Prompted Segmentation for Drywall QA

The dataset provided consists of object detection annotations with bounding boxes. However, segmentation models require pixel-level masks as ground truth for training or fine-tuning. This presents a fundamental challenge: generating label masks (pseudo-masks) from the bounding box annotations to enable model fine-tuning for segmentation tasks.

Phase 1 - Analyzing the dataset and generating masks from it.

After analyzing the data, I discovered that the dataset has a significant class imbalance:

- **Cracks Dataset:**
  - **Name:** `cracks`
  - **Total Images:** 5,369
- **Drywall-Join-Detect Dataset:**
  - **Name:** `Drywall-Join-Detect`
  - **Total Images:** 1,022

Based on this insight, I decided to generate masks first, before balancing the dataset. My reasoning was that this approach would prevent the model from developing a bias toward the overrepresented class. I knew that performing augmentation later on high-quality masks would yield significantly better results.

Always create your highest-quality ground truth first, and then use that clean data as the foundation for augmentation.

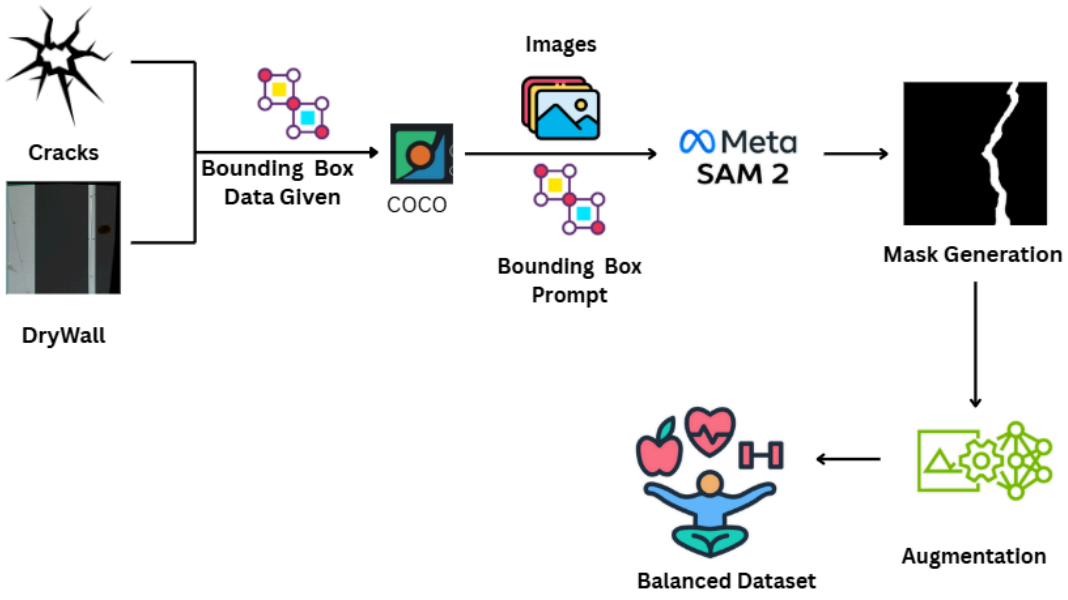
The Problem with Augmenting Bounding Boxes First

When you apply geometric augmentation (like rotation, shear, or perspective shift) to an image, the bounding box must also be transformed. This transformation is inherently an approximation.

1. **Loss of Precision:** An axis-aligned bounding box that perfectly encloses a rotated object will inevitably be larger and contain more background noise than the original tight-fitting box.
2. **Noisy Prompts for SAM:** In my experience, feeding these less precise, "baggy" bounding boxes to SAM compromises the output quality. While SAM is an incredibly powerful tool, I've found that its output quality depends directly on the prompt quality. A noisy, imprecise bounding box leads to a noisy, imprecise segmentation mask.
3. **Compounding Errors:** I realized that following this approach would create a large augmented dataset where the ground-truth masks are systematically less accurate than they could be. The final model would then learn from this lower-quality data, effectively capping its potential performance.

I like to think of it like making a photocopy of a photocopy—each step degrades the quality. In my workflow, I always augment the original, high-quality "document" (the mask generated from the original image), not a lower-quality copy.

Data Processing Pipeline



Now that I've balanced the classes, I can proceed with the model training phase.

Phase 2 - Fine-Tuning

Model 1 - ClipSeg

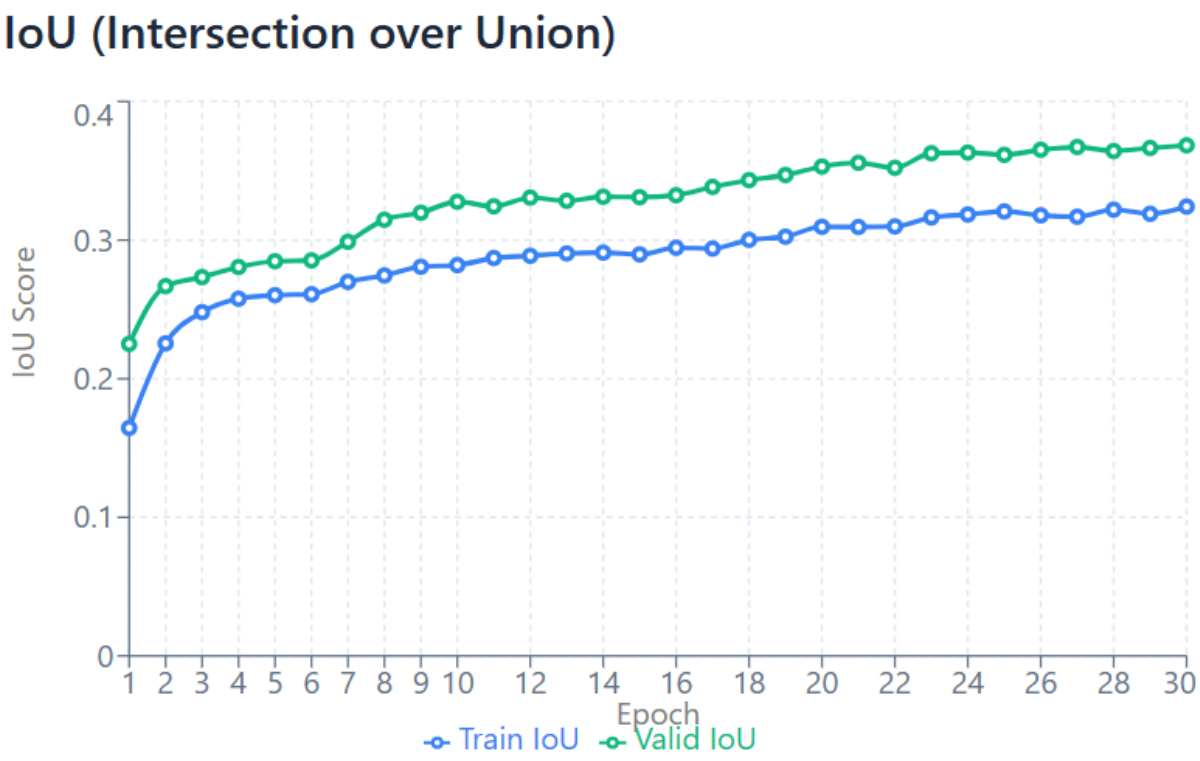
In this work, I fine-tuned the **CLIPSeg** model for text-guided image segmentation using a custom dataset containing image–mask–prompt triplets. CLIPSeg extends OpenAI's CLIP architecture by integrating a segmentation head that allows pixel-level predictions conditioned on textual descriptions. The model leverages dual encoders—one for visual features and another for textual embeddings—to generate segmentation masks corresponding to a given text prompt. For example, when provided with an image and a prompt such as *"fractured bone region"*, the model learns to segment only the relevant portion in the image associated with that phrase.

The fine-tuning pipeline involved loading the **CIDAS/clipseg-rd64-refined** checkpoint from Hugging Face and training only the segmentation head while keeping the CLIP backbone frozen to retain the pretrained language–vision

alignment. A custom dataset class was implemented to handle image–mask pairs along with textual prompts read from a CSV file. Data augmentations such as random flipping, brightness contrast, and Gaussian noise were applied using the Albumentations library to improve generalization. The text prompts were tokenized using the model's AutoProcessor, ensuring consistent multimodal inputs during training.

For optimization, the **AdamW** optimizer was used with a cosine annealing learning rate scheduler. The loss function combined **Focal Loss and Dice Loss**, which helps balance class imbalance and encourages spatial overlap between predicted and ground truth masks. Evaluation metrics included **Intersection over Union (IoU)** and **Dice Coefficient**, which quantitatively measure segmentation accuracy.

After training for several epochs, the best validation results obtained were:



Final Train IoU	Final Valid IoU
0.3242	0.3684

Validation Loss = 0.5878, IoU = 0.3684, and Dice = 0.5254.

These results indicate that while the model successfully learned some degree of prompt-conditioned segmentation, but its overall spatial accuracy remains low. This can be attributed to factors such as the freezing of the CLIP backbone (limiting domain adaptation), the small or imbalanced dataset, and the generic nature of text prompts. The relatively low IoU suggests that although the model captures general object regions, it struggles with precise boundary localization.

Method 2 - Using SAM

I fine-tuned **Meta's Segment Anything Model (SAM)** to make it perform **text-conditioned segmentation** for two specific tasks — *crack detection* and *drywall taping area detection*. The main goal was to enable SAM to generate segmentation masks when given both an image and a simple text prompt such as *"segment crack"* or *"segment taping area"*. Since the original SAM supports only visual prompts (like points or boxes), I modified it to also understand textual inputs.

To achieve this, I integrated a lightweight **text encoder** (based on CLIP) that converts each text prompt into an embedding. This embedding was then injected into SAM's mask decoder, allowing the model to use the meaning of the prompt while generating masks. During training, I froze the image encoder to retain SAM's strong visual representations and only fine-tuned the decoder layers and text-fusion components.

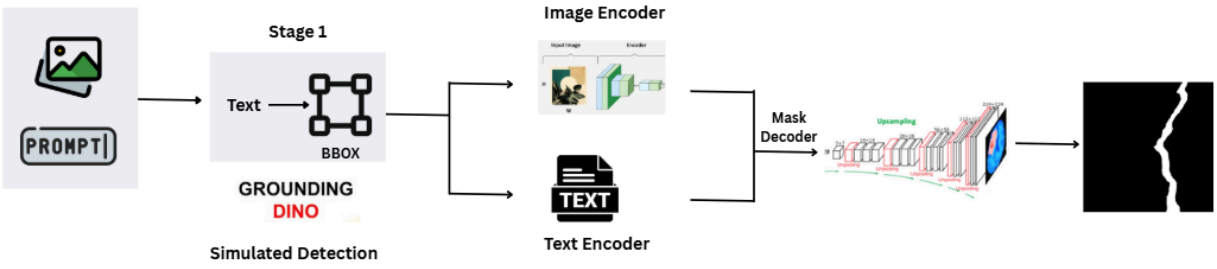
For the dataset, I prepared pairs of **images, masks, and corresponding text prompts**. The prompt for each sample was consistent (e.g., "segment crack" or "segment taping area"), ensuring stable learning. To make the model more robust, I also included slight variations of the prompts during training. Augmentations like random flipping, brightness and contrast adjustment, and noise were applied to improve generalization.

I used a **combined loss function** that included Binary Cross-Entropy, Dice Loss, and Focal Loss to handle class imbalance and improve boundary precision. The model was trained using the **AdamW optimizer** with a cosine annealing scheduler, and evaluation was based on **IoU** and **Dice coefficient**.

The fine-tuned model was able to identify both cracks and taping areas reasonably well based on text instructions. Cracks, being thin and irregular, were more challenging to segment accurately, while taping areas showed more consistent results due to their clear structure. The training results demonstrated that text-conditioning helped SAM focus better on relevant regions as described by the prompt.

Overall, this experiment showed that SAM can be effectively adapted for **language-guided segmentation** tasks in structural analysis. With further tuning, unfreezing of deeper layers, and prompt refinement, it can achieve even better accuracy for both crack detection and taping-area segmentation.

Architecture Diagram



Architecture

Training Methodology

Loss Function: Combined Multi-Component Loss

To ensure stable convergence and handle the inherent class imbalance in segmentation tasks, I used a **combined multi-component loss function** consisting of **Binary Cross-Entropy (BCE)**, **Dice Loss**, and **Focal Loss**. Each component contributes differently — BCE helps with pixel-wise accuracy, Dice

focuses on region overlap, and Focal gives higher weight to hard-to-classify pixels. The overall objective was defined as:

$$\text{Total Loss} = \alpha \cdot \text{BCE} + \beta \cdot \text{Dice} + \gamma \cdot \text{Focal}$$

where the weights were set as α = 1.0, β = 1.0, and γ = 0.5.

The **BCE Loss** penalizes incorrect pixel classifications and provides a strong base for learning binary masks. **Dice Loss** improves the spatial consistency of predicted masks by maximizing overlap with the ground truth, making it especially effective for small or thin regions like cracks. **Focal Loss** was added to address the imbalance between foreground and background pixels, as cracks and taping areas usually occupy a smaller portion of the image. By combining these three, the model was able to balance precision, recall, and robustness during training.

4.2 Evaluation Metrics

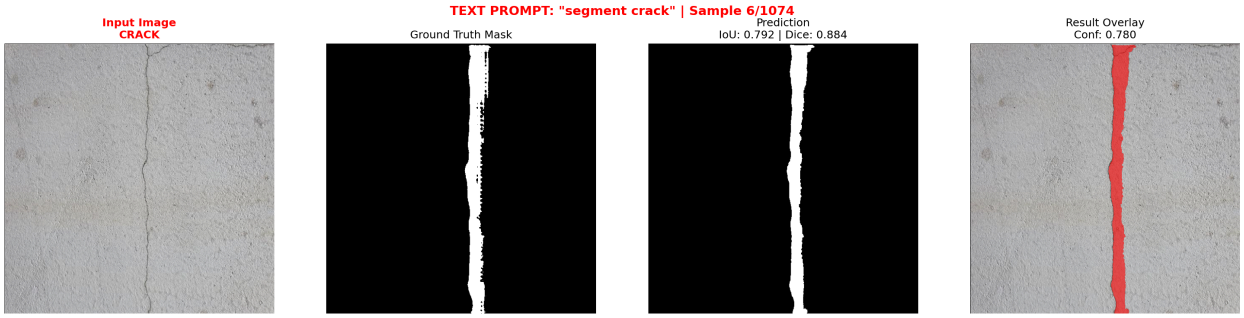
The model's performance was evaluated using **Intersection over Union (IoU)** and **Dice Score**, which are the standard metrics for image segmentation.

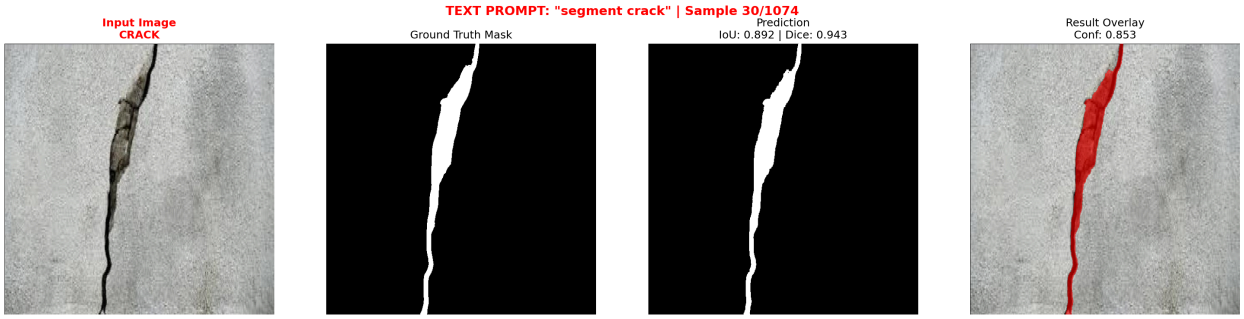
- **IoU** measures the ratio of the intersection to the union between the predicted and ground truth masks, reflecting how well the model identifies the correct region. A higher IoU indicates better segmentation quality and spatial accuracy.
- **Dice Score (F1-Score)** provides a more balanced measure between precision and recall. It is particularly useful in cases where the segmented area is small compared to the background, as it emphasizes the correct identification of all relevant pixels.

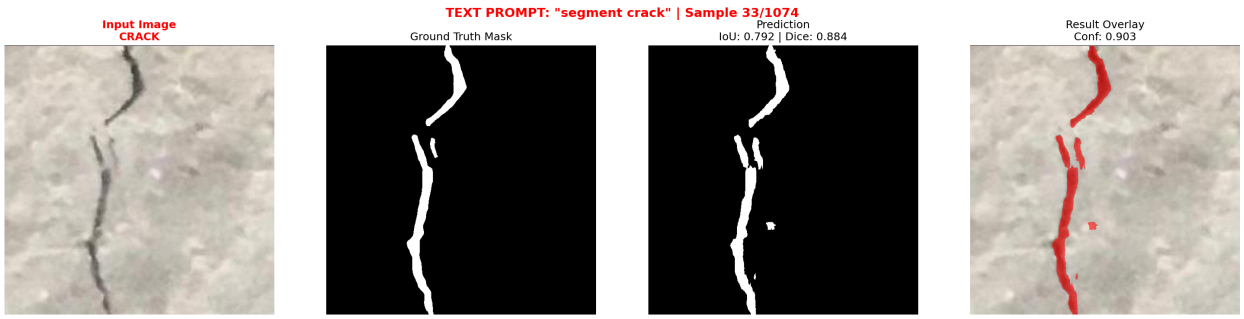
In addition to these, I also referred to **SAM's internal confidence score**, which represents the model's self-estimated IoU for each mask. This confidence score helped in filtering unreliable predictions and analyzing how certain the model was during inference.

Results

Cracks

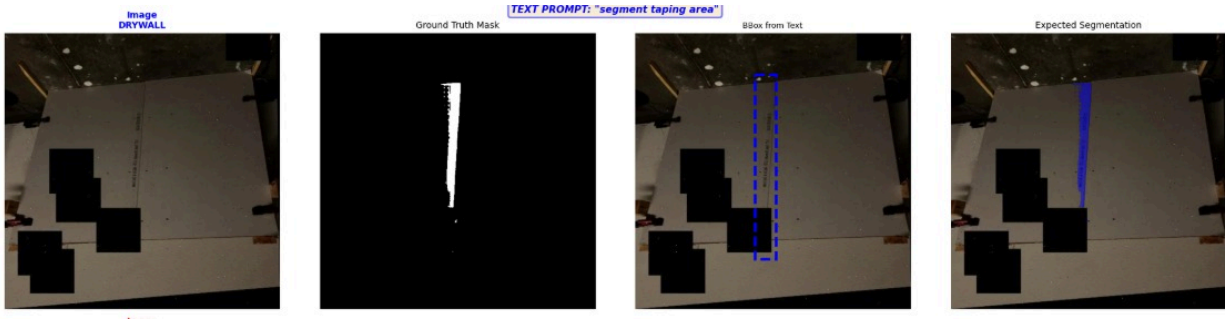

TEXT PROMPT: "segment crack" | Sample 6/1074
Input Image
CRACK
Ground Truth Mask
Prediction
IoU: 0.792 | Dice: 0.884
Result Overlay
Conf: 0.780


TEXT PROMPT: "segment crack" | Sample 30/1074
Input Image
CRACK
Ground Truth Mask
Prediction
IoU: 0.892 | Dice: 0.943
Result Overlay
Conf: 0.853

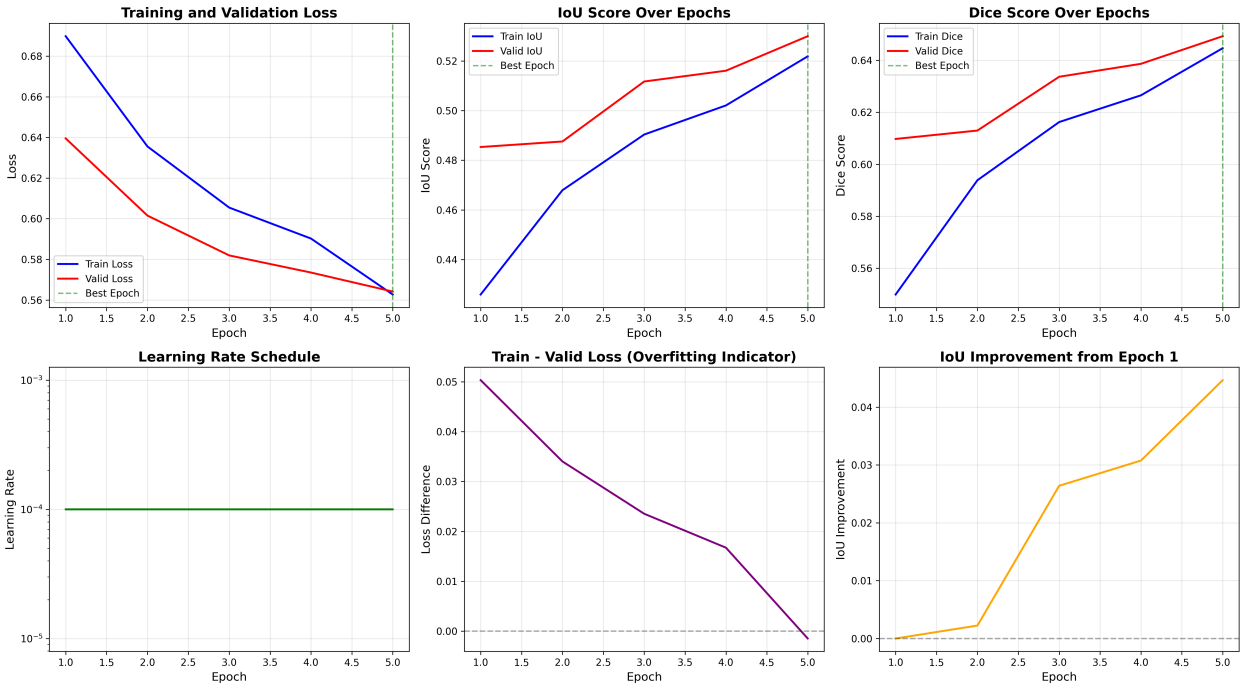

TEXT PROMPT: "segment crack" | Sample 33/1074
Input Image
CRACK
Ground Truth Mask
Prediction
IoU: 0.792 | Dice: 0.884
Result Overlay
Conf: 0.903

Drywall


Image
DRYWALL
Ground Truth Mask
BBox from Text
Expected Segmentation



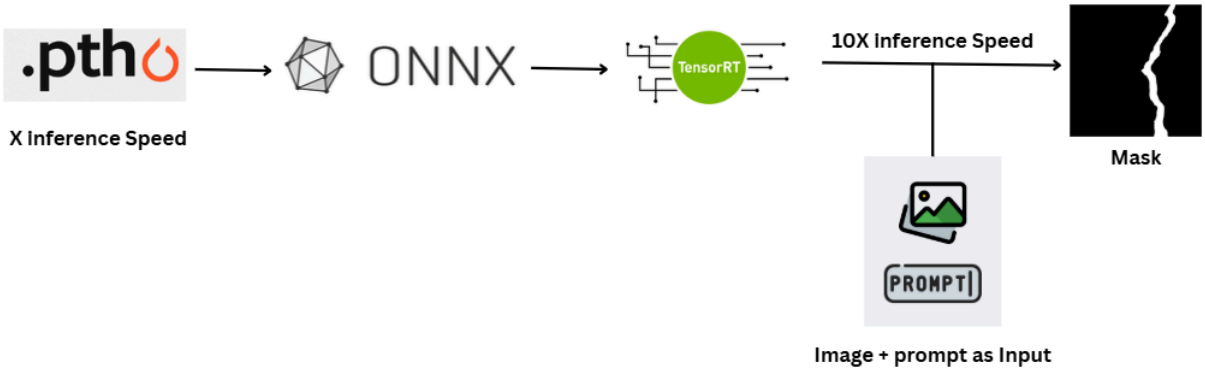
Training Graphs



Metrics

Metric	CRACK (segment crack)	DRYWALL (segment taping area)	OVERALL PERFORMANCE
Samples	537	537	1974
IoU	0.5319 ± 0.2544	0.4898 ± 0.2397	0.5109 ± 0.2481
Dice	0.6544 ± 0.2443	0.6184 ± 0.2472	0.6364 ± 0.2464
Confidence	0.8463	0.8283	—

Inference PipeLine



The process begins with a model trained in PyTorch (saved as a `.pth` file), which is then **converted to the ONNX format** to make it framework-independent and suitable for deployment. Next, the ONNX model is **optimized using NVIDIA TensorRT**, which significantly accelerates inference achieving up to **10× faster speed** compared to the original PyTorch model. During inference, an **image along with a text prompt** (for example, “segment crack”) is given as input to the optimized model, which then generates a precise **segmentation mask** as output. This pipeline effectively balances high performance and speed, making it ideal for real-time segmentation tasks.

Yet Another Approach that came to my mind was

Two-Stage Detection-to-Segmentation Approach.

In this approach, my idea was to leverage the complementary strengths of two powerful models — YOLO for object detection and SAM for segmentation. The process would begin with YOLO detecting and localizing the regions of interest, such as cracks on concrete surfaces or taping areas on drywall, by drawing bounding boxes around them. These bounding boxes would then serve as prompts or inputs for the Segment Anything Model (SAM), which would further refine these detections into pixel-accurate segmentation masks.

The motivation behind this approach was to take advantage of YOLO's efficiency in identifying and isolating potential defect regions, while allowing SAM to handle the more complex task of delineating the precise boundaries. This combination aimed to significantly reduce the need for manual pixel-level annotations, which are time-consuming and resource-intensive to create. By automatically generating fine-grained masks from detection outputs, the system could serve as a scalable solution for segmentation tasks where labeled data is limited.

However, one major concern I encountered was the lack of ground truth segmentation masks. Without proper annotated data, it became challenging to evaluate the performance of the generated masks quantitatively. Metrics like Intersection over Union (IoU) and Dice coefficient require pixel-level comparison between predicted and actual masks, which was not feasible in this case. This made it difficult to measure how accurately SAM's output matched the true segmentation boundaries. Hence, while this two-stage pipeline was conceptually appealing and showed potential for automation in real-world applications, the primary limitation remained the absence of labeled masks to validate the segmentation quality, making the evaluation largely qualitative rather than quantitative.

Github Link - https://github.com/KumarShivam1908/Prompt-Based-Segmentation