Technical Assessment - Junior ML Engineer @ Biz-Tech Analytics

Q1: Choosing the Right Approach

I would start with **detection**, since the task is to identify whether the product has a label or not. Detection makes sense because we're looking for the presence (or absence) of a specific object on the product. If detection struggles due to very subtle visual differences, I'd move towards **segmentation**, which provides pixel-level detail and could help distinguish the label area more precisely. As a fallback, a simpler **binary classification** could be tried by cropping the region where the label should be and just classifying "label vs no-label." This way, I'd move from the broadest to the most fine-grained approach depending on results.

Q2: Debugging a Poorly Performing Model

The first thing I'd do is check for **data mismatch** — the 1000 training images may not match the conditions of the factory (lighting, angle, noise). I'd visualize random training vs test images to spot these differences. Next, I'd see if the labels are correct and consistent, since annotation errors are common. I'd also check for **overfitting** by comparing training vs validation accuracy. If overfitting is happening, I'd try data augmentation, more diverse samples, or fine-tuning a pre-trained model. Honestly, I may not be the absolute best right now, but I am dedicated towards learning and iterating step by step to debug effectively.

Q3: Accuracy vs Real Risk

Accuracy alone isn't the right metric here because it hides the fact that defective products are being missed. In this case, the real-world risk is that even a single defective product reaching the customer is costly. I'd focus on **recall (or sensitivity)** for defective products, since we care about catching *all* of them. Precision also matters, but it's less risky to have false alarms than to miss actual defects. So I'd monitor precision–recall trade-offs or use **F1 score** instead of plain accuracy.

Q4: Annotation Edge Cases

I'd keep blurry or partially visible objects in the dataset, but I'd mark them clearly as such during annotation. The reason is that in real-world factory settings, not every image will be perfect — the model should be somewhat robust to imperfect data. However, including too many low-quality images could also confuse the model and reduce overall accuracy. The trade-off is between **robustness vs clarity**: keeping them teaches the model to handle real-world noise, but too many could lower precision. Ideally, I'd keep them in smaller proportions or use them as a test set for robustness checks.