Note: We have not received any script from the company

Identified Problems

1. **NaN loss after a few epochs**:
    - This issue typically arises when:
        - The learning rate is too high, causing large weight updates that result in unstable gradients.
        - There may be numerical instability in the model, especially if the activation function outputs become too large or small.
        - Incorrect initialization or improper data preprocessing.
2. **GPU usage is below 30%**:
    - Low GPU utilization often occurs if the model or data pipeline is inefficient:
        - **Batch size** could be too small, not fully utilizing the GPU.
        - The model might be bottlenecked by the CPU, particularly in data loading or preprocessing.
        - The model architecture or computation might not be parallelized properly across GPU resources.
3. **Training is slower than expected**:
    - Possible causes include:
        - **Batch size** might be too small.
        - **Suboptimal data pipeline**, such as inefficient data augmentation or loading, may be slowing things down.
        - **Inefficient model design** or using too many layers unnecessarily.

Suggested Fixes

- **NaN loss issue**:
    - **Reduce the learning rate**: This can stabilize the training process.
    - **Gradient clipping**: Limit the gradient values to prevent NaN errors due to exploding gradients.
    - **Check data preprocessing**: Ensure the data is normalized or scaled properly.
    - **Add regularization**: Introduce dropout or L2 regularization to prevent overfitting and help with the NaN issue.
- **GPU under-utilization**:
    - **Increase batch size**: Larger batches will utilize more GPU memory and speed up training.
    - **Use a more efficient data pipeline**: Leverage tf.data for better parallelism and optimized input pipeline.
    - **Enable mixed precision**: This will use lower-precision arithmetic (float16) where possible, helping to speed up training and reduce GPU memory usage.
- **Slower training**:
    - **Model optimization**: Simplify the model architecture or use pre-trained models.
    - **Efficient data loading**: Ensure data is preloaded and augmented efficiently.