**, the first attribute (attr_1) was swapped with the fourth attribute (attr_4) since, unlike other categories where attr_1 corresponds to color, for Sarees, color information is found in attr_4.
   - This adjustment contributed further to score improvement.

These two strategies were applied consistently in both approaches, showcasing a thoughtful approach to category-specific nuances and data preprocessing.

Evaluation Metrics:

For both approaches, the following metrics were chosen to evaluate the model performance:

1. **Metrics Chosen:**
   - **F1-Score (Macro and Micro):** These metrics were chosen to evaluate the balance between precision and recall for each class and overall.
   - **Accuracy:** Used to measure the overall accuracy of the model on both training and validation sets.
   - **Cross-Entropy Loss:** Since this is a multi-class, multi-output problem, cross-entropy loss was used to compute the error for each attribute,

providing a better understanding of performance across various attributes.

Results:

The F1-scores at the **Category Level** for both approaches are presented in the following table format:

For Approach 1 (OpenCLIP/ViT-Large-14) as it is best

Category	Micro F1-Score	Macro F1 Score	Harmonic Mean of Macro and Micro
Men T-Shirts	0.89	0.91	0.897
Saree	0.75	0.68	0.72
Kurtis	0.87	0.89	0.86
Women Tops & Tunics	0.88	0.75	0.81
Women T-Shirts	0.74	0.71	0.71

For Approach 2 which is Finetuning of Qwen2VL -2B The only metric is Loss which is after 3 Epochs of training

Category	Loss
Men-Tshirts	0.031
Saree	0.04
Kurtis	0.02
Women Tops & Tunics	0.017
Women T-Shirts	0.027

The public score I achieved for Approach 2 is better, so I consider it my final attempt for this problem.

Error Analysis: The model fails to produce accurate results when using loss functions other than Cross Entropy Loss, such as BCELoss or others, because this is a multi-class, multi-output problem. Therefore, it's better to use different final layers for each attribute. Initially, I didn't follow this approach, which caused the loss to increase rather than decrease. In some cases, the loss even became negative and continued decreasing in the negative direction. It's crucial to pay attention to this aspect.

Additionally, it's important to ensure that the output dimensions of the model match the fully connected layers when passing them to the outputs.

Conclusion:

Summary of Results: The model's performance significantly improved from 0.47, my first submission, to 0.61, my final submission, thanks to changes in methodology and the adoption of new, unique processing techniques. I had hoped to increase the score even further, but due to limited resources for fine-tuning larger models, my final score reached a plateau at 0.61. However, I believe this is a decent result for a multi-class, multi-output challenge.

I'm confident that if I fine-tune the Qwen2VL-7B-Instruct model (with 7 billion parameters), I could achieve a score of 0.8 or even 0.9. I strongly suspect that the top 10 participants are also using the Qwen2VL-7B-Instruct model, likely following a similar approach to mine, especially using Llama Factory for fine-tuning with the same parameters.

Limitations and Future Improvements: The main limitation has been the lack of resources. Since I am using a MacBook M1 Air, only a few models are supported by MLX, and there is currently no multi-modal support available on MLX. This means I have to run everything on either Kaggle or Colab, with Kaggle being my platform of choice. However, even with 30GB of memory on Kaggle, it still feels a bit restrictive.

In the future, I hope to access more powerful resources to fine-tune larger models more effectively.