Politecnico di Torino

Department of Control and Computer Engineering

Master Degree Thesis

Title:

Few Shot Adaptation of VLM for Panoptic Segmentation

Masoud Karimi

Supervised by

Andrea Bottino

Co-Supervisor

Federico D'Asaro

Table of Contents

Few Shot Adaptation of VLM for Panoptic Segmentation

July 11, 2024

Abstract

In this master thesis, an approach is presented to fine-tune a foundation model for image segmentation called the Segment Everything Everywhere Model (SEEM) (90) using adapters, which are lightweight modules incorporated into both the segmentation head and vision backbone to adapt to new tasks. Recognizing the need for a large amount of annotated data for fine-tuning SEEM and the time-consuming nature of data annotation, generative AI specifically Stable Diffusion (SD) (65) model is employed to create a dataset comprising images and corresponding masks for fine-tuning SEEM. The fine-tuning results indicate that integrating adapters into the vision backbone significantly improves performance, yielding up to a 15% enhancement compared to using adapters solely in the segmentation head. Furthermore, synthetic datasets generated by generative AI demonstrated their benefits, resulting in a notable 13% improvement in segmentation task performance.
The project codes for fine-tuning SEEM https://github.com/msdkarimi/Segment-Everything-Everywhere-All-At-Once
The project codes for generating annotated dataset using Stable diffusionhttps://github.com/msdkarimi/synthetic-dataset

1 Introduction

The rapid advancement of foundational models in artificial intelligence (AI) has revolutionized natural language processing (NLP), computer vision, and other domains. These models, pre-trained on massive datasets, provide a powerful starting point due to their extensive pre-training. In computer vision, foundational models have significantly enhanced image segmentation tasks. Notable models like the Segment Anything Model (SAM) (42)

and Segment Everything Everywhere Model (SEEM) (90) exemplify this progress, offering robust generalization across various tasks.

However, fine-tuning these models often necessitates a substantial amount of annotated data, which is both time-consuming and resource-intensive to produce. Despite the benefits of extensive pre-training and transfer learning, achieving optimal performance still requires a considerable amount of annotated examples.

This master thesis focuses on fine-tuning SEEM, a foundational model, to perform image segmentation using prompts or in an automatic manner. Given the constraints of limited annotated data and the labor-intensive nature of manual annotation, we explore the use of generative AI to overcome these limitations. Specifically, we utilize the Stable Diffusion model (65) to generate images and their corresponding segmentation masks. Although the Stable Diffusion model produces high-quality images, it does not directly generate segmentation masks. To address this, we extract masks of the classes of interest from the model's internal representations. The generated dataset of images and masks is then used to fine-tune the SEEM model for specific tasks.

This thesis evaluates the performance of a fine-tuned SEEM model on real-world images, with the goal of assessing how well the generated dataset aligns with the target task's data distribution. The primary objectives are twofold: first, to fine-tune the SEEM model by incorporating adapters at various points, and second, to leverage generative AI for creating a synthetic dataset.

Ultimately, the thesis aims to determine the effectiveness of the fine-tuned model in addressing real-world problems. It seeks to provide insights into the alignment and performance of the generated dataset compared to the original task's data distribution. By using foundational models and generative AI, this approach aims to reduce the reliance on labor-intensive data annotation while maintaining high performance in image segmentation tasks.

2 State of the Art

2.1 Image Segmentation

Image segmentation is a key topic in image processing and computer vision with applications such as scene understanding, medical image analysis, robotic perception, video surveillance, augmented reality, and image compression, among many others. Various algorithms for image segmentation have been developed in the literature. Recently, due to the success of deep learning models in a wide range of vision applications, there has been a substantial amount of works aimed at developing image segmentation approaches using deep learning models.

2.1.1 Background

Image segmentation is an essential component in many visual understanding systems. It involves partitioning images (or video frames) into multiple segments or objects. Segmentation plays a central role in a broad range of applications, including medical image analysis (e.g., tumor boundary extraction and measurement of tissue volumes), autonomous vehicles (e.g., navigable surface and pedestrian detection), video surveillance, and augmented reality to count a few.

Numerous image segmentation algorithms have been developed in the literature, from the earliest methods, such as thresholding (58), histogram-based bundling, region- growing (56), k-means clustering (53), watersheds (52), to more advanced algorithms such as active contours (40), graph cuts (5) and sparsity- based (71)-(51) methods.

Over the past few years, however, deep learning (DL) models have yielded a new generation of image segmentation models with remarkable performance improvements often achieving the highest accuracy rates on popular benchmarks resulting in a paradigm shift in the field.

Image segmentation can be formulated as a classification problem of pixels with semantic labels (semantic segmentation) or partitioning of individual objects (instance seg-

mentation). Semantic segmentation performs pixel-level labeling with a set of object categories (e.g., human, car, tree, sky) for all image pixels, thus it is generally a harder undertaking than image classification, which predicts a single label for the entire image. Instance segmentation extends semantic segmentation scope further by detecting and delineating each object of interest in the image (e.g., partitioning of individual persons).

Image segmentation is arguably the most important yet challenging problem in computer vision. In the past, we have witnessed significant progress in a wide range of segmentation tasks including instance, semantic and panoptic segmentation ((46), (14), (81), (28))

Most recently, we are observing a clear trend toward more flexible segmentation models in different aspects:

- **From closed-set to open-vocabulary segmentation**. Many recent works proposed to either leverage contrastive learning methods or pretrained multi-modal foundation models (e.g., CLIP (60)) to make the segmentation models more transferable to unseen concepts ((22), (89), (83));
- **From generic to referring segmentation**. In addition to generic segmentation that segments an image thoroughly given a predetermined set of concepts, language-based referring segmentation provides a user-friendly way of segmenting a specific region referred by an arbitrary text phrase ((86), (37), (49));
- **From one-shot to interactive segmentation**. In practice, segmentation models do not necessarily produce satisfactory masks in one round. As such, people are also studying how to progressively refine the segmentation results through intimate interactions between humans and models ((15), (50), (12))

Overview of Deep Neueal Networks This section provides an overview of some of the most prominent deep learning architectures used by the computer vision community, including convolutional neural networks (CNNs) (44), recurrent neural networks (RNNs) and long short term memory (LSTM) (34) and encoder-decoders (2).

It is worth mentioning that in some cases the DL-models can be trained from scratch

on new applications/datasets (assuming a sufficient quantity of labeled training data), but in many cases there are not enough labeled data available to train a model from scratch and one can use transfer learning to tackle this problem. In transfer learning, a model trained on one task is re-purposed on another (related) task, usually by some adaptation process toward the new task. For example, one can imagine adapting an image classification model trained on ImageNet to a different task, such as texture classification, or face recognition. In image segmentation case, many people use a model trained on ImageNet (a larger dataset than most of image segmentation datasets), as the encoder part of the network, and re-train their model from those initial weights. The assumption here is that those pre-trained models should be able to capture the semantic information of the image required for segmentation, and therefore enabling them to train the model with less labeled samples.

Convolutional Neural Networks (CNNs) CNNs are among the most successful and widely used architectures in the deep learning community, especially for computer vision tasks. CNNs were initially proposed (24), based on the hierarchical receptive field model of the visual cortex.

Subsequently, Waibel et al. (80) introduced CNNs with weights shared among temporal receptive fields and backpropagation training for phoneme recognition, and LeCun et al. (44) developed a CNN architecture for document recognition Fig. 1.

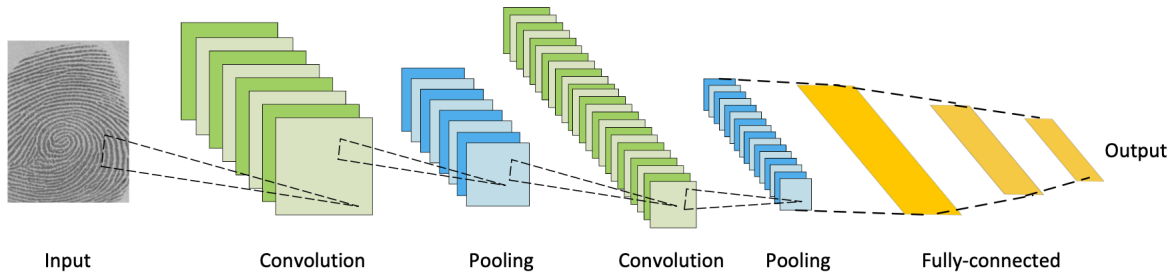


Figure 1: Architecture of convolutional neural networks. From (44).

CNNs mainly consist of three type of layers:

- convolutional layers, where a kernel (or filter) of weights is convolved in order to extract features

- nonlinear layers, which apply an activation function on feature maps (usually element-wise) in order to enable the modeling of non-linear functions by the network
- pooling layers, which replace a small neighborhood of a feature map with some statistical information (mean, max, etc.) about the neighborhood and reduce spatial resolution

The units in layers are locally connected; that is, each unit receives weighted inputs from a small neighborhood, known as the receptive field, of units in the previous layer. By stacking layers to form multi-resolution pyramids, the higher-level layers learn features from increasingly wider receptive fields. The main computational advantage of CNNs is that all the receptive fields in a layer share weights, resulting in a significantly smaller number of parameters than fully-connected neural networks. Some of the most well-known CNN architectures include: AlexNet (43), VGGNet (69) and ResNet (31).

Recurrent Neural Networks (RNNs) and the LSTM RNNs (67) are widely used to process sequential data, such as speech, text, videos, and time-series, where data at any given time/position depends on previously encountered data. At each time-stamp the model collects the input from the current time X_i and the hidden state from the previous step h_{i-1} , and outputs a target value and a new hidden state Fig. 2.

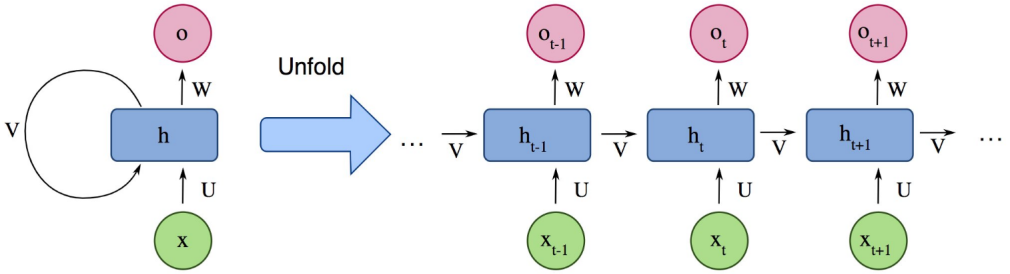


Figure 2: Architecture of a simple recurrent neural network.

RNNs are typically problematic with long sequences as they cannot capture long-term dependencies in many real-world applications (although they exhibit no theoretical limitations in this regard) and often suffer from gradient vanishing or exploding problems. However, a type of RNNs called Long Short Term Memory (LSTM) (34) is designed to avoid these issues.

The LSTM architecture Fig. 3 includes three gates (input gate, output gate, forget gate), which regulate the flow of information into and out from a memory cell, which stores values over arbitrary time intervals.

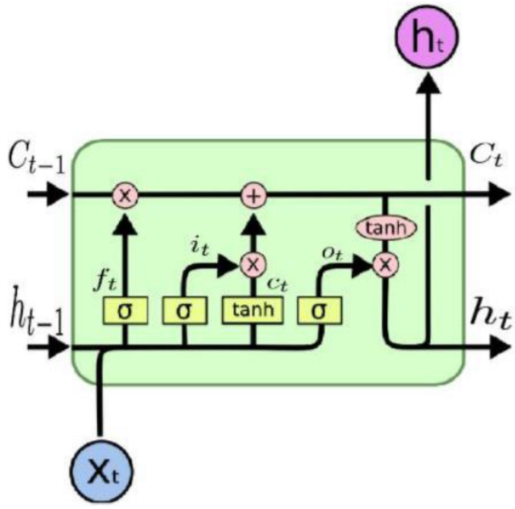


Figure 3: Architecture of a standard LSTM module

Encoder-Decoder and Auto-Encoder Models Encoder-Decoder models are a family of models which learn to map data-points from an input domain to an output domain via a two-stage network: The encoder, represented by an encoding function $z = f(x)$, compresses the input into a latent-space representation; the decoder, $y = g(z)$, aims to predict the output from the latent space representation (2), (27).

The latent representation here essentially refers to a feature (vector) representation, which is able to capture the underlying semantic information of the input that is useful for predicting the output. These models are extremely popular in image-to-image translation problems, as well as for sequence-to-sequence models in NLP. Fig. 4 illustrates the block-diagram of a simple encoder-decoder model.

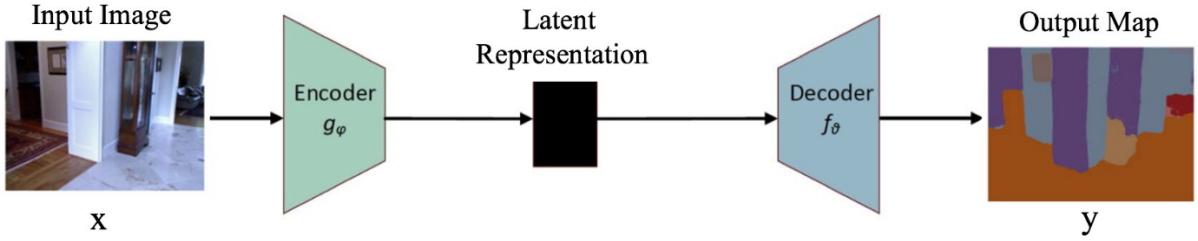


Figure 4: The architecture of a simple encoder-decoder model

A powerful and flexible architecture for image segmentation tasks is U-Net (66). The U-Net model is a convolutional neural network architecture that was originally designed for biomedical image segmentation. Fig. 5 The U-Net characterized by its encoder-decoder structure, use of skip connections, and efficiency with limited data. Its ability to combine high-level and low-level features makes it a popular choice for various applications beyond its original domain of biomedical imaging.

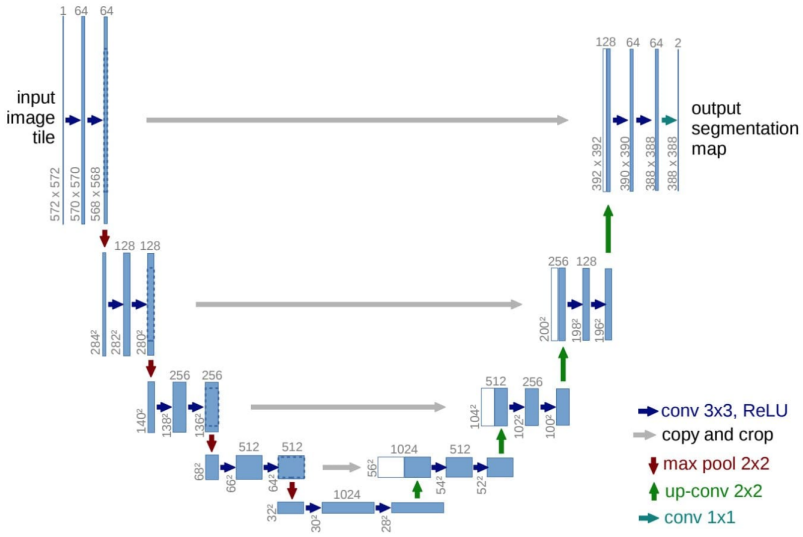


Figure 5: The U-net model. The blue boxes denote feature map blocks with their indicated shapes

Transformer-based Models : Vanilla Transformer (79) is a seminal model in the transformer-based research field. It is an encoder-decoder structure that takes tokenized inputs and consists of stacked transformer blocks. Each block has two sub-layers: a multi- head self-attention (MHSA) layer and a position-wise fully- connected feed-forward network (FFN). The MHSA layer allows the model to attend to different parts of the input sequence while the FFN processes the output of the MHSA layer. Both sub-layers use residual connections and layer normalization for better optimization.

In the vanilla transformer, the encoder and decoder both use the same architecture. However, the decoder is modified to include a mask that prevents it from attending to future tokens during training. Additionally, the decoder uses sine and cosine functions to produce positional embeddings, which allow the model to understand the order of the input sequence. Subsequent models such as BERT and GPT- 2 have built upon its architecture and achieved state-of-the- art results on a wide range of natural language

processing tasks.

- **Self-Attention**: he core operator of the vanilla trans- former is the self-attention (SA) operation. Suppose the input data is a set of tokens $X = [x_1, x_2, ..., x_N] \in \mathbb{R}^{N \times c}$. N is the token number and c is token dimension. The positional encoding P may be added into $I = X + P$.

  The input embedding I goes through three linear projection layers ($W^q \in \mathbb{R}^{c \times d}, W^k \in \mathbb{R}^{c \times d}, W^v \in \mathbb{R}^{c \times d}$) to generate Query (Q), Key (K), and Value (V):

$$Q = IW^q, K = IW^k, V = IW^v, \tag{1}$$

  where d is the hidden dimension. The Query and Key are usually used to generate the attention map in SA. Then the SA is performed as follows:

$$O = SA(Q, K, V) = Softmax(QK)V. \tag{2}$$

  According to Equ. 2, given an input X, self-attention allows each token x_i to attend to all the other tokens. Thus, it has the ability of global perception compared with local CNN operator.

- **Multi-Head Self-Attention**: In practice, multi-head self-attention (MHSA) is more commonly used. The idea of MHSA is to stack multiple SA sub-layer in parallel, and the concatenated outputs are fused by a projection matrix $W^{fuse} \in \mathbb{R}^{d \times c}$:

$$O = MHSA(Q, K, V) = concat([SA_i, ..SA_H])W^{fuse}, \tag{3}$$

  where $SA_i = SA(Q_i, K_i, V_i)$ and H is the number of the head. Different heads have individual parameters. Thus, MHSA can be viewed as an ensemble of SA.

- **Feed-Forward Network**: The goal of feed-forward net- work (FFN) is to enhance the non-linearity of attention layer outputs. It is also called multi-layer perceptron (MLP) since it consists of two successive linear layers with non-linear activation layers.

Meta-Architecture

- **Backbone**: Before ViTs, CNNs were the standard approach for feature extraction in computer vision tasks. To ensure a fair comparison, many research works (7) and (29) used the same CNN models, such as ResNet50 (30). ViT, utilizes a standard transformer encoder for feature extraction. It has a specific input pipeline for images, where the input image is split into fixed-size patches, such as 16×16 patches. These patches are then processed through a linear embedding layer. Then, the positional embeddings are added to each patch. Afterward, a standard transformer encoder encodes all patches. It contains multiple multi-head self-attention and feed-forward layers.

  For instance, given an image $I \in \mathbb{R}^{H \times W \times 3}$, ViT first reshapes it into a sequence of flattened 2D patches: $I_p \in R^{N \times P^2 \times 3}$, where N is the number of patches and P is the patch size. With patch embedding operations, the final input is $I_{in} \in \mathbb{R}^{N \times P^2 \times C}$, where C is the embedding channel. To perform classification, an extra learnable embed- ding "classification token" (CLS) is added to the sequence of embedded patches. After the standard transformer for all patches, $I_{out} \in \mathbb{R}^{N \times P^2 \times C}$ is obtained. For segmentation tasks, ViT is used as a feature extractor, meaning that I_{out} is resized back to a dense map $F \in \mathbb{R}^{H \times W \times C}$.

- **Neck**: Feature pyramid network (FPN) has been shown effective in object detection and instance segmentation (47), (48) and (74) for scale variation modeling.

  FPN maps the features from different stages into the same channel dimension C for the decoder. Several works (59), (26) design stronger FPNs via cross-scale modeling using dilation or deformable convolution.

  For example, Deformable DETR (88) proposes a deformable FPN to model cross-scale fusion using deformable attention. Lite-DETR (45) further refines the deformable cross-scale attention design by efficiently sampling high-level features and low-level features in an interleaved manner. The output features are used for decoding the boxes and masks. The role of FPN is the same as previous detection-based or FCN-based segmentation methods.

- **Object Query**: Object query is first introduced in DETR (7). It plays as the

dynamic anchors that are used in detectors (29), (63). In practice, it is a learnable embedding $Q_{obj} \in \mathbb{R}^{N_{ins} \times d}$. N_{ins} represents the maximum instance number. The query dimension d is usually the same as feature channel c. Object query is refined by the cross-attention layers. Each object query represents one instance of the image. During the training, each ground truth is assigned with one corresponding query for learning. During the inference, the queries with high scores are selected as output. Thus, object query simplifies the design of detection and segmentation models by eliminating the need for hand-crafted components such as non-maximum suppression (NMS).

- **Transformer Decoder**: Transformer decoder is a crucial architecture component in transformer-based segmentation and detection models. Its main operation is cross-attention, which takes in the object query Q_{obj} and the image/video feature F.

  It outputs a refined object query, denoted as Q_{out}. The cross-attention operation is derived from the vanilla transformer architecture, where Q_{obj} serves as the query, and F is used as the key and value in the self-attention mechanism.

  After obtaining the refined object query Q_{out}, it is passed through a prediction FFN, which typically consists of a 3-layer perceptron with a ReLU activation layer and a linear projection layer. The FFN outputs the final prediction, which depends on the specific task. For example, for classification, the refined query is mapped directly to class prediction via a linear layer. For detection, the FFN predicts the normalized center coordinates, height, and width of the object bounding box. For segmentation, the output embedding is used to perform dot product with feature F , which results in the binary mask logits. The transformer decoder iteratively repeats cross-attention and FFN operations to refine the object query and obtain the final prediction. The intermediate predictions are used for auxiliary losses during training and discarded during inference. The outputs from the last stage of the decoder are taken as the final detection or segmentation results. The detailed process is shown in Fig. 6 (b).

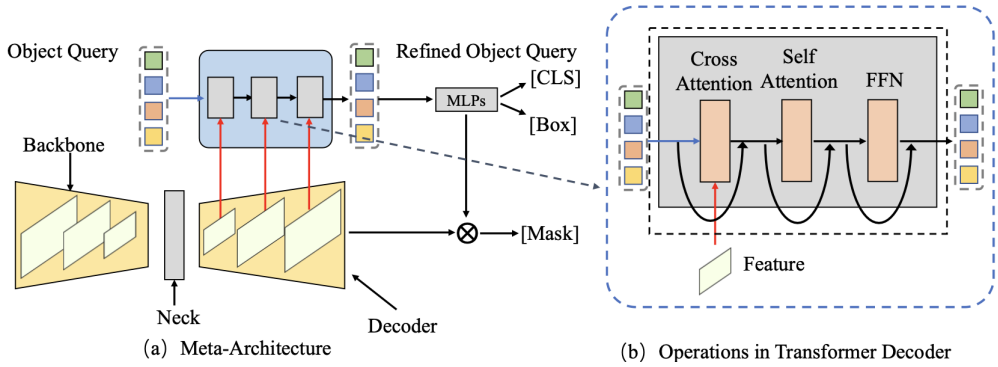


Figure 6: Illustration of (a) meta-architecture and (b) common operations in the decoder.

- **Mask Prediction Representation**: Transformer-based segmentation approaches adopt two formats to represent the mask prediction: pixel-wise prediction as FCNs and per- mask-wise prediction as DETR. The former is used in semantic-aware segmentation tasks, including SS, VSS, VOS, and etc. The latter is used in instance-aware segmentation tasks, including IS, VIS, and VPS, where each query represents each instance.

Dataset :

Defining Things and Stuff : Semantic classes can be either things or stuff. The literature provides definitions for several aspects of stuff and things, including:

- **Shape:** Things have characteristic shapes such as car, cat, phone, whereas stuff is amorphous such as sky, grass, water (6). Stuff mostly exhibits as colors or textures and has less well-defined shapes (19).
- **Size:** Things occur at characteristic sizes with little variance, whereas stuff regions are highly variable in size (6). An stuff is defined by a homogeneous or repetitive pattern of fine-scale properties, but has no specific or distinctive spatial extent or shape. A thing has a specific size and shape (32).
- **Parts:** Thing classes have identifiable parts, whereas stuff classes do not (e.g. a piece of grass is still grass, but a wheel is not a car) (6).

- **Instances:** Stuff classes are typically not countable, and have no clearly defined instances. Whilst on the other hand, things are countable, such that we can find multiple person or can in a scene (19)

- **Texture:** Stuff classes are typically highly textured, so they have no consistent shape but fairly consistent texture (75)

  Several works have shown that different techniques are required for the detection of stuff and things. Moreover, several works have shown that stuff is a useful contextual cue to detect things and vice versa (6).

  Stuff classes should (1) be mutually exclusive; (2) in their ensemble, cover the vast majority of the stuff surface appearing in the dataset; (3) be frequent enough; (4) have a good level of granularity, around the base level for a human. However, these criteria conflict with each other: if we label all vegetations as vegetation, the labels are too general. On the other extreme, if we create a separate class for every single type of vegetation, the la- bels are too specific and infrequent (6).

  Although stuff classes cover the majority of the image surface, one might argue they are just irrelevant background pixels. The COCO dataset is annotated with five captions per image, which have been written explicitly to describe its content, and therefore capture the most relevant aspects of the image for a human. To emphasize the importance of stuff for scene understanding (6). analyze these captions, counting how many nouns point to things and stuff respectively. They use a Part-Of- Speech (POS) tagger (77) to automatically detect nouns.

  Results indicate that stuff covers more than a third of the nouns (38.2%). This clearly shows the importance of stuff according to the COCO image captions.

- **Spatial Context between Stuff and Things:** (6) analyze spatial context by considering the relative image position of one class with respect to another. For simplicity, here they explain how to compute the spatial context for one particular reference class, i.e. car. For every image containing a car, they extract a set of car regions, i.e. connected components of car pixels in the annotation map.

  Next they compute a histogram of image pixels surrounding the car regions, with two spatial dimensions (distance, angle) and one dimension for the class label. To

determine in which spatial bin a certain pixel lands, they (1) compute the distance between the pixel and the nearest point in the car region (normalized by image size); (2) compute the relative angle with respect to the cen- ter of mass of the car region.

Results reveals several interesting contextual relations. For instance, Trains are typically found above railroads (thing-stuff). TVs are typically found in front of persons (thing-thing). Tiled walls occur above tiled floors (stuff-stuff), and roads are flanked by persons on both sides (stuff-thing). Note that these contextual relations are not necessarily symmetric: most cars appear above a road, but many roads have other things above them.

- **Importance of stuff and things:** In terms of surface cover, (6) measure the frequencies of stuff and thing pixels in the COCO-Stuff annotations. Results show that the majority of pixels are stuff (69.1%). They also compute statistics for the labeled regions in COCO-Stuff, i.e., connected components in the pixel annotation map. They use such regions as a proxy for class instances, as stuff classes do not have instances. They found that 69.4% of the regions are stuff and 30.6% things.

2.1.2 State Of The Art models

Segment Anything Model : The Segment Anything Model (SAM) (42) is an innovative image segmentation model developed by Meta AI Research, renowned for its versatility and powerful zero-shot learning capabilities. The model is designed to segment any object in an image based on various types of user prompts, including masks, boxes, and points. SAM's architecture leverages a large dataset, consisting of over 1 billion masks from 11 million images, which allows it to generalize well across diverse segmentation tasks without the need for additional training. This foundational aspect makes SAM particularly useful for applications where rapid and accurate segmentation is required without the availability of task-specific training data.

Architecture and Technology : SAM employs a transformer-based architecture, which is particularly adept at capturing long-range dependencies and contextual information in images. This architecture leverages self-attention mechanisms to process and

integrate information from various parts of the image, ensuring comprehensive feature extraction. Additionally, SAM incorporates a multi-task learning framework, allowing it to jointly learn from various segmentation tasks. This framework enhances its ability to generalize across different types of segmentation challenges.

Key Features and Performance : One of SAM's standout features is its ability to accept diverse input prompts, such as points, boxes, and masks, to guide the segmentation process. This flexibility makes SAM exceptionally user-friendly and adaptable to different user needs and application contexts. In terms of performance, SAM has demonstrated high accuracy across a wide range of benchmarks and datasets. It is optimized for both speed and computational efficiency, making it suitable for real-time applications and scalable deployment in various environments.

Applications and Impact : SAM's versatility makes it applicable across numerous domains. In general computer vision, it can be used for object detection, recognition, and scene understanding. In the medical field, SAM can assist in accurately segmenting anatomical structures and detecting lesions in medical images. For autonomous driving, SAM helps in identifying and segmenting objects in complex driving environments, contributing to safer and more reliable self-driving systems. Overall, SAM represents a significant advancement in making segmentation more accessible and practical for diverse real-world applications.

Segment Everything Everywhere Model : The Segment Everything Everywhere Model (SEEM) (90) is a cutting-edge image segmentation model designed to handle various segmentation tasks simultaneously. Developed to operate as a universal segmentation interface, SEEM incorporates diverse prompting mechanisms to achieve a high degree of versatility. The model's architecture supports different types of spatial queries, including points, boxes, scribbles, and masks, allowing it to generalize across multiple segmentation tasks without the need for extensive retraining. This makes SEEM an effective tool for applications requiring adaptive and prompt-based segmentation capabilities.

Architecture and Technology : SEEM is designed to serve as a universal image segmentation model by integrating a versatile and interactive architecture. The model's architecture is built upon a joint visual-semantic space, which allows for the dynamic composition of visual and text prompts, enhancing its adaptability and compositionality. Additionally, SEEM incorporates learnable memory prompts to retain segmentation history, improving consistency and accuracy over time. By leveraging a text encoder to encode text queries and mask labels into the same semantic space, SEEM achieves open-vocabulary segmentation, making it highly effective for various segmentation challenges with minimal supervision. This comprehensive design ensures SEEM's capability to perform interactive segmentation, generic segmentation, referring segmentation, and video object segmentation across multiple datasets, establishing it as a robust and versatile tool for image segmentation tasks.

Key Features and Performance :

- **Versatility**. SEEM introduces a new visual prompt to unify different spatial queries including points, boxes, scribbles and masks, which can further generalize to a different referring image.
- **Compositionality**: SEEM learns a joint visual-semantic space between text and visual prompts, which facilitates the dynamic composition of two prompt types required for various segmentation tasks.
- **Interactivity**: SEEM incorporates learnable memory prompts into the decoder to retain segmentation history through mask-guided cross-attention from decoder to image features
- **Semantic-awareness**: SEEM uses a text encoder to encode text queries and mask labels into the same semantic space for open-vocabulary segmentation.

Applications and Impact : SEEM's broad applicability makes it suitable for various domains, including medical imaging, remote sensing, and robotics. In medical imaging, SEEM can provide detailed and accurate segmentations of anatomical structures and

pathological findings, aiding in diagnosis and treatment planning. In remote sensing, it can analyze satellite images for environmental monitoring, urban planning, and disaster management. In robotics, SEEM's real-time segmentation capabilities are crucial for navigation and object manipulation tasks. By providing a robust and versatile segmentation solution, SEEM is poised to significantly impact numerous fields, enhancing the precision and efficiency of segmentation-related tasks.

Comparison Between SAM and SEEM : SEEM offers significant advantages over the SAM by integrating richer contextual understanding and more versatile segmentation capabilities. While SAM excels at identifying and segmenting any object within an image based on prompts, SEEM enhances this by combining promptable segmentation with additional contextual and semantic information, allowing it to not only segment objects but also understand their relationships and broader scene context. This integration enables SEEM to deliver more accurate and comprehensive segmentation results, making it more effective for complex visual tasks and applications that require nuanced understanding of scenes, such as autonomous driving, medical imaging, and advanced visual search systems.

2.2 Generative Modeling

With the development of neural networks and the increase in computational power, deep generative modeling has become one of the leading directions in AI. Its applications vary from typical modalities considered in machine learning, i.e., text analysis, image analysis, audio analysis, to problems in active learning, reinforcement learning, graph analysis and medical imaging (76).

In some applications, it is indeed important to generate (synthesize) objects or modify features of objects to create new ones. However, in others like active learning it is important to ask for uncertain objects, i.e., objects with low $p(x)$ that should be labeled by an oracle. In reinforcement learning, on the other hand, generating the next most likely situation (state) is crucial for taking actions by an agent. For medical applications, explaining a decision, e.g., in terms of the probability of the label and the object, is more informative to a human doctor than simply assisting with a diagnosis label. If an AI

system would be able to indicate how certain it is and quantify whether the object is suspicious (i.e., low $p(x)$) or not, then it might be used as an independent specialist that outlines its own opinion (76).

These examples clearly indicate that many fields, if not all, could highly benefit from (deep) generative modeling. Obviously, there are many mechanisms that AI systems should be equipped with. However, we claim that the generative modeling capability is definitely one of the most important ones, as outlined in the above-mentioned cases.

As Fig. 7 depicts, we can divide deep generative modeling into three main groups:

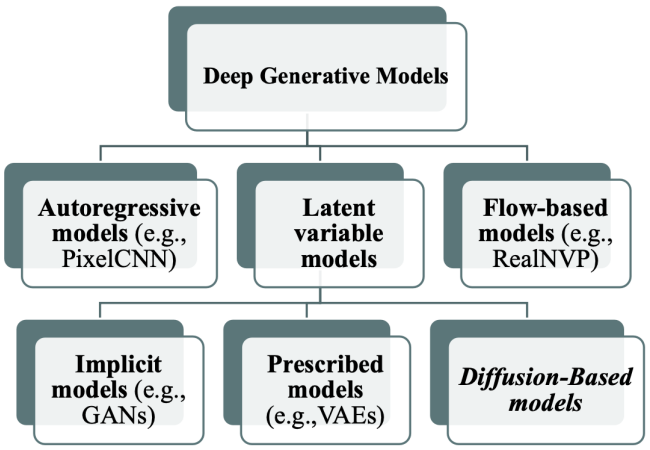


Figure 7: Generative Modeling types

2.2.1 Background

Autoregressive generative models : Modeling the distribution of natural images is a landmark problem in unsupervised learning. This task requires an image model that is at once expressive, tractable and scalable. Since images are high dimensional and highly structured, estimating the distribution of natural images is extremely challenging (78).

One effective approach to tractably model a joint distribution of the pixels in the image is to cast it as a product of conditional distributions; The factorization turns the joint modeling problem into a sequence problem, where one learns to predict the next pixel given all the previously generated pixels. The generation proceeds row by row and pixel by pixel (78).

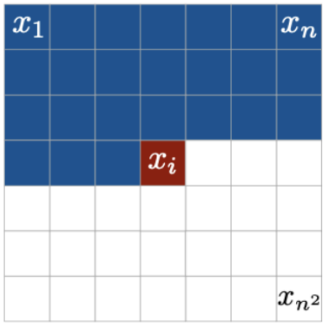


Figure 8: To generate pixel x_i one conditions on all the previously generated pixels left and above of x_i.

Architectures : The majority of research is focused on improving network architectures to increase their receptive fields and memory, ensuring the network has access to all parts of the input to encourage consistency, as well as increasing the network capacity, allowing more complex distributions to be modelled (3).

- **Recurrent Neural Networks:** A natural architecture to apply is that of standard recurrent neural networks (RNNs) such as LSTMs and GRUs which model sequential data by tracking information in a hidden state. However, RNNs are known to forget information, limiting their receptive field thus preventing modelling of long-range relationships. This can be improved by stacking RNNs that run at different frequencies allowing long data such as multiple seconds of audio to be modelled. Nevertheless, their sequential nature means that training can be too slow for many tasks.

- **Causal Convolutions:** An alternative approach is that of causal convolutions, which apply masked or shifted convolutions over a sequence. When stacked, this only provides a receptive field linear with depth, however, by dilating the convolutions to skip values with some step the receptive field can be orders of magnitude higher.

- **Self-Attention:** More recently self-attention (known as Transformers when used in an encoder-decoder setup) has made significant strides improving not only autoregressive models, but also other generative models due to its parallel nature, stable training, and ability to effectively learn long-distance dependencies. This is achieved using an attention scheme that can reference any previous input where an entirely independent process is used per time step so that there are no dependencies.

Standard autoregressive networks are popular for text/audio generation (3). While autoregressive models are extremely powerful density estimators, sampling is inherently a sequential process and can be exceedingly slow on high dimensional data. Additionally, data must be decomposed into a fixed ordering; while the choice of ordering can be clear for some modalities (e.g. text and audio), it is not obvious for others such as images and can affect performance depending on the network architecture used.

Additionally, there is no natural latent representation associated with autoregressive models. It means that the model doesn't inherently learn a structured, interpretable representation of the data (23).

Flow-based models: While training autoregressive models through maximum likelihood offers plenty of benefits including stable training, density estimation, and a useful validation metric, the slow sampling speed and poor scaling properties handicaps them significantly (Bond-Taylor, et al., 2022). Moreover, they lack a latent representation, therefore, it is not obvious how to manipulate their internal data representation that makes it less appealing for tasks like compression or metric learning (76)

Normalizing flows are a technique that also allows exact likelihood calculation while being efficiently parallelisable as well as offering a useful latent space for downstream tasks (3).

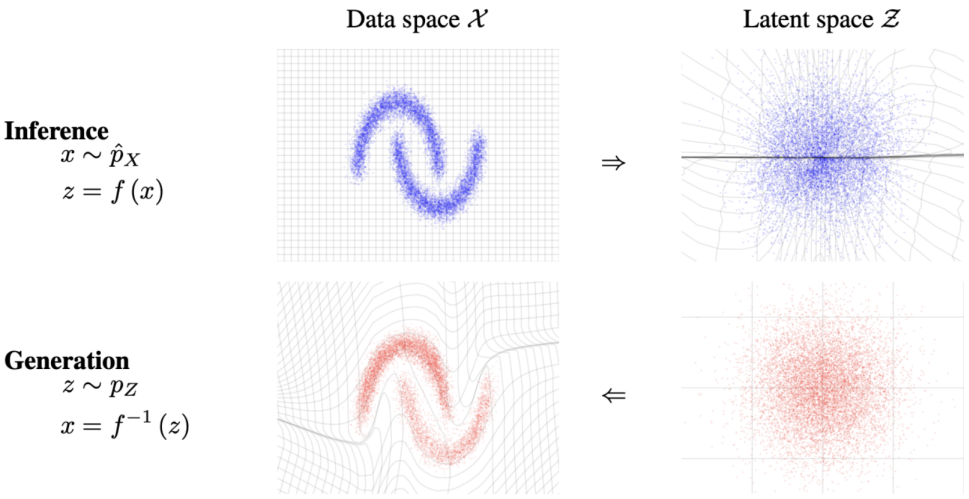


Figure 9: Example of Flow-based model

Flow-based models learn an invertible, stable, mapping between a data distribution

$\hat{p}_X$ and a latent distribution p_Z (typically a Gaussian). Here we show a mapping that has been learned on a toy 2-d dataset. As shown in Fig. 9, The function $f(x)$ maps samples x from the data distribution in the upper left into approximate samples z from the latent distribution, in the upper right. This corresponds to exact inference of the latent state given the data. The inverse function, $f^{-1}(z)$, maps samples z from the latent distribution in the lower right into approximate samples x from the data distribution in the lower left. This corresponds to exact generation of samples from the model (23).

So, Normalizing flow is a way of mapping a data's complex probability distribution $q(x)$ to a simple latent distribution $p(z)$ using a set of invertible, bijective and continuous functions $z = f(x)$ such that both f and f^{-1} are differentiable (62).

Although flow models are based on modeling the exact data distribution, they are often computationally expensive since they depend on the calculation of jacobians, have scalability and expressiveness issues on large and complex data distributions, and because of the invertibility constraint in calculations, require the input and output dimensions to be the same (62). Besides, normalizing flow models are typically less parameter efficient than other generative models (3).

Latent variable models :

Both flow-based models and autoregressive models, model the likelihood function directly, that is, either by factorizing the distribution and parameterizing conditional distributions or by utilizing invertible transformations (neural networks) for the change of variables formula as in flows (76).

Unlike flow-based models, they do not require the invertibility of neural networks and, thus, we can use any arbitrary architecture for encoders and decoders. In contrast to auto regressive models, they learn a low-dimensional data representation and we can control the bottleneck (i.e., the dimensionality of the latent space) (76).

Generative Adversarial Networks : Generative Adversarial Networks (GANs) are a class of deep learning models designed for generative tasks, such as image generation, style transfer, and data synthesis. GANs consist of two neural networks, the generator

and the discriminator, that are trained competitively.

GANs are a class of generative models that implicitly learns the probability distribution $q(x)$ of the dataset using an adversarial approach where two networks, the discriminator D and the Generator G play a two-player min-max game. The discriminator's objective is to maximize the binary classification of distinguishing real and generated images, whereas the generator's objective is to fool the discriminator into misclassifying the generated images (62).

The adversarial nature of GANs makes them notoriously difficult to train. Nash equilibrium is hard to achieve since non-cooperation cannot guarantee convergence (3). This results in a highly unstable training process as optimizing D can lead to the deterioration of G and vice-versa (62).

Mode collapse is another problem where the generator's objective function converges to a specific data distribution instead of the whole training set, thus only generating images belonging to this small subset (62). This issue on the other hand for discriminator leads to jump between modes resulting in catastrophic forgetting, where previously learned knowledge is forgotten when learning something new (3).

In addition, when the discriminator is trained to optimality, the gradients of D approach zero. This causes the problem of vanishing gradients for the generator, where it has no guidance into which direction to follow for achieving optimality (62).

To prune these problems, various modifications on the vanilla GAN were introduced, which either suggested architectural optimizations or loss function optimizations (38). As studied by (17) the loss function optimizations were divided into optimizing the discriminator's or generator's loss objective (62). These include minimizing the f-divergences along with the Jensen-Shannon divergence (57), weight normalization for stabilizing D's training (54), WGAN and WGAN-GP (1) which changed the objective of D from binary classification to a probability output by applying the Earth Mover (EM) or Wasserstein distance, EBGAN (62).

In Conditional GAN (cGAN), the generator and discriminator are conditioned on additional information, such as class labels or attributes. This enables controlled generation,

like generating images with specific attributes.

Variational Autoencoders : Before going to Variational Autoencoder, we will first discuss Auto Encoder. Auto Encoder is a self- supervised neural network that learns how to encode the input into lower dimensions, then decode and reconstruct the data again to be as close as the input as efficiently as possible Fig. 10.

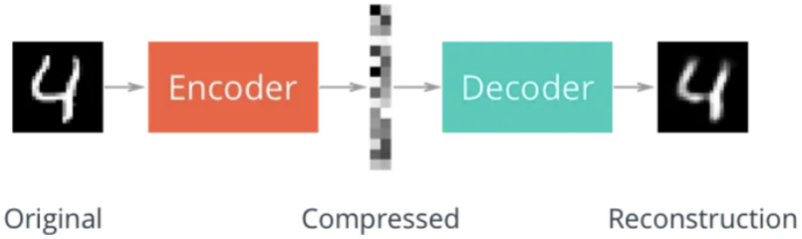


Figure 10: Example of Autoencoder model

An autoencoder consists of three parts:

- Encoder, the layers that encode the input data into a lower dimension representation.
- Compressed, the layer that contains the encoded/compressed representation and the lowest dimension. Also known as the Bottleneck.
- Decoder, the layers that learn to decode or reconstruct back the encoded representation into the data as close as the input data.

For Autoencoder to learn the best encoding and decoding, Autoencoder will aim to minimize the reconstruction error, which is basically the difference between the reconstructed data and the input.

Variational autoencoders transfer your input onto a distribution Fig. 11 (amortized variational Inference), and instead of translating it to a fixed vector, you feed a sample from that distribution to your decoder network. A variational auto-encoder's loss function consists of two terms: one for the reconstruction loss and the other for the KL divergence.

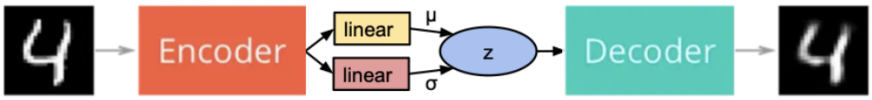


Figure 11: Example of Variational Autoencoder model

Variational Inference enables an efficient computation of a lower bound of likelihood for the observed data distribution. This lower bound is popularly referred to as the Evidence Lower BOund (ELBO).

$$\ln p(\mathbf{x}) = \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x}|\mathbf{z})] - KL[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})]}_{\text{ELBO}} + \underbrace{KL[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})]}_{\geq 0}. \tag{4}$$

If model takes too simplistic posterior, we can end up with a bad VAE anyway. By considering Fig. 12, If the ELBO is a loose lower bound of the log-likelihood, then the optimal solution of the ELBO could be completely different than the solution of the log-likelihood (76).

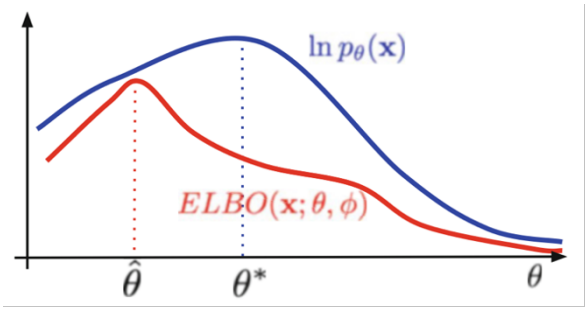


Figure 12: The ELBO is a lower bound on the log likelihood. As a result, $\hat{\theta}$ maximizing the ELBO does not necessarily coincide with θ^* that maximizes $\ln p(x)$. The looser the ELBO is, the more this can bias maximum likelihood estimates of the model parameters

VAEs constitute a very powerful class of models, mainly due to their flexibility. Unlike flow-based models, they do not require the invertibility of neural networks and, thus, we can use any arbitrary architecture for encoders and decoders. In contrast to ARMs, they learn a low-dimensional data representation and we can control the bottleneck (i.e., the dimensionality of the latent space). However, they also suffer from several issues. Except the ones mentioned before (i.e., a gap between the ELBO and the log-likelihood function for too simplistic variational posteriors), the potential problems are the following:

- posterior collapse: This causes the model to initially learn to ignore z and go after low hanging fruit, explaining the data with the more easily optimized decoder. Once this has happened, the decoder ignores the encoder and little to no gradient signal passes between the two, yielding an undesirable stable equilibrium with the KL cost term at zero (4).

- hole problem: imagine that we have the standard Gaussian prior and the aggregated posterior (i.e., an average of variational posteriors over all training data). As a result, there are regions where there prior assigns high probability but the aggregated posterior assigns low probability, or other way around. Then, sampling from these holes provides unrealistic latent values and the decoder produces images of very low quality (64).

Diffusion Models : The goal is to define a forward (or inference) diffusion process which converts any complex data distribution into a simple, tractable, distribution, and then learn a finite-time reversal of this diffusion process which defines the generative model distribution (70).

Denoising diffusion models represent a recent emerging topic in computer vision, demonstrating remarkable results in the area of generative modeling. A diffusion model is a deep generative model that is based on two stages, a forward diffusion stage and a reverse diffusion stage. In the forward diffusion stage, the input data is gradually perturbed over several steps by adding Gaussian noise. In the reverse stage, a model is tasked at recovering the original input data by learning to gradually reverse the diffusion process, step by step (18). Generally, Diffusion models are a class of probabilistic generative models that learn to reverse a process that gradually degrades the training data structure (18). The longer the trajectory the smaller the diffusion rate can be made.

There are three formulations of diffusion models, namely denoising diffusion probabilistic models, noise conditioned score networks, and the approach based on stochastic differential equations that generalizes over the first two methods. Each formulation obtains remarkable results in image generation, surpassing GANs while increasing the diversity of the generated samples (18).

The most significant disadvantage of diffusion models remains the need to perform multiple steps at inference time to generate only one sample. Despite the important amount of research conducted in this direction, GANs are still faster at producing images (18).

In image generation diffusion models can be categorized in two classes, Unconditional Image Generation and Conditional Image Generation.

- **Unconditional Image Generation**: Such models do not require supervision signals, being completely unsupervised. We consider this as the most basic and generic setting for image generation.
- **Conditional Image Generation**: Diffusion models could be applied to conditional image synthesis. The condition is commonly based on various source signals, in most cases some class labels being used. Some methods perform both unconditional and conditional generation.

2.2.2 State Of The Art

DALL·E : DALL·E (61), developed by OpenAI, is a groundbreaking neural network-based model designed for generating images from textual descriptions. It utilizes the Transformer architecture, which is well-known for its success in natural language processing tasks. Specifically, DALL·E combines a variational autoencoder (VAE) with autoregressive modeling. The model is trained on a diverse dataset of text-image pairs, enabling it to understand and generate a wide range of images based on textual prompts. DALL·E's architecture allows it to handle complex scenes and objects described in natural language, producing high-quality images that align closely with the given descriptions.

Architecture : The DALL·E architecture can be considered as a two-stage process involving a discrete variational autoencoder (dVAE) and a Transformer model. In the first stage, the dVAE is used to compress high-resolution images into a sequence of discrete latent codes. This involves encoding images into a lower-dimensional latent space while preserving crucial details that can be used for high-fidelity image reconstruction. The

dVAE helps in reducing the complexity of the image generation task by transforming the images into a manageable format for the subsequent text-to-image translation.

In the second stage, the Transformer model takes over, where it is trained to map textual descriptions to the sequences of latent codes produced by the dVAE. This involves processing the input text using the attention mechanism to understand and generate the sequence of latent codes that best represent the described image. The Transformer effectively learns the relationships and dependencies between the textual input and the visual features encoded by the dVAE. Once the latent codes are generated, they are decoded back into images using the decoder part of the dVAE, resulting in the final output image.

This two-stage process ensures that the model can handle the high dimensionality of image data while leveraging the powerful sequence modeling capabilities of Transformers to generate coherent and contextually accurate images from textual descriptions.

Imagen : Imagen (68), developed by Google Research, is a state-of-the-art text-to-image generation model that combines the strengths of large transformer-based language models and high-resolution image generation techniques. At its core, Imagen leverages a diffusion model for generating high-quality images from textual descriptions. The diffusion model gradually refines a noisy image into a clear, detailed picture, guided by the input text. This process allows Imagen to produce images with remarkable detail and coherence, capturing complex scenes and nuanced visual elements described in the text.

Architecture : Imagen consists of a text encoder that maps text to a sequence of embeddings and a cascade of conditional diffusion models that map these embeddings to images of increasing resolutions Fig. 14. Imagen revolves around the innovative use of diffusion models and transformer-based language models. At its core, Imagen employs a diffusion model to generate high-quality images from textual descriptions.

The text-to-image generation pipeline leverages a large pre-trained language model, such as T5 (Text-to-Text Transfer Transformer), to process and understand the input text comprehensively. The language model's representation of the text is then used to condition

the diffusion process, guiding each step of image refinement based on the semantic content of the description.

The diffusion model starts with a noisy image and iteratively refines it into a clear, detailed picture, guided by the input text. Classifier guidance (21) is a technique to improve sample quality while reducing diversity in conditional diffusion models using gradients from a pretrained model $p(c|z_t)$ during sampling. This process, which gradually denoises the image, allows for fine control over image generation, ensuring that the final output is both coherent and visually appealing. Classifier-free guidance (33) is an alternative technique that is used in Imagen. A method to increase sample quality while decreasing sample diversity in diffusion models.

Instead, Imagen involves training a single diffusion model on both conditional and unconditional objectives by randomly omitting the conditioning information during training (for example, with a 10% probability)

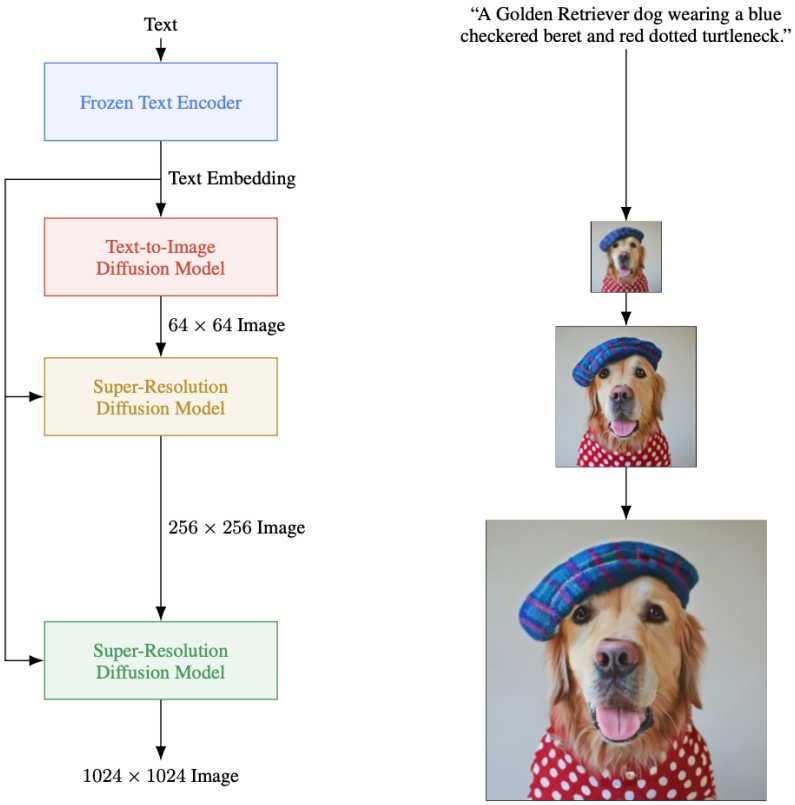


Figure 13: Visualization of Imagen. Imagen uses a frozen text encoder to encode the input text into text embeddings. A conditional diffusion model maps the text embedding into a 64 × 64 image. Imagen further utilizes text-conditional super-resolution diffusion models to upsample the image, first 64 × 64 → 256 × 256, and then 256 × 256 → 1024 × 1024.

Latent Diffusion Model : The Latent Diffusion Model (65) (LDM) is a notable advancement in the field of generative modeling, specifically for producing high-quality images from text descriptions. It leverages the principles of diffusion models, which iteratively refine a noisy image until a clear and detailed picture emerges, guided by the input text. The model starts with a noisy version of the desired image and progressively denoises it through a series of steps, each step informed by the textual input. This iterative process allows Stable Diffusion to generate images with remarkable fidelity and coherence, capturing intricate details and complex scenes described in natural language.

Architecture : The stable diffusion model integrates a two-stage process consisting of a Variational Autoencoder (VAE) component and a diffusion process, combining their strengths to achieve superior image generation. It worth noting that the VAE and diffusion process are trained separately.

In the first stage, the model maps the input image to a latent representation. In the second stage, a diffusion process is applied to this latent representation, classifier-free guidance (33) technique is used to increase sample quality while decreasing sample diversity in diffusion models. During training, noise is added to the latent representation to form a pure Gaussian distribution. A U-Net model is then responsible for identifying and removing this added noise from the latent representation. Finally, it reconstructs the image using the decoder part of the first stage.

During inference, the model begins by iteratively removing noise from an initially pure noise image. This process is conditioned on the input text, guiding the denoising steps to form a clean latent representation. Finally, the model reconstructs the image using the decoder component from the first stage, resulting in a high-quality image that aligns with the given textual input.

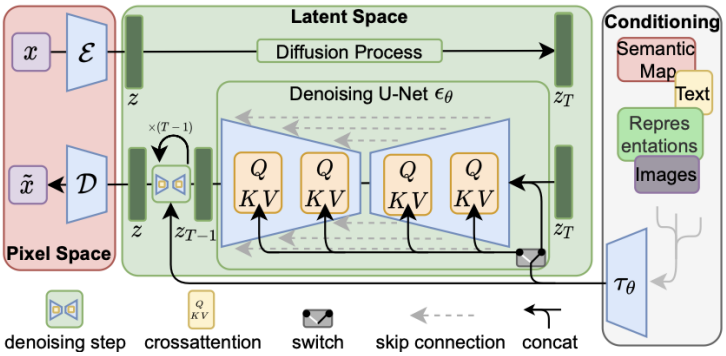


Figure 14: LDMs either via concatenation or by a more general cross-attention mechanism.

2.2.3 Why Generative Modeling

One significant reason for utilizing generative AI to create images and corresponding masks is the scarcity of available data needed to fine-tune models effectively. When the dataset is limited, training models can be challenging due to insufficient representation of various features and scenarios. Generative AI helps overcome this limitation by synthesizing new, high-quality data that can augment the existing dataset, ensuring the model has ample examples to learn from and generalize better.

Another critical reason is that annotating datasets manually is a time-consuming and labor-intensive process. Labeling images and creating precise masks require considerable effort and expertise, often leading to delays and increased costs. By employing generative AI, it is possible to automate the generation of annotated data, streamlining the process and significantly reducing the workload on human annotators. This efficiency enables faster model development and deployment, as well as more scalable and cost-effective data annotation workflows.

3 Method

This section will provide a more detailed explanation of the segmentation model, the synthetic dataset generation process and dataset preparation steps. The whole approach is as following, first we try to generate sunthetic images and coresponding masks. Subsequently we use the synthetic dataset to fine-tune the SEEM model as shown in the Fig.

16.

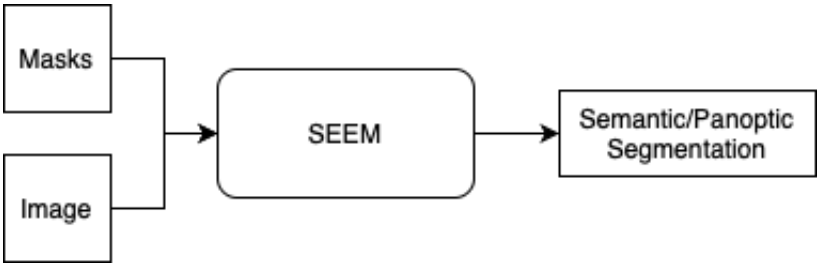


Figure 15: Schematic of pipeline of project trained on real dataset.

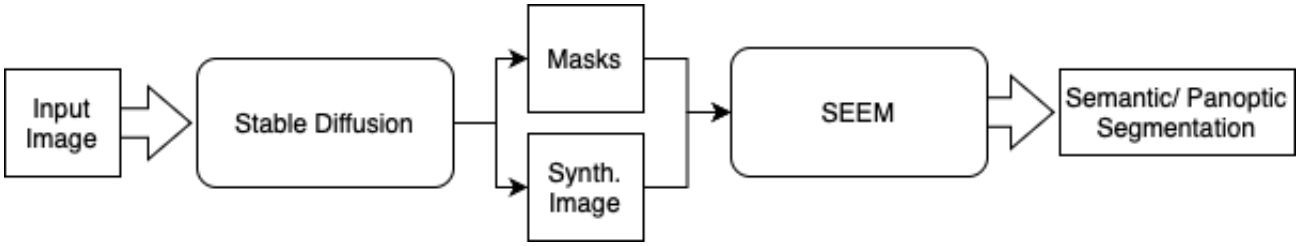


Figure 16: Schematic of pipeline of project trained on symthetic dataset.

3.1 Image Segmentation

Transfer learning leverages the knowledge learned from large-scale natural image datasets to solve specific problems in the desired domain. Recent advances in large foundational language models such as Chat-GPT and GPT-4 have shown impressive performances in various language tasks (60). With their superior transferability to different tasks, many have explored transferring foundational language models to medical tasks ((36), (32)).

Segment Everything Everywhere All At Once Model (90) is proposed as a foundation model for image segmentation that generates high-quality object masks from input prompts (e.g., points, boxes, masks and text) [SEEM]. Due to its promising performance in several computer vision benchmarks, SEEM has attracted much attention. In the following sections, by means of transfer learning, the SEEM is fine-tuned on custom dataset. SEEM is fine-tuned on a collection of wet damage images.

3.1.1 Model Design

Overview : In Segment Everything Everywhere All At Once Model (SEEM), they strive for a universal interface for segmenting everything everywhere all at once in an image. On this interface, they are targeted at unifying all segmentation tasks with a single model in a promptable manner. To achieve this goal, we propose a new prompting scheme in mask decoder that has four important properties: versatility, compositionality, interactivity, and semantic-awareness. Specifically, they propose to encode points, masks, text, boxes, and even a referred region from another image into prompts in the same joint visual-semantic space (90).

By training on diverse segmentation tasks, SEEM learns to deal with various prompts, align the visual and text prompts, and promote their synergy via cross-attention between them. As a result, SEEM after pretraining attains competitive performance across all segmentation tasks.

Since the prompts of all 5 different types are mapped to the joint visual-semantic space, we can feasibly combine prompts to resolve the ambiguity to obtain better segmentation results and enable zero-shot adaptation to unseen user prompts.

SEEM employs a generic encoder-decoder architecture but also employs a sophisticated interaction scheme between queries and prompts. Then, SEEM-Decoder predicts the masks M and semantic concepts C based on the query outputs O_m(mask embeddings) and O_c (class embeddings), which interact with text, visual, and memory prompts (90).

In the upcoming section, we will delve into each component of the SEEM model in greater detail, aiming to enhance comprehension of its individual parts and, ultimately, the entire model pipeline.

Language Encoder For the language encoder, SEEM adopts a Unified Contrastive Learning. (85) propose a new learning paradigm, called Unified Contrastive Learning (UniCL) with a single learning objective to seamlessly prompt the synergy of two data types. Extensive experiments show that UniCL is an effective way of learning semantically rich yet discriminative representations, universally for image recognition in zero-shot,

linear-probing, fully finetuning and transfer learning scenarios.

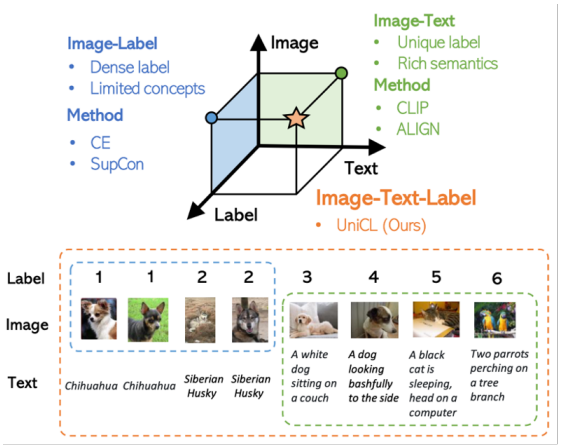


Figure 17: Unified contrastive learning paradigm in the image-text-label space

As illustrated in Fig. 2. Instead of isolating image-label and image-text data, we define an image-text-label space and show how we can eliminate the boundary between two data types. As shown in Fig. 2 left part, supervised learning (41) on image-label data typically aims at mapping images to discrete labels, and completely ignores the textual concept associated with each label during the training.

In contrast, language-image contrastive learning (60) aims at learning a pair of visual and textual encoders to align images and texts as shown in Fig. 2 right part. This learning method implicitly assumes that each image-text pair has a unique label.

Comparing these two learning paradigms side by side, we can see that both of them actually reside in the common image-text-label space, which is constructed by mapping each label to a textual concept for supervised learning, and assigning each textual description a unique label for language-image pretraining, as shown in Fig. 2 bottom.

UniCL method combines image-label and image-text data together to learn discriminative and semantic-rich representations, which are beneficial to a variety of downstream tasks.

UniCL used CLIP (60) as its tokenizer to tokenize the input text to the model, but it worth noting that, the language encoder is kept frozen during both training the original SEEM and fine-tuned version.

During the training or validation phase, we pass the text related to the input image to the language encoder, afterwards the input text is tokenized by means of CLIP tokenizer. Next, the tokenized text is feed to the language encoder to obtain the embeddings of the text of related input image.

It worth nothing that during the training the model, every time that we want to calculate the representative embedding of the input label, a neutral template randomly is selected and filled with the class label, for example, we could have:

- An image of "class label"

There is also possibility to just use class label instead of the template text.

If we assume the output of the vision feature(embeddings) as $\hat{v}$ and the output of the language encoder as $\hat{u}$ we can calculate the similarity of the vision features and language embeddings as follow:

$$u_i = \frac{\hat{u_i}}{\|\hat{u_i}\|} \text{ and } v_i = \frac{\hat{v_i}}{\|\hat{v_i}\|} \text{ , then } s_{i,j} = \tau * u_i^T.v_i \tag{5}$$

τ here is called temperature, In this way we can calculate the predictions for each class (labels).

Vision Backbone SEEM uses few vision backbones as for its vision part of pipeline. In the paper they used the results of Da-ViT-d3, but the weights of this model have not been published yet, thus, instead we used Focal-L as backbone which has better results in comparison with other models.

Focal Modulation Networks are an alternative to Vision Transformers, where self-attention is completely replaced by a focal modulation mechanism for modeling token interactions in vision to build an attention-free architecture.

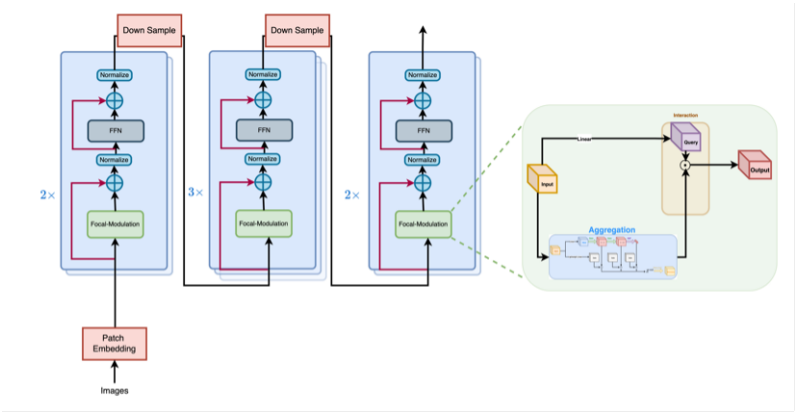


Figure 18: FocalNets Architecture

Focal modulation (84) comprises three components:

- focal contextualization implemented using a stack of depth-wise convolutional layers, to encode visual contexts from short to long ranges.
- gated aggregation to selectively gather contexts into a modulator for each query token
- element-wise affine transformation to inject the modulator into the query.

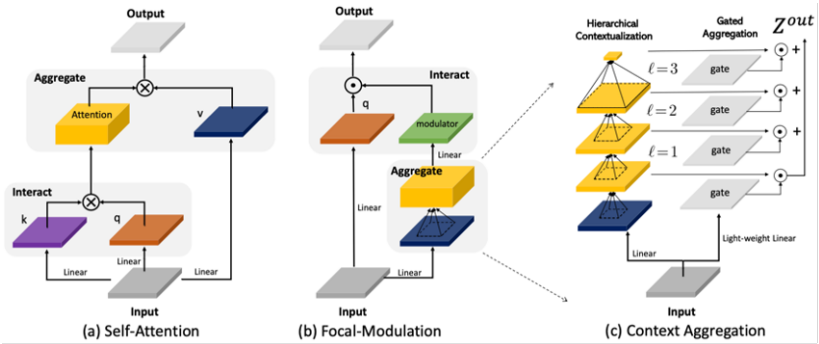


Figure 19: Left: Comparing SA (a) and Focal Modulation (b) side by side. Right: Detailed illustration of context aggregation in Focal Modulation (c).

Fig. 4 depicts the difference between the self-attention and the focal modulation blocks.

Feature Pyramid Networks Feature pyramids are a basic component in recognition systems for detecting objects at different scales. Pyramid networks are scale-invariant

in the sense that an object's scale change is offset by shifting its level in the pyramid. Intuitively, this property enables a model to detect objects across a large range of scales by scanning the model over both positions and pyramid levels.

Aside from capability of ConvNets of representing higher-level semantics, ConvNets are also more robust to variance in scale and thus facilitate recognition from features computed on a single input scale. But even with this robustness, pyramids are still needed to get the most accurate results (84).

The use of multiple feature levels creates a pyramid-like structure, where higher-level features correspond to lower spatial resolutions and lower-level features correspond to higher spatial resolutions. This is advantageous for detecting objects of different scales.

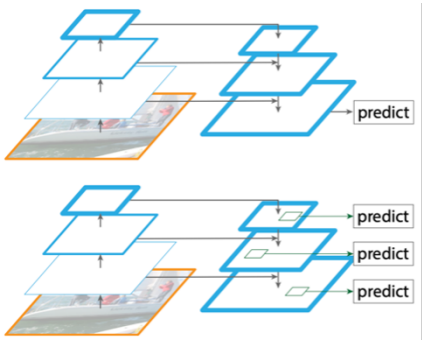


Figure 20: Feature Pyramid Network

In the SEEM vision backbone consists of four different levels, in each level spatial dimensions are halved, and the number of channels doubled except for very last layer. The output of vision backbone then is passed to a network called Pixel Decoder to aggregate features of different levels in order to obtain a feature mask which has a good semantic knowledge and high spatial dimension and align the number of channels for segmentation head.

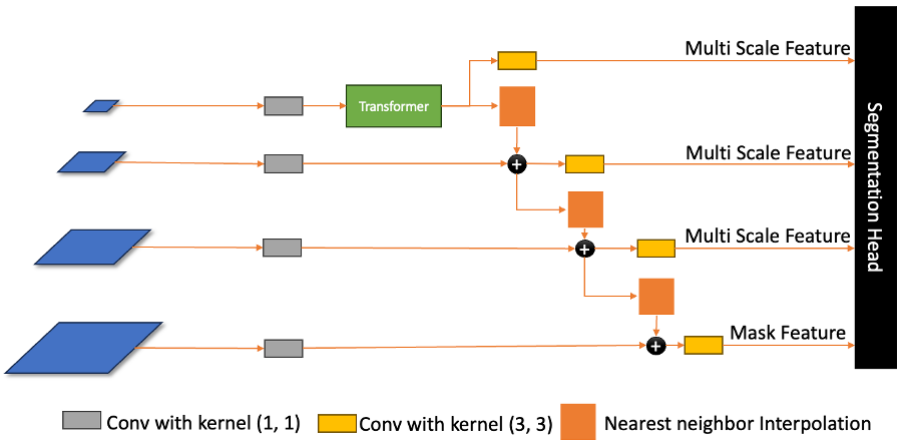


Figure 21: Pixel Decoder

As depicted in Fig. 6, first the features of are passed to a convolution layer with kernel (1,1) to change the channels number to allow us to add different resolutions. Afterwards, we need to also interpolate the spatial dimension to be able to add different resolutions. Finally, each feature map is passed to convolution layer with kernel (1,1) and all feature maps are passed to segmentation head.

Segmentation Head To handle small objects, (14) propose an efficient multi-scale strategy to utilize high-resolution features. It feeds successive feature maps from the pixel decoder's feature pyramid into successive Transformer decoder layers in a round robin fashion. Finally, we incorporate optimization improvements that boost model performance without introducing additional computation.

Masked Attention Context features have been shown to be important for image segmentation ((9), (10), (87)). However, recent studies (25), (72) suggest that the slow convergence of Transformer-based models is due to global context in the cross-attention layer, as it takes many training epochs for cross-attention to learn to attend to localized object regions (72). mask2former paper hypothesize that local features are enough to update query features and context information can be gathered through self-attention. For this they propose masked attention, a variant of cross-attention that only attends within the foreground region of the predicted mask for each query.

High-resolution features High-resolution features improve model performance, especially for small objects (8). However, this is computationally demanding. Thus, (14) propose an efficient multi-scale strategy to introduce high-resolution features while controlling the increase in computation. Instead of always using the high-resolution feature map, they utilize a feature pyramid which consists of both low- and high-resolution features and feed one resolution of the multi-scale feature to one Transformer decoder layer at a time.

Specifically, the feature pyramid produced by the pixel decoder with resolution 1/32, 1/16 and 1/8 of the original image is used. For each resolution, , following (8) a sinusoidal positional embedding $e_{pos} \in \mathbb{R}^{H_i W_i * C}$ is added, and a learnable scale-level embedding $e_{lvl} \in R^{1*C}$, following (88). We use those, from lowest-resolution to highest-resolution for the corresponding Transformer decoder layer as shown in Figure 7. They repeat this 3-layer Transformer decoder L times. The final Transformer decoder hence has 3L layers. More specifically, the first three layers receive a feature map of resolution H1 = H/32, H2 =H/16,H3 =H/8 and W1 =W/32,W2 =W/16, W3 = W/8, where H and W are the original image resolution. This pattern is repeated in a round robin fashion for all following layers.

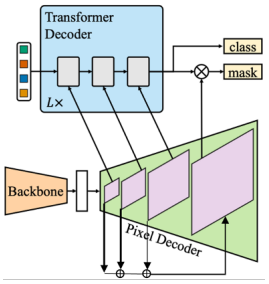


Figure 22: mask, class prediction pipeline

Segmentation Head Segmentation head is Transformer decoder (figure 8) with masked attention adopted from (14) The key components of Transformer decoder include a masked attention operator, which extracts localized features by constraining cross-attention to within the foreground region of the predicted mask for each query, instead of attending to the full feature map.

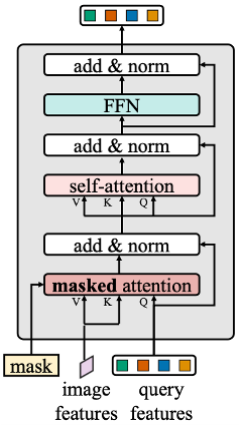


Figure 23: Segmentation Head

3.1.2 Pipeline

After loading the model with pretrained weights, first of all, we need to register our dataset, by means of register module. After creating the instance of default trainer, and initializing it, the train function is called to begin the train process. Afterwards, the input batch which contains the input image(s) and corresponding meta data related to grounding task and other task are loaded. The Vision Backbone is responsible to provide the vision embedding which are a feature pyramid network for in four different resolutions.

Afterwards, as depicted in Feature Pyramid Networks, features of different resolutions are added together by upsampling to obtain a feature that are semantically reach and also spatially detailed, this feature map is called mask, which will be used in the segmentation head alongside with features of different resolutions.

Next, the embedding for each category is generated by inputting a sentence containing the class name into the Language Encoder. These embeddings are then used to determine the similarity between vision and language tokens.

Finally, the Segmentation Head performs segmentation task based on input image and corresponding text.

3.1.3 Fine-tuning, Methods and Strategies

Linear probing is a straightforward approach to maintaining the pre-trained model fixed by only tuning a specific lightweight classification head for every task. However, linear probing tends to have an unsatisfactory performance and misses the opportunity of pursuing strong but non-linear features, which indeed benefit deep learning (11).

Recently, transformers have witnessed remarkable success in a broad range of computer vision fields. Benefiting from the dynamic modeling capability and the long-range dependence of the attention mechanism, various vision transformers soon rose in many computer vision tasks such as object detection and semantic segmentation, surpassing CNN models and reaching state-of-the-art performance (13).

We needed to incorporate knowledge about task domain images into SEEM. Therefor we aimed to fine-tune either vision backbone or segmentation head of SEEM model or both simultaneously. In this context, our approach involved preserving the parameters of the entire model while incorporating task-specific domain knowledge by means of adapters, ensuring that the original knowledge remained intact and unaltered.

Adapters have been proven to be an effective strategy for fine-tuning large-scale models (82), (13). This enables an existing model to excel in both new and the original tasks. Due to the parameter-sharing property of adapters, they facilitate transfer learning between different tasks (16).

Methods Fine-tuning large pre-trained models is an effective transfer mechanism in computer vision and NLP. However, in the presence of many downstream tasks, fine-tuning whole model is parameter inefficient, an entire new model is required for every task. As an alternative, transfer learning could be done with adapter modules. Adapter modules yield a compact and extensible model; they add only a few trainable parameters per task, and new tasks can be added without revisiting previous ones (35).

During the fine-tuning phase, we only choose the newly added parameters to optimize and keep rest ones fixed. Specifically, the original model parts, load weights from the pre-trained checkpoint and keeps parameters frozen. The newly added parameters are

updated on the specific data domain with the task-specific losses.

To demonstrate adapter's efectiveness, (35) transfer the recently proposed BERT Transformer model to 26 diverse text classification tasks, including the GLUE benchmark. Adapters attain near state-of-the-art performance, whilst adding only a few parameters per task. On GLUE, they attain within 0.4% of the performance of full fine-tuning, adding only 3.6% parameters per task. By contrast, fine-tuning trains 100% of the parameters per task.

Strategies

Freeze vision backbone, adapter fine-tuning for segmentation head Adaptors are used mainly in two parts of SEEM's segmentation head. First, we inspired our work based on (73) to fine-tune decoder. Secondly, to fine-tune prediction head of decoder, method used by (11) is incorporated.

- VL-ADAPTER: Fig. 9(b) left. Adapters (35) are sub-networks with small parameters that are inserted after every attention and feed-forward layer in a model. With adapters, the models learn downstream tasks by updating only a small number of parameters. The adapters consist of a pair of downsampling and upsampling layers, and a residual connection. To be more specific, (73) denote the input of the adapter as $x \in \mathbb{R}^{d_i}$ and the weight matrices for downsampling and upsampling layers to be $\theta^D \in \mathbb{R}^{d_i * d}$ and $\theta^U \in \mathbb{R}^{d * d_i}$ where d_i and d are the input and hidden dimensions, respectively. The mechanism of adapters is defined as:

$$h = f_{\theta^U}(\sigma(f_{\theta^D}(x)) + x \tag{6}$$

  Where $\sigma(\cdot)$ is an activation function, and GELU is used in the paper. With adapters, the parameter complexity (i.e., the number of added parameters) is $O(d_i, d)$, and it usually is $2 \sim 3\%$ of the whole model's parameters. Based on paper all layer normalization layers of the decoder are also updated to be adapted to the data distribution of downstream data.

As Fig. 9 depicts, VL-Adapters are used in each layer of SEEM decoder as shown in the figure.

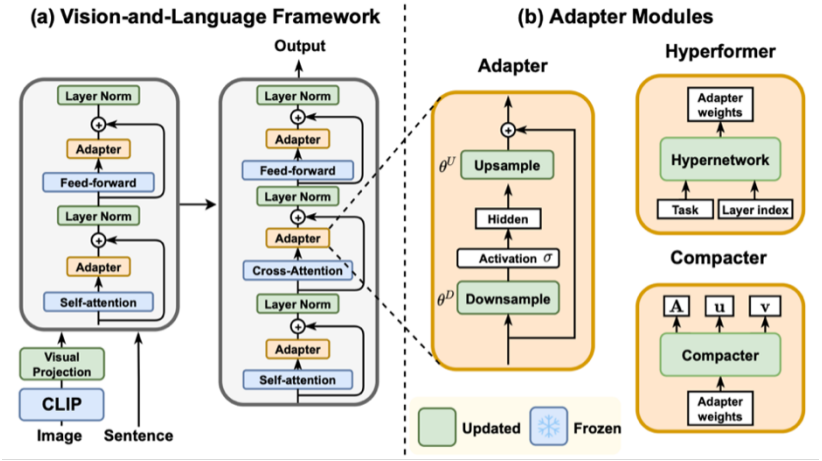


Figure 24: VL-ADAPTER: Parameter-Efficient Transfer Learning for Vision-and-Language Tasks

- AdaptFormer: (11) propose a lightweight module, namely AdaptFormer, to adapt vision transformers by updating the weights of Adapt-Former. (11) introduce learnable parameters from the model perspective, which is different from(39) proposed Visual Prompt Tuning VPT, which inserts learnable parameters into the token space. AdaptFormer is conceptually simple yet effective. It consists of two fully connected layers, a non-linear activation function, and a scaling factor.

  Specifically, the right branch of figure 3 is designed to be a bottleneck structure for limiting the number of parameters purpose, which includes a down-projection layer with parameters $W_{down} \in \mathbb{R}^{d*\hat{d}}$ an up-projection layer with parameters $W_{up} \in \mathbb{R}^{\hat{d}*d}$, where $\hat{d}$ is the bottleneck middle dimension and satisfies $\hat{d} \ll d$, there is a ReLU layer between these projection layers for non-linear property. This bottleneck module is connected to the original MLP network (left branch) through the residual connection via a scale factor s.

  For a specific input feature $x_l^{'}$, the right branch in AdaptMLP produces the adapted features, $\hat{x}_l$, formally via:

$$\hat{x}_l = ReLU(LN(x_l^{'}) \cdot W_{down}) \cdot W_{up} \tag{7}$$

  Then both the features $\hat{x}_l$ and $x_l^{'}$ are fused with x_l by residual connection,

$$x_l = MLP(LN(x_l^{'})) + s \cdot \hat{x}_l + x_l^{'} \tag{8}$$

As Fig. 10 demonstrates, this module is set in parallel to the feed-forward network (FFN) in the prediction head of SEEM decoder. Please consider that I the implementation of SEEM, prediction head is a different module respect to SEEM decoder transformer layer. Fig. 10 is used solely to illustrate how the AdaptFormer module is utilized.

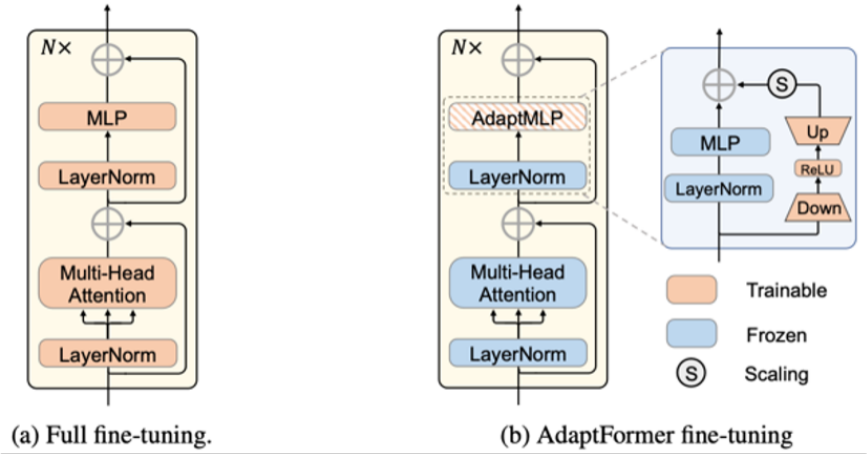


Figure 25: Comparison of previous full and AdaptFormer fine-tuning.

This design is turned out to be effective for model transfer when processing scalable visual tokens for both image and video data (i.e., image data consists of a small scale of visual tokens while video data consists of a large scale) (11)

Adapter fine-tuning for backbone, adapter fine-tuning for segmentation head As the most parameter-heavy part of SEEM, globally updating the image encoder during fine-tuning incurs significant computational costs and might leads to loss of knowledge. To this end, we also decided to fine-tune the vision backbone with adapters as well, alongside with fine-tuning the segmentation head with adapters. Since the vision backbone is a feature pyramid network, we incorporated adapters in each level of vision backbone within each block for focal modulation block.

The adapter which is used in the vision backbone is different from those used in the segmentation head, the following sections is dedicated to this fine-tuning technique and the way that is implemented.

To incorporate new task knowledge into the image encoder at a lower cost, (16) introduce new Adapter technology. Specifically, in this method we freeze all parameters of the original image encoder during fine-tuning and deploy an adapter for each focal modulation block, as shown in Fig. 11.

- (16) adapt the image encoder along both channel and spatial dimensions. For the channel dimension, they first compress the resolution of the input feature map to C×1×1 using global average pooling. Then, a linear layer is used to compress the channel embeddings and another linear layer to restore them, with a compression ratio of 0.25. Finally, we obtain the weights of the channel dimension through a sigmoid function and multiply them with the input feature map as the input for the next level. For the spatial dimension, the spatial resolution of the feature map downsampled by a factor of two using a convolutional layer and restore the spatial resolution using a transposed convolution, maintaining the same number of channels as the input.

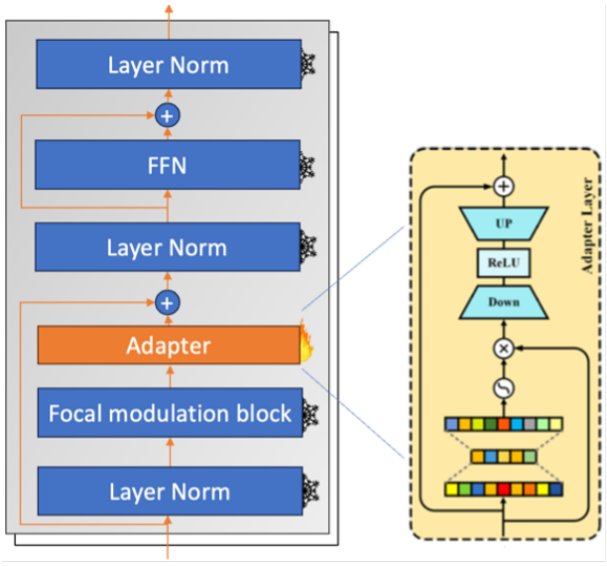


Figure 26: A layer of FocalNets, the position of adapter used for fine tuning.

Freeze vision backbone, fine-tune whole segmentation head In this approach, we maintain the original SEEM model unchanged. However, unlike previous strategies, we do not employ adapters. Instead, we allow all weights of the original SEEM's segmentation head to be updated during training, while keeping the vision backbone weights frozen.

Adapter fine-tuning for vision backbone, fine-tune whole segmentation head In this strategy we try to fine-tune both vision backbone and the segmentation head, but in different way with respect to Adapter fine-tuning for backbone, adapter fine-tuning for segmentation head. In way that we let the segmentations head to be intact, but on the other hand, we try to adapt the vision backbone in way that to let it lean new task data distribution and also preserve the original knowledge from pretraining.

Through this comparative study, we can gain a better understanding of which strategy performs more effectively than others for any new task.

3.2 Generating Synthetic Dataset

Preparing training data for deep vision models is a labor-intensive task. To address this, generative models have emerged as an effective solution for generating synthetic data. While current generative models produce image-level category labels, (55) proposes a novel method for generating pixel-level semantic segmentation labels using the text-to-image generative model Stable Diffusion (SD). By utilizing the text prompts, cross-attention, and self-attention of SD.

Text-to-image diffusion models have the remarkable ability to generate high-quality images with diverse open-vocabulary language descriptions. This demonstrates that their internal representation space is highly correlated with open concepts in the real world. Text-image discriminative models like CLIP, on the other hand, are good at classifying images into open-vocabulary labels. We leverage the frozen internal representations of both these models to perform unsupervised object discovery: saliency segmentation and object localization.

3.2.1 Method

Our objective is to generate a synthetic dataset $D = (I_i, S_i)_{i=1}^{N}$, consisting of high-fidelity images I and pixel-level semantic masks S. These images and masks capture both the semantic and location information of the target classes $C = \{c_1, c_2, ..., c_K\}$, where K represents the number of classes. The purpose of constructing this dataset is to train a

semantic segmenter ϕ without relying on human annotation.

Following (55), we perform a three-step process:

- Firstly, we prepare relevant text prompts P containing the target classes
- Secondly, using Stable Diffusion (SD) as our model, we generate images $I_i \in \mathbb{R}^{H\times W\times 3}$ and their corresponding semantic segmentations
- Lastly, we train a semantic segmenter ϕ on D and evaluate its performance on the test set of standard semantic segmentation datasets 4.4.

Generating Segmentation from Self and Cross-attention Maps We build our segmentation generator on Stable Diffusion (SD) by leveraging its self and cross-attention layers. Given a text prompt P first encoded by a text encoder into text embedding $e \in \mathbb{R}^{\Lambda\times d_e}$ with the text length λ and the number of dimensions d_e, SD seeks to output the final latent state $z_0 \in \mathbb{R}^{H\times W\times dz}$, where H, W, d_z are height, width, and number of channels of z_0, reflecting the content encoded in e from the initial latent state $z_T \sim \mathcal{N}(0, I)$ after T denoising steps.

At each denoising step t, a UNet architecture with L layers of self and cross-attention is used to transform z_t to z_{t-1}. In particular, at layer l and time step t, the self-attention layer captures the pairwise similarity between positions within a latent state z_t^l in order to enhance the local feature with the global context in z_t^{l+1}. In the meantime, the cross-attention layer models the relationship t between each position of the latent state z_t^l and each token of the text embedding e so that z_t^{l+1} can express more of the content encoded in e.

Formally,the self-attention map $A_S^{l,t} \in [0,1]^{HW\times HW}$ and cross-attention map $A_C^{l,t} \in [0,1]^{HW\times\lambda}$ at layer l and time step t are computed as follows:

$$A_S^{l,t} = Softmax(\frac{Q_z K_z^T}{\sqrt{d_l}}), \qquad A_C^{l,t} = Softmax(\frac{Q_z K_e^T}{\sqrt{d_l}}), \tag{9}$$

where Q_z,K_z,K_e are the query of z, key of z, and key of e, respectively, obtained by

linear projections and taken as inputs to the attention mechanisms, and d_l is number of features at layer l.

Since we only want to obtain the cross-attention map of the class labels C_i of image i for semantic segmentation, we are intrested in taking the softmax over the class name part C_i rather than entire of the text prompt P. After this, we obtain $A_C^{l,t} \in [0,1]^{HW \times M}$, where M is the number of classes in the image.

(55) observed that using different ranges of time-steps only affects the final result marginally. Finally, by taking average over these cross and self attention maps over layers and time-steps:

$$A_S = \frac{1}{L \times T} \sum_{l=1}^{L} \sum_{t=o}^{T} A_S^{l,t}, \qquad A_C = \frac{1}{L \times T} \sum_{l=1}^{L} \sum_{t=o}^{T} A_C^{l,t} \tag{10}$$

Although the cross-attention maps A_C already exhibit the location of the target classes in the image, they are still coarse-grained and noisy as depicted in Fig. B2. Thus, (55) proposes to use the self-attention map A_S to enhance A_C for a more precise object location.

This is because the self-attention maps capturing the pairwise correlations among positions within the latent z_t can help propagate the initial cross-attention maps to the highly similar positions, e.g., non-salient parts of the object, thereby enhancing their quality. Therefore, (55) proposes self-attention exponentiation where the self-attention map A_S is powered to τ before multiplying to the cross-attention map A_C as:

$$A_C^* = (A_S)^\tau \cdot A_C, \qquad A_C^* \in [0,1]^{HW \times M} \tag{11}$$

Finally, by selecting a threshold we select the parts of A_C^* which are greater that threshold to be selected as mask of given classes.

Generating Task-Aligned Images In the text-to-image task, the model begins with pure noise and incrementally refines the latent variable using a guided prompt. We observed that, for some classes, the generated images do not align with the task domain

Fig. 27. To tackle this issue, we experimented with the image-to-image version of the model. This approach allows us to use images aligned with the task domain. By leveraging the encoder from the first stage, we map the image i to its latent space z_ϕ^i. Subsequently, using the given prompt for the image i and sampling from the latent space z_ϕ^i, we can generate images that align with the task domain.

Figure 27: example of generated image for class 'light mold' that is not aligned with task domain.

Figure 28: examples of generated image for class 'light mold' that is aligned with task domain.

3.3 Dataset Preparation

To effectively train the SEEM model, it is necessary to create a dataset containing annotated ground truth images for panoptic learning. Additionally, to train the model for the grounding task, it is imperative to generate grounding annotations file. The subsequent sections will outline the process of creating each mentioned file and defining its structure.

3.3.1 Dataset Format

Panoptic Ground Truth Image Generation For panoptic learning preparation, two types of files are required: panoptic images and panoptic_annotations.json file. Panoptic images are images that are annotated at pixel level, in this way whole image scene is annotated and feed to model.

After obtaining annotations of original image which are segmentation mask in form of polygons and bounding box related to each category shown in the original image. In order to create panoptic ground truth, it's essential to assign a distinct color to each category. Subsequently, utilizing the panoptic-api, we can convert each color into a unique identifier and color, particularly if there are multiple instances of a category within an image.

For example, in a scene comprising a segmentation mask for the sky and two persons, where the sky category is classified as a 'stuff', all pixels within the segmentation mask will share the same color previously designated. Conversely, since the person class is defined as a 'thing', each person's segmentation mask will exhibit a different color, one assigned according to a predefined color and the other derived from the predefined color for the person category.

Panoptic Ground Truth Annotation File In order to generate a panoptic ground truth annotations file, it is necessary to possess a dictionary containing three primary keys: 'images', 'annotations' and 'categories'. Within this structure, the 'images' field should comprise the following essential attributes at a minimum:

- {"file_name": string, "height": int, "width": int, "id": int}.

The 'annotations' field consists of a list of objects corresponding to each image. The following attributes are essential for the 'annotations' field:

- {"segments_info": [], "file_name": string, "image_id": int}

Each 'segments_info' has following information about each segment in the image:

- {"id": int(generated by panopticapi), "category_id": int, "iscrowd": 0, "bbox": [float, float, float, float], "area": float}

The categories must have at least followed characteristics:

- {"isthing": 0/1, "id": int, "name": string}

Semantic Segmentation Ground Truth Image Generation To acquire semantic segmentation ground truth images, we can utilize the panoptic-api. Through this API, semantic segmentation ground truth files can be generated from existing panoptic ground truth images.

It's important to note that our category IDs might not be contiguous or may not start from zero. Consequently, IDs of categories must start from zero and be contiguous, covering the range [0, classes). This adjustment is necessary because when the model applies Soft-Max to the calculated scores, the outputs should fall within the range of zero to the number of classes. If we don't consider this fact, during the evaluation we will obtain miss-calculated metrics.

Grounding Annotations In order to generate a grounding annotation file, it is necessary to include segmentation details and one or more sentences associated with the segmentation. As we have broadened the content of each image scene to encompass various classes beyond the primary ones of interest(stuff categories).

Using sentences, we can establish connections between each segmentation and its position or content. For example, for the provided image, descriptive grounding texts could be formulated as follows:

“the bird on the right hand side”, “the seagull on the right”, “standing on lower portion of rock” and “bird with shorter neck”.

Figure 29: Grounding task example

In order to generate a grounding annotations file, it is necessary to possess a dictionary containing two primary keys: ‘images’, ‘annotations’ . Within this structure, the ‘images’ field should comprise the following essential attributes at a minimum:

- {”file_name”: string, ”height”: int, ”width”: int, ”id”: int}

The ’annotations’ field consists of a list of objects corresponding to each segmentation. The following attributes are essential for the ’annotations’ field:

- {”segmentation”: [polygon], ”area”: float, ”iscrowd”: 0, ”image_id”: int, ”category_id”: int, ”bbox”: [float, float, float, float], ”sentences”: []}

For each segmentation there is a field called ‘sentences’ which is list of sentences for that segmentation, ‘sentences’ has following format:

- {”tokens”: [], ”raw”: ”the sentence”, ”sent_id”: 0, ”sent”: ” the sentence”}

During the training process, a sentence will be selected at random. Next, utilizing a language backbone, the sentence will be embedded. This embedding will then be merged with image features supplied by a vision backbone to enhance the learning procedure.

3.3.2 Prepare Dataset for Train

SEEM dataloader follows Detectron2 framework that contains a dataset register and a dataset mapper. Accessing data from a dataset can be done through the DatasetCatalog, while metadata such as class names can be retrieved from the MetadataCatalog.

Dataset Catalog A global dictionary that stores information about the datasets and how to obtain them. It contains a mapping from strings (which are names that identify a dataset, e.g. "coco_2014_train") to a function which parses the dataset and returns the samples in the format of list[dict]. The returned dicts should be in Detectron2 Dataset format.

The purpose of having this catalog is to make it easy to choose different datasets, by just using the strings in the config.

Metadata Catalog MetadataCatalog is a global dictionary that provides access to MetaData of a given dataset. The metadata associated with a certain name is a singleton: once created, the metadata will stay alive and will be returned by future calls to 'get(name)'.

It's like global variables, so don't abuse it. It's meant for storing knowledge that's constant and shared across the execution of the program, e.g.: the class names in COCO.

Registration Detectron2 dataset registration involves the process of integrating custom datasets into the Detectron2 framework for use in training and evaluation of object detection models. Before registering a dataset, it needs to be properly organized. This includes structuring the dataset into a format compatible with Detectron2, typically consisting of images and corresponding annotations (e.g., bounding boxes, masks). Annotations must be in a supported format, such as COCO JSON.

In order to register a dataset, DatasetCatalog.register(name, func) must be called.

- name (str) – the name that identifies a dataset, e.g. "coco_2014_train".

- func (callable) – a callable which takes no arguments and returns a list of dicts. It must return the same results if called multiple times (20)

Mapping A callable which takes a sample (dict) from dataset and returns the format to be consumed by the model. DatasetMapper class is used to map each element of dataset. It loads images, transforms images and annotations, and converts annotations to an 'Instances' object.

4 Results

4.1 Dataset

To determine the most effective fine-tuning strategy for the SEEM model, four different datasets were chosen. '$Cityscape + PlantVillage$' dataset contains images with a distribution similar to those on which the original SEEM was trained. The other three datasets, selected from the medical domain, differ significantly from the original training domain of SEEM. In addition, 'Polyp'[1] and 'Liver'[2] are binary dataset whereas the '$Cityscape + PlantVillage$[3]' and 'MRI'[4] are multi-class dataset.

The '$Cityscape + PlantVillage$' dataset includes 3,000 images for training and 500 images for validation, covering 5 categories. Meanwhile, the MRI dataset comprises 500 training images and 100 validation images, covering 4 categories. For the binary datasets, the Polyp dataset contains 800 training images and 150 validation images, while the Liver dataset has 500 training images and 100 validation images.

Experiments demonstrate that with a batch size of 4, the model achieved its optimal performance after 3,150 steps of optimizer.

Examples of generated images and their corresponding masks are displayed in Figure B6. This generated dataset is employed for fine-tuning the SEEM model using the

[1] Kvasir-SEG: A Segmented Polyp Dataset
[2] Abdominal CT and MRI
[3] PlantVillage dataset
[4] Abdominal CT and MRI

candidate fine-tuning strategy, as detailed in Section 4.4.1.

4.2 Evaluation Metrics

4.2.1 Mean Intersection over Union

Mean Intersection over Union (mIoU) is an evaluation metric commonly used in computer vision tasks, particularly in image segmentation. It is used to measure the accuracy of a segmentation model by comparing the overlap between the predicted segmentation masks and the ground truth masks.

To evaluate the performance of a segmentation model using mIoU, several steps are followed. Firstly, for each class, the intersection over union (IoU) is computed. This involves calculating the intersection between the predicted mask and the corresponding ground truth mask.

The intersection represents the area where the predicted and ground truth masks overlap. Additionally, the union of the predicted and ground truth masks is calculated, representing the total area covered by both masks. The IoU for a class is obtained by dividing the intersection by the union. This process is repeated for each class.

To calculate the IOU, after calculating the confusion matrix, we can easily calculate IOU as follow:

$$IOU = \frac{TruePositive}{GroundTruthPositive + PredictedPositives - TruePositive} \tag{12}$$

- Ground Truth Positive: represents the total number of actual instances for each class. This is obtained by summing the elements of each column of confusion matrix
- Predicted Positives: represents the total number of predicted instances for each class. This is obtained by summing the elements of each row of confusion matrix
- Reason for subtracting the True Positive is that is has been counted twice (both in Ground Truth Positives and Predicted Positives).

Next, the IoU values obtained for each class are averaged to compute the mean IoU (mIoU). This is done by summing up the IoU values for all classes and dividing the sum by the total number of classes. The mIoU provides an overall measure of the accuracy of the segmentation model, taking into account the overlap between the predicted masks and the ground truth masks for each class.

The resulting mIoU value ranges from 0 to 1, where a value of 1 indicates a perfect alignment between the predicted masks and the ground truth masks for all classes, while a value of 0 signifies no overlap between them.

Using mIoU as an evaluation metric allows for a comprehensive assessment of the segmentation model's performance, considering both true positives and false positives across all classes. It enables fair comparisons between different segmentation models or different variations of a single model, helping researchers and practitioners gauge the effectiveness of their segmentation algorithms.

4.2.2 Precision

Precision (also known as positive predictive value) measures the proportion of predicted positive instances that are actually correct. A high precision indicates that the model's positive predictions are generally correct, but it does not account for how many relevant segments are missed.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{13}$$

4.2.3 Recall

Recall (also known as sensitivity or true positive rate) measures the proportion of actual positive instances (e.g., pixels belonging to the target object) that are correctly identified by the model.

A high recall indicates that the model successfully captures most of the relevant segments, but it does not account for the accuracy of those segments.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{14}$$

4.3 Experimental Details

In the following section we will talk about the different hyper-parameters in used during the experiments

4.3.1 Segmentation Task

For the segmentation task, weight decay is a crucial hyperparameter. The original SEEM model was pretrained with a weight decay of 0.05. Experiments indicate that reducing this parameter slightly improves results, but there is a risk of overfitting. Another important hyperparameter is the learning rate. Experiments reveal that using an initial learning rate of 1×10^{-4} leads to better convergence and improved results.

4.3.2 Dataset Generation

For dataset generation, a key hyperparameter is the threshold applied to the results of A_C^* for mask extraction, with experiments showing optimal values between 0.6 and 0.7.

Another important hyperparameter is called strength. Higher values lead to more modifications in the latent representation of the input image. Experiments indicate that a value of 0.4 ensures the generated images remain within the distribution of the input image.

4.4 Results

In this section we discuss the performance of different strategies for fine-tuning the SEEM in order to identify the the promising one, next, it will be used to assess the performance of model on synthetic dataset and understand if using synthetic dataset will be promising or not.

4.4.1 Fine-tuning Strategy Results

As demonstrated in Table 1, we can conclude that methodologies Freeze vision backbone, adapter fine-tuning for segmentation head and Adapter fine-tuning for vision backbone, adapter fine-tuning for segmentation head are not beneficial, Therefore, we will use the other two methods for the remaining evaluations.

Based on the evaluation of the results in Table 1, we can conclude that Across all datasets the "Adapter in Vision, Fully Fine-Tune Seg. Head" method consistently achieves the highest mIOU scores, indicating its superior performance. The "Freeze Vision, Fully Fine-Tune Seg. Head" method also performs well but is consistently outperformed by the adapter method. In contrast, the "Freeze Vision, adapter in Seg. Head" and "Adapter in Vision and Seg. Head" methods show relatively lower mIOU scores across all datasets, demonstrating that fully fine-tuning the segmentation head yields better results in vision tasks.

Table 1: Comparison of Different Fine-tuning Methods

Datasets	Methods	mIOU
Polyp	FVBAFSH[5]	55.57
	AFTBAFSH[6]	54.47
	FVBFWSH[7]	68.46
	AFTVBFWSH[8]	**69.18**
Cityscape + Plant Village	FVBAFSH	62.34
	AFTBAFSH	64.81
	FVBFWSH	75.73
	AFTVBFWSH	**78.72**
Liver	FVBFWSH	76.04
	AFTVBFWSH	**80.24**
MRI	FVBFWSH	59.46
	AFTVBFWSH	**64.72**

4.4.2 Fine-Tuning Performance on Synthetic Dataset

In this section, we assess the performance of the candidate fine-tuning strategy to determine the effectiveness of the generated dataset for transfer learning. The chosen dataset for this section consists of 9 categories relevant to the project's actual task, with class labels as follows: 1) Light Mold, 2) Dark Mold, 3) Light Stain, 4) Dark Stain, 5) Liquid Spillage, 6) Peeling, 7) Broken Pipe, 8) Raised, and 9) Pipe.

Table A2 shows, In the initial results without synthetic data, the model shows very low performance across most categories, with an overall mIOU of just 13.38. Categories like 1, 4, and 5 have zero or near-zero scores, indicating the model struggles significantly with the limited real data. The highest category performance is in category 9 with a score of 45.03, but overall, the model's efficacy is notably poor.

Upon introducing synthetic data, there is a marked improvement in performance. With a dataset size of 0.5K, the $mIOU$ increases to 21.46. Specific categories like 2, 3, and 5 exhibit substantial gains—category 2 jumps from 2.56 to 34.86, and category 5 leaps from 0.00 to 49.93. This improvement demonstrates that even a modest amount of synthetic data can significantly enhance the model's ability to generalize and recognize different categories better.

Increasing the synthetic dataset to 6K further enhances performance, raising the mIOU to 26.73. Categories 2, 3, and 5 continue to see improvement, with scores climbing to 47.37, 34.07, and 63.58, respectively. However, some categories, such as category 6, exhibit a slight decline in performance when transitioning from 0.5K to 6K. This suggests that although more data generally improves results, the quality and balance of the data are also crucial considerations.

The introduction of synthetic data generated by generative AI significantly enhances the performance of the model, as evidenced by the increases in mIOU and individual category scores. While a larger synthetic dataset (6K) generally leads to better results

[8]Freeze Vision Backbone, Adapter Fine-Tuning for Segmentation Head
[8]Adapter Fine-Tuning for Backbone, Adapter Fine-Tuning for Segmentation Head
[8]Freeze Vision Backbone, Fine-Tune Whole Segmentation Head
[8]Adapter Fine-Tuning for Vision Backbone, Fine-Tune Whole Segmentation Head

compared to a smaller one (0.5K), it is also apparent that more data alone is not the sole determinant of success. Ensuring high-quality and well-balanced synthetic data is crucial to achieving optimal performance. Thus, leveraging generative AI to create synthetic datasets is a viable strategy to overcome limitations due to the scarcity of real data, provided that attention is paid to both the quantity and quality of the generated data.

	IOU%									mIOU
Categories	1	2	3	4	5	6	7	8	9	
No Synthetic Aug.(baseline)	0.00	2.56	9.26	0.20	0.00	**27.48**	10.80	**25.07**	45.03	13.38
Synthetic Aug. 0.5K	0.00	34.86	23.81	0.00	49.93	3.82	**41.06**	2.27	37.36	21.46
Synthetic Aug. 6K	0.00	**47.37**	**34.07**	**6.78**	**63.58**	3.75	38.89	0.70	**45.40**	**26.73**

Table 2: Comparing how well the SEEM model performs on datasets of varying sizes and characteristics

5 Conclusion

In conclusion, the integration of synthetic data generated by generative AI has demonstrated significant improvements in the performance of our image segmentation model. The introduction of synthetic datasets has led to marked enhancements in the model's mean Intersection over Union (mIOU) scores and the recognition of specific categories. For instance, with a modest synthetic dataset of 0.5K images, the mIOU improved by 8%. Increasing the synthetic dataset to 6K images resulted in an even higher mIOU of 26.73. For some categories, there were significant improvements, However, it was noted that certain areas, such as one particular area, showed a slight decline in performance as the dataset size increased. This suggests that while more data generally enhances model performance, the quality of images that we use to sample from is important factors to consider. These results underscore the potential of synthetic data, particularly with larger datasets, to significantly boost the model's ability to generalize and recognize different categories effectively.

The evaluation of fine-tuning methods revealed that the "Adapter in Vision and Fully Fine-Tune Seg. Head" approach consistently achieved the highest mIOU scores across all datasets, indicating its superior effectiveness. The "Freeze Vision, Fully Fine-Tune Seg. Head" method also performed well, though it was consistently outperformed by the

adapter method.

To further improve the results, it is essential to enhance the quality and balance of the synthetic data. Refining the generative AI models to produce more realistic and diverse images will ensure better representation of real-world variations. Additionally, combining synthetic data with real annotated data can leverage the strengths of both, providing authenticity and increasing volume and diversity. Regularly evaluating and fine-tuning the model, along with exploring advanced data augmentation techniques and optimizing model architecture, will also contribute to better performance. Implementing active learning strategies can further enhance training efficiency and effectiveness, ensuring continuous improvement in the model's accuracy and reliability.

Future work should focus on hybrid dataset development and advanced data augmentation techniques to enhance image segmentation models. This includes creating balanced integration methods for synthetic and real data, developing quality assessment tools for synthetic datasets, and employing domain adaptation and transfer learning to bridge the gap between synthetic and real-world data. By combining these strategies, models can be trained on richer and more varied datasets, leading to improved accuracy, robustness, and generalization in real-world applications.

Overall, the integration of synthetic data generated by generative AI, combined with effective fine-tuning strategies, has proven to be a viable solution to the challenges posed by limited annotated data. While increasing the quantity of synthetic data can lead to better model performance, ensuring the quality and balance of this data is essential. Therefore, leveraging generative AI for synthetic dataset creation, coupled with appropriate fine-tuning techniques, presents a promising approach to enhancing image segmentation models, ultimately leading to more accurate and reliable results.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2016.

[3] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, November 2022.

[4] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[6] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context, 2018.

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[8] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.

[10] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[11] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition, 2022.

[12] Xi Chen, Zhiyan Zhao, Yilei Zhang, Manni Duan, Donglian Qi, and Hengshuang Zhao. Focalclick: Towards practical interactive image segmentation, 2022.

[13] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions, 2023.

[14] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation, 2022.

[15] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion, 2021.

[16] Junlong Cheng, Jin Ye, Zhongying Deng, Jianpin Chen, Tianbin Li, Haoyu Wang, Yanzhou Su, Ziyan Huang, Jilong Chen, Lei Jiang, Hui Sun, Junjun He, Shaoting Zhang, Min Zhu, and Yu Qiao. Sam-med2d, 2023.

[17] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, January 2018.

[18] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10850–10869, September 2023.

[19] Jifeng Dai, Kaiming He, and Jian Sun. Convolutional feature masking for joint object and stuff segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015.

[20] detectron2 contributors Revision 8c4a333c. detectron2.data, 2019-2020.

[21] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021.

[22] Zheng Ding, Jieke Wang, and Zhuowen Tu. Open-vocabulary universal image segmentation with maskclip, 2023.

[23] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp, 2017.

[24] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[25] Peng Gao, Minghang Zheng, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fast convergence of detr with spatially modulated co-attention, 2021.

[26] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection, 2019.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[28] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[32] Geremy Heitz and Daphne Koller. Learning spatial context: Using stuff to find things. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 30–43, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[33] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[35] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[36] Mingzhe Hu, Shaoyan Pan, Yuheng Li, and Xiaofeng Yang. Advancing medical imaging with language models: A journey from n-grams to chatgpt, 2023.

[37] Shijia Huang, Feng Li, Hao Zhang, Shilong Liu, Lei Zhang, and Liwei Wang. A unified mutual supervision framework for referring expression segmentation and generation, 2022.

[38] Guillermo Iglesias, Edgar Talavera, and Alberto Díaz-Álvarez. A survey on gans for computer vision: Recent research, analysis and taxonomy. *Computer Science Review*, 48:100553, May 2023.

[39] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning, 2022.

[40] Michael Kass, Andrew P. Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 2004.

[41] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2021.

[42] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[44] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[45] Feng Li, Ailing Zeng, Shilong Liu, Hao Zhang, Hongyang Li, Lei Zhang, and Lionel M. Ni. Lite detr: An interleaved multi-scale encoder for efficient detr. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18558–18567, June 2023.

[46] Feng Li, Hao Zhang, Huaizhe xu, Shilong Liu, Lei Zhang, Lionel M. Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation, 2022.

[47] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[48] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[49] Jiang Liu, Hui Ding, Zhaowei Cai, Yuting Zhang, Ravi Kumar Satzoda, Vijay Mahadevan, and R. Manmatha. Polyformer: Referring image segmentation as sequential polygon generation, 2023.

[50] Qin Liu, Zhenlin Xu, Gedas Bertasius, and Marc Niethammer. Simpleclick: Interactive image segmentation with simple vision transformers, 2023.

[51] Shervin Minaee and Yao Wang. An admm approach to masked signal decomposition using subspace representation. *IEEE Transactions on Image Processing*, 28(7):3192–3204, 2019.

[52] Laurent Najman and Michel Schmitt. Watershed of a continuous function. *Signal Processing*, 38(1):99–112, 1994. Mathematical Morphology and its Applications to Signal Processing.

[53] Dhanachandra Nameirakpam, Khumanthem Singh, and Yambem Chanu. Image segmentation using k -means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 12 2015.

[54] Jiquan Ngiam, Zhenghao Chen, Pang W Koh, and Andrew Y Ng. Learning deep energy models. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1105–1112, 2011.

[55] Quang Nguyen, Truong Vu, Anh Tran, and Khoi Nguyen. Dataset diffusion: Diffusion-based synthetic dataset generation for pixel-level semantic segmentation, 2023.

[56] R. Nock and F. Nielsen. Statistical region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1452–1458, 2004.

[57] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization, 2016.

[58] Otsu and Nobuyuki. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[59] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10213–10224, June 2021.

[60] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[61] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[62] Gaurav Raut and Apoorv Singh. Generative ai in vision: A survey on models, metrics and applications, 2024.

[63] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[64] Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *ArXiv*, abs/1810.00597, 2018.

[65] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[67] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[68] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[69] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[70] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

[71] J.-L. Starck, M. Elad, and D.L. Donoho. Image decomposition via the combination of sparse representations and a variational approach. *IEEE Transactions on Image Processing*, 14(10):1570–1582, 2005.

[72] Zhiqing Sun, Shengcao Cao, Yiming Yang, and Kris Kitani. Rethinking transformer-based set prediction for object detection, 2021.

[73] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks, 2022.

[74] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):1922–1933, 2022.

[75] Joseph Tighe and Svetlana Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3001–3008, 2013.

[76] Jakub M. Tomczak. *Deep Generative Modeling*. Springer Cham, 1st edition, 2022.

[77] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259, 2003.

[78] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks, 2016.

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[80] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

[81] Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers, 2021.

[82] Junde Wu, Wei Ji, Yuanpei Liu, Huazhu Fu, Min Xu, Yanwu Xu, and Yueming Jin. Medical sam adapter: Adapting segment anything model for medical image segmentation, 2023.

[83] Mengde Xu, Zheng Zhang, Fangyun Wei, Han Hu, and Xiang Bai. Side adapter network for open-vocabulary semantic segmentation, 2023.

[84] Jianwei Yang, Chunyuan Li, Xiyang Dai, Lu Yuan, and Jianfeng Gao. Focal modulation networks, 2022.

[85] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Bin Xiao, Ce Liu, Lu Yuan, and Jianfeng Gao. Unified contrastive learning in image-text-label space, 2022.

[86] Linwei Ye, Mrigank Rochan, Zhi Liu, and Yang Wang. Cross-modal self-attention network for referring image segmentation, 2019.

[87] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[88] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.

[89] Xueyan Zou, Zi-Yi Dou, Jianwei Yang, Zhe Gan, Linjie Li, Chunyuan Li, Xiyang Dai, Harkirat Behl, Jianfeng Wang, Lu Yuan, Nanyun Peng, Lijuan Wang, Yong Jae Lee, and Jianfeng Gao. Generalized decoding for pixel, image, and language, 2022.

[90] Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. Segment everything everywhere all at once, 2023.

Appendix A: Tables

Recall%									
Categories	1	2	3	4	5	6	7	8	9
Initial results	0.00	?10.34	32.45	2.34	0.00	32.56	13.48	29.54	52.15
Synthetic 0.5K	0.00	48.86	34.45	0.00	58.76	8.75	60.65	15.45	41.63
Synthetic 6K	0.00	47.90	70.42	24.65	68.91	3.88	51.79	1.61	54.64

Table A1: Comparison of the SEEM model's Recall metric performance across datasets of different sizes and characteristics.

Precision%									
Categories	1	2	3	4	5	6	7	8	9
Initial results	0.00	55.86	72.45	0.00	0.00	67.45	60.87	35.87	37.36
Synthetic 0.5K	0.00	62.98	77.48	0.00	61.34	68.34	55.93	32.84	69.52
Synthetic 6K	0.00	97.72	39.76	8.54	89.15	51.81	60.96	1.22	72.87

Table A2: Comparison of the SEEM model's Precision metric performance across datasets of different sizes and characteristics.

Appendix B: Appendix-Figures

Figure B1: generated image based on prompt: "a pipe next to wall with some grass on it"

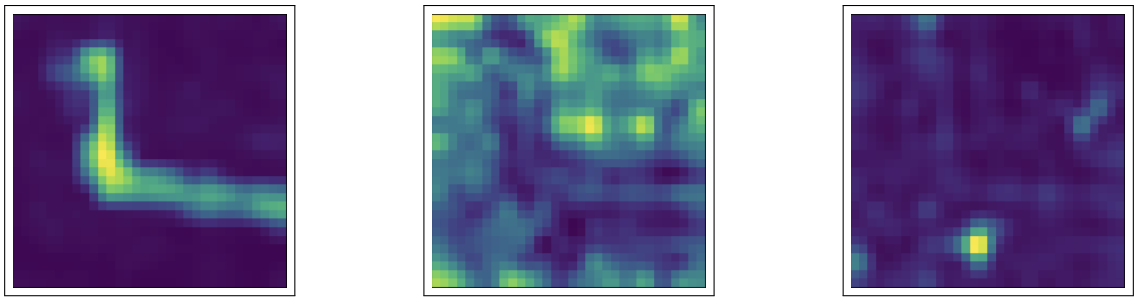


Figure B2: cross-attention maps for tokens 'pipe', 'wall' and 'grass' respectively

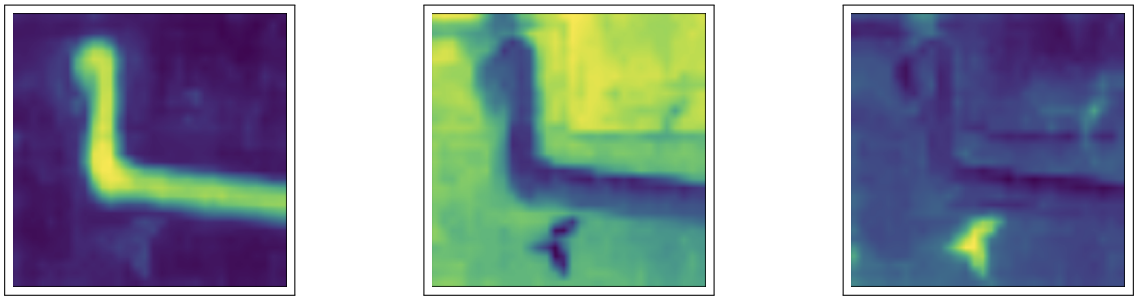
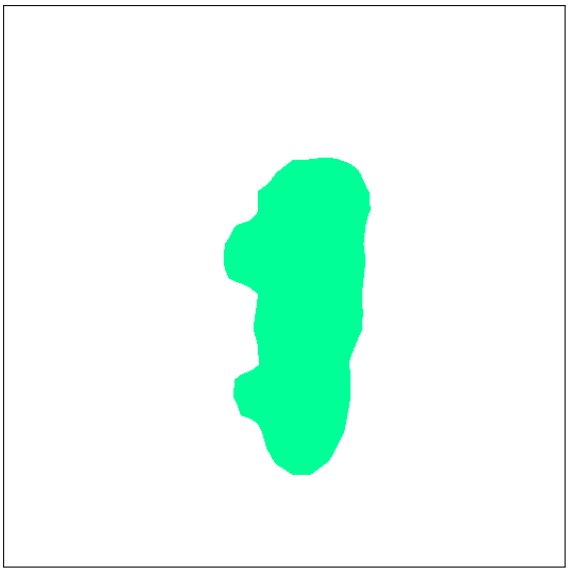


Figure B3: The feature maps generated by multiplying cross-attention and self-attention maps for tokens 'pipe', 'wall', and 'grass', respectively.

Figure B4: generated masks for tokens 'pipe', 'wall' and 'grass' respectively

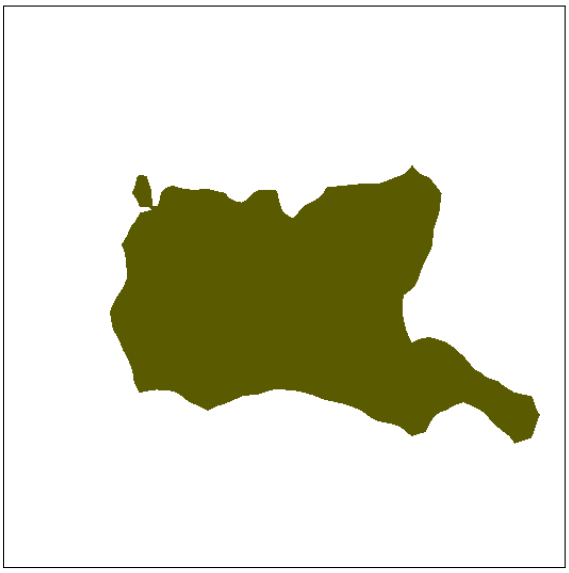
Examples of generated images and corresponding masks are show in Fig. B6

(a) Generated Image for 'Broken Pipe' category



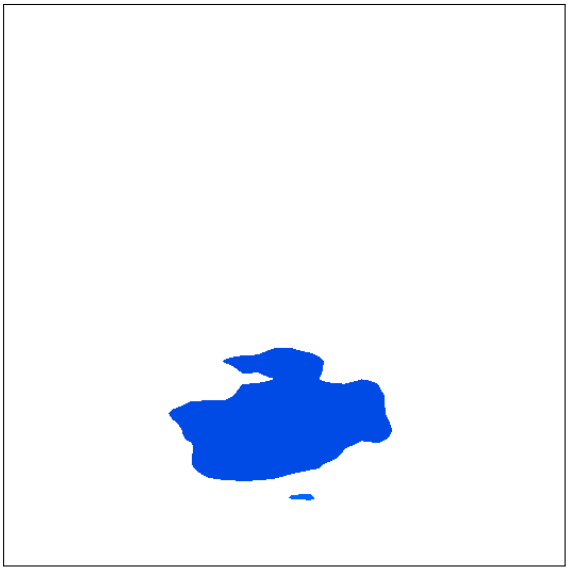
(b) Generated mask for 'Broken Pipe' category

(c) Generated Image for 'Dark Mold' category

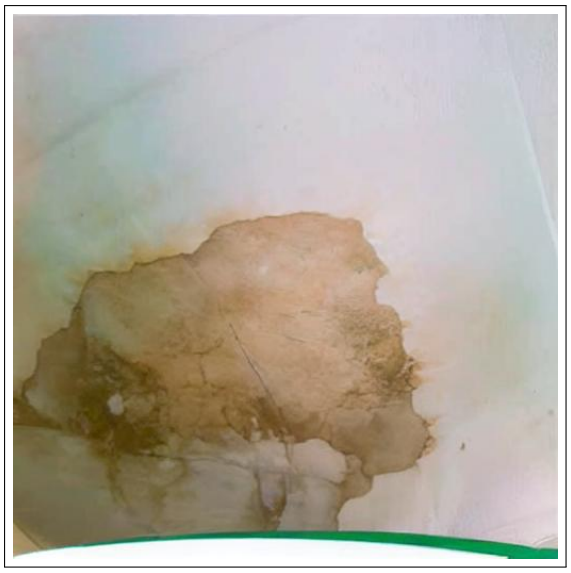


(d) Generated mask for 'Dark Mold' category

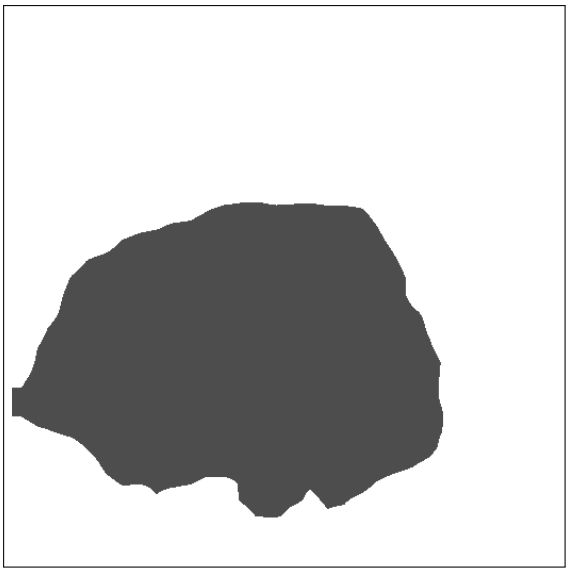
(e) Generated Image for 'Water spillage' category



(f) Generated mask for 'Water spillage' category

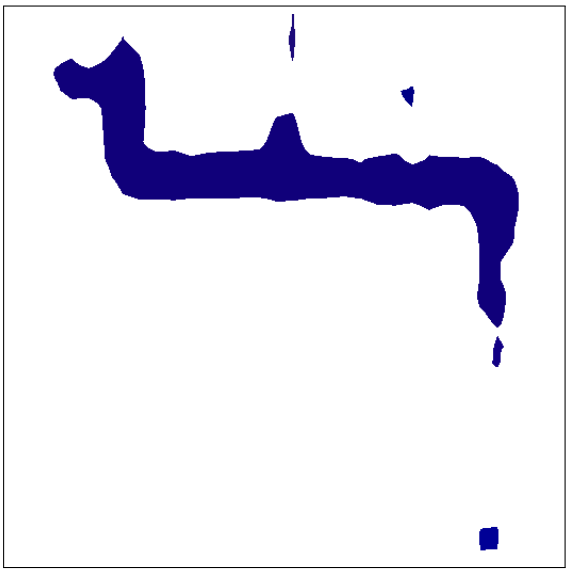


(a) Generated Image for 'Dark Stain' category



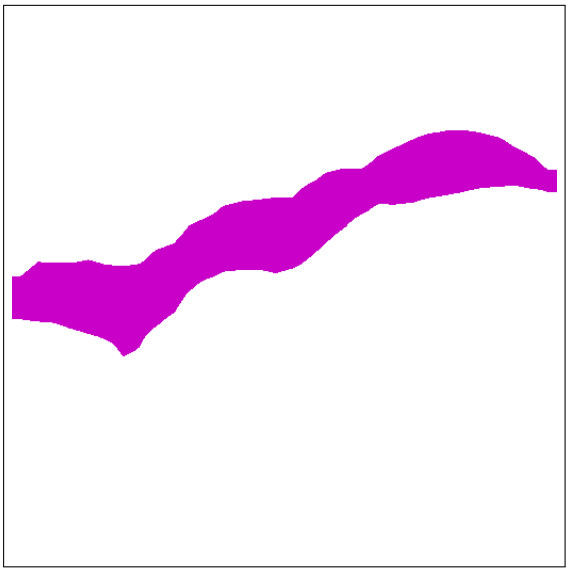
(b) Generated mask for 'Dark Stain' category

(c) Generated Image for 'Pipe' category



(d) Generated mask for 'Pipe' category

(e) Generated Image for 'Peeling' category



(f) Generated mask for 'Peeling' category

Figure B6: Generated Images and Corresponding masks