POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Diagnostic Text as Supervision:
## A Contrastive Approach to Language-Conditioned Learning for Semantic Segmentation

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Luca Olivieri**

Student ID: 247455
Advisor: Prof. Matteo Matteucci
Co-advisors: Nico Catalano, Sara Pidò, Chiara Plizzari
Academic Year: 2024-2025

# Abstract

Deep Learning has achieved remarkable success in semantic segmentation, enabling models to learn complex visual patterns and reach state-of-the-art performance. However, conventional training procedures rely heavily on numerical loss functions that do not necessarily encourage robust or semantically meaningful representations. As a result, models may exploit spurious correlations or superficial cues, limiting generalization and negatively affecting performance. This work investigates a novel training paradigm for semantic segmentation that integrates language-based diagnostic feedback into the optimization process. Instead of relying solely on abstract numerical errors, the proposed approach leverages multi-modal language models to generate semantically interpretable descriptions of prediction errors, such as incorrect object boundaries or improper merging of adjacent regions. These textual diagnostics are integrated into the training via a contrastive loss, which guides the segmentation model to align its visual features with the textual descriptions, grounding them in meaningful spatial properties. A comprehensive experimental pipeline is developed to evaluate this language-guided training strategy through systematic experiments that analyze the impact of diagnostic text on learning dynamics and segmentation performance. The results highlight both the potential and the limitations of integrating linguistic guidance into spatial vision tasks. Specifically, experiments reveal that the proposed contrastive objective interferes with the primary segmentation task, introducing instability and progressively degrading performance as its influence increases. These findings frame the joint optimization of pixel-level prediction and diagnostic text alignment as a challenging objective for segmentation encoders within the examined setup. Overall, this work advances the understanding of how foundation language models can shape representation learning in semantic segmentation, laying the groundwork for more generalizable and semantically grounded Deep Learning systems.

**Keywords:** semantic segmentation; multi-modal large language models; language-guided training; contrastive learning; representation learning

# Abstract in lingua italiana

Il Deep Learning ha ottenuto grandi successi nella segmentazione semantica, permettendo ai modelli di apprendere pattern visivi complessi e raggiungere prestazioni all'avanguardia. Tuttavia, le procedure di apprendimento convenzionali fanno largo uso di funzioni di loss numeriche che non sempre favoriscono rappresentazioni robuste o semanticamente significative. I modelli rischiano quindi di sfruttare correlazioni spurie o indizi superficiali, riducendo la generalizzazione e peggiorando le prestazioni. Questo lavoro esplora un nuovo approccio alla segmentazione semantica che integra feedback diagnostici basati sul linguaggio nel processo di ottimizzazione. Anziché affidarsi solo a errori numerici astratti, il metodo proposto utilizza modelli linguistici multi-modali per generare descrizioni interpretabili degli errori di previsione, come contorni degli oggetti errati o fusione impropria di regioni adiacenti. Queste diagnostiche testuali vengono integrate nell'apprendimento tramite una loss contrastiva, che guida il modello di segmentazione ad allineare le sue rappresentazioni visive con le descrizioni testuali, ancorandole in proprietà spaziali significative. Una pipeline sperimentale completa viene sviluppata per valutare questa strategia di apprendimento guidata dal linguaggio tramite esperimenti sistematici che analizzano l'impatto dei testi diagnostici sulle dinamiche di apprendimento e sulle prestazioni di segmentazione. I risultati mostrano sia il potenziale sia i limiti dell'integrazione del linguaggio nei task visivi. Nello specifico, gli esperimenti rivelano come l'obiettivo contrastivo interferisca con il compito principale di segmentazione, introducendo instabilità e degradando progressivamente le prestazioni al crescere della sua influenza. Ciò delinea l'ottimizzazione congiunta tra previsione a livello di pixel e allineamento testuale come un obiettivo complesso per gli encoder di segmentazione nel setup esaminato. Complessivamente, questo lavoro amplia la comprensione di come i modelli linguistici possano plasmare l'apprendimento delle rappresentazioni nella segmentazione semantica, gettando le basi per sistemi di Deep Learning più generalizzabili e semanticamente robusti.

**Parole chiave:** segmentazione semantica; modelli linguistici multi-modali; apprendimento guidato dal linguaggio; apprendimento contrastivo; apprendimento delle rappresentazioni

# Contents

# 1 | Introduction

## An overview on Deep Learning and Semantic Segmentation

In recent years, Deep Learning (DL) has transformed many areas of machine learning. Thanks to the ability of Neural Networks (NNs) to learn complex patterns from data, these models now achieve state-of-the-art results across a wide range of tasks and domains. One of the most successful applications of DL architectures is found in Semantic Segmentation (SS) [58], a Computer Vision (CV) task in which models are leveraged to identify class regions in an image. This task greatly benefited from the ability of NNs to reason with both global context and local details, which is crucial for providing accurate results.

The learning of DL models relies on loss functions, which measure the difference between predictions and ground truth, driving gradient-based optimization [22]. Despite their effectiveness, minimizing a loss does not always lead to generalizable behavior [9]. Loss functions can unintentionally push models toward brittle or superficial patterns instead of meaningful representations. For example, in image classification, cross-entropy loss may be minimized by exploiting spurious cues like background textures rather than actual object features. In regression, mean squared error penalizes large errors more heavily, but this does not guarantee that the model learns contextually meaningful relationships. The choice of the loss function strongly affects whether a model captures robust, semantic features or relies on dataset-specific shortcuts. However, this influence is often largely understood through experimentation rather than theoretical guarantees. This makes it hard to ensure that predictions reflect human-relevant reasoning rather than superficial statistical patterns [68].

To address this issue, researchers have explored methods that integrate human knowledge into the learning process. By incorporating expertise and semantically meaningful guidance, these approaches steer DL models toward robust, generalizable representations instead of superficial patterns. Human insight helps models focus on relevant features, such as object boundaries or spatial relationships, connecting abstract optimization to real-world understanding. These frameworks, grouped under the umbrella term of Human-In-The-Loop (HITL), improve the robustness, reliability, and cross-domain transferability

of automated systems [70, 71].

## Integrating Human Knowledge in Semantic Segmentation

In the context of SS, HITL techniques yield promising results in Interactive Semantic Segmentation, where visual hints (e.g., scribbles, bounding boxes, or arrows) or language cues (e.g., "segment the car") serve as supervisory signals to refine segmentation masks [28, 71]. However, existing methods either underutilize the full expressive potential of language or operate exclusively during the network's prediction phase at inference time, leaving HITL techniques that directly influence the training procedure largely underexplored.

Indeed, implementing HITL during training presents several significant challenges, particularly regarding architecture design and the sustainability of human intervention. Indeed, human supervision is expensive: annotators may spend several minutes per image to provide detailed corrections, and large-scale datasets often require thousands of hours of expert time. For instance, training a modern segmentation model on datasets like Cityscapes [17] or ADE20K [101] would require continuous human feedback across hundreds of thousands of training iterations, making real-time human involvement impractical. Therefore, HITL approaches must achieve significant automation and efficiency improvements to be feasible at the scale of full training.

Given these constraints, foundation models offer a promising solution to scale HITL approaches. Foundation models are large-scale neural networks pre-trained on vast, diverse datasets to learn general-purpose representations and patterns [10]. Rather than being designed for a single narrow task, they acquire broad knowledge that can be adapted to many different applications. A prime example comes from Natural Language Processing (NLP): models like GPT [11, 66] are trained on enormous text corpora and learn not just grammar and vocabulary, but also reasoning patterns, common knowledge, and even human biases present in their training data. This enables them to perform remarkably well on tasks ranging from translation to question-answering to creative writing, often producing credible output that approaches human-level performance. For instance, a language model can draft coherent research summaries, generate code documentation, or compose structured reports.

This paradigm has recently extended beyond text. Modern multi-modal foundation models, such as GPT-4 [1], are trained to process and relate multiple data modalities, including text, images, and audio, within a shared representation space. By aligning linguistic and perceptual information, they can perform tasks such as image captioning, visual question answering, and cross-modal reasoning, further expanding the applicability of foundation

models in HITL settings.

Crucially for HITL applications, foundation models can automate tasks that traditionally required human experts. Within SS, multi-modal foundation models can leverage their broad visual and contextual understanding to provide preliminary masks or refinements similar to what a human annotator would produce, dramatically reducing the amount of actual human intervention needed during training [45, 53, 77, 92]. Rather than requiring continuous human feedback across hundreds of thousands of iterations, the foundation model serves as a learned proxy for human judgment, stepping in where direct human involvement would be too expensive.

## Guiding Semantic Segmentation through Diagnostic Text

Foundation language models, capable of analyzing segmentation masks, offer a potential approach to address some limitations of conventional loss functions in the SS domain, such as sensitivity to spurious and non-generalizable correlations. Instead of relying exclusively on numerical error measures, foundation models can generate textual descriptions that provide semantically interpretable feedback. For example, a language model might produce statements such as "the predicted mask includes part of the sidewalk within the road region" or "the boundary between the car and background is segmented roughly", translating quantitative errors into observations about spatial relationships and object boundaries. The task of SS is well-suited to this approach, as errors often manifest as distinct spatial differences that can be described using human-understandable language.

Using this linguistic grounding, the learning process can be guided by automatically generated textual feedback to steer the model toward learning more robust, generalizable representations rather than exploiting implicit artifacts or low-level correlations. In this paradigm, the loss function not only minimizes abstract error metrics that may be satisfied by learning spurious shortcuts, but actively encourages the model to follow language-based objectives tied to meaningful semantic properties that work across different contexts. This approach transforms the potentially fragile nature of gradient descent into a semantically-grounded process where the model is encouraged to focus on robust visual features, such as object boundaries, spatial relationships, and contextual coherence, that align with how humans understand scenes, rather than latching onto superficial statistical patterns. In this work, I explore a concrete mechanism to enable this interaction between foundation language models and segmentation models.

## Contributions

In the context of segmentation training frameworks, I propose a novel training framework integrating a contrastive loss component that conditions the internal representations learned by the segmentation model based on the content of automatically generated diagnostic text. This contrastive objective encourages the model to form representations that align with linguistic descriptions of meaningful visual properties and prediction errors, rather than solely optimizing for abstract numerical targets that may be achieved through spurious correlations. For instance, if the diagnostic text identifies "incorrect merging of adjacent buildings," the model's feature space is shaped to distinguish these structures based on their semantic identity and spatial relationships. By linking the model's learned representations to natural language descriptions grounded in human understanding, this approach creates a feedback loop where the mathematical optimization process is steered toward semantically meaningful, generalizable features. The goal is to develop a training procedure where loss minimization is decoupled from spurious shortcuts and is instead guided by textual representations of robust visual properties, aiming to enable more generalizable and reliable DL for SS.

To validate this approach, I design a comprehensive pipeline and conduct extensive experiments across diverse scenarios to systematically evaluate its behavior and effectiveness. Through rigorous ablation studies, I analyze the contribution of each module to the training process, examining how different architectural components and design choices influence learning dynamics and overall performance. This investigation yields significant insights into both the potential and limitations of leveraging language-based feedback within the SS domain. This work contributes to advancing our understanding of how linguistic guidance can be effectively integrated into DL pipelines for vision tasks.

# Outline

This section presents the overall structure of the thesis and summarizes the content of each chapter.

**Chapter 1** introduces the main themes of the work, outlining its motivation and defining its scope.

**Chapter 2** reviews the foundational concepts underlying this work, including the task of SS, language models, vision-language models, and the broader field of language supervision.

**Chapter 3** describes the proposed methodology, detailing the generation and encoding of

diagnostic text and explaining how these components interact with segmentation models within the language-guiding pipeline.

**Chapter 4** describes the experimental setup, the evaluation framework, and the empirical results. It begins with the preliminary study for module selection, then analyzes the language-guiding mechanism in its entirety. Each experiment is presented individually, with corresponding results reported and discussed.

**Chapter 5** recaps the principal outcomes of the thesis, examines its limitations, and outlines potential directions for future research developments.

# 2 | Background

This chapter presents the fundamental notions and concepts upon which this work is built, and hence deserve to be described and explained thoroughly.

Section 2.1 provides foundational knowledge on the task of SS.

Section 2.2 introduces the text generation task and the techniques commonly used to address it.

Section 2.3 illustrates vision-language models, which operate at the intersection of the visual and textual modalities.

Section 2.4 contextualizes prompt engineering, which covers the methods leveraged to interact with language models.

Section 2.5 covers the concepts and state-of-the-art techniques related to SS language supervision, which analyzes how to leverage language supervisory signals to improve models' performances.

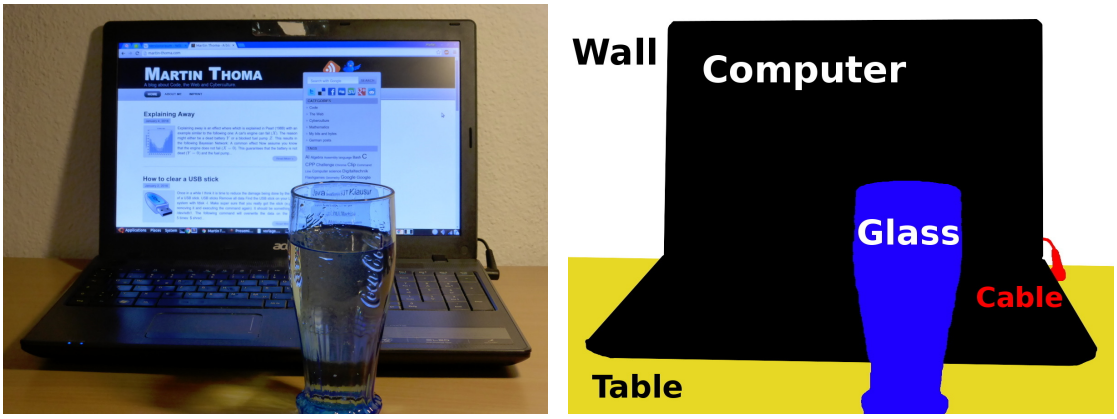
## 2.1. Semantic Segmentation

This section describes the task, the models, and the datasets related to semantic segmentation, one of the frameworks considered in this work.

### 2.1.1. Task

SS is a core task in image processing and CV, and it falls within the broader field of image segmentation. Its objective is to partition images into semantic regions by assigning a semantic class label to each pixel [18]. The resulting pixel-wise labels are organized into segmentation masks, which are typically visualized using color maps, as shown in Figure 2.1.

SS requires understanding an image at both global and local levels: globally, to recognize the overall scene and objects present, and locally, to precisely delineate object boundaries




(a) Example Scene. (b) Visualization of a found segmentation.

Figure 2.1: An example of a scene and a possible visualization of a found segmentation, from [79].

at the pixel level. To highlight this contrast, it is useful to first consider image classification, where the task is to assign a single class label to an entire image. Unlike image classification, which treats each image independently, SS must account for the relationships between neighboring pixels, combining local detail with global context to produce coherent, pixel-wise predictions [18].

The problem of SS has been of interest long before the advent of DL, hence numerous algorithms from supervised and unsupervised learning have been proposed (e.g., $k$-means clustering and Markov Random Fields [79]). However, over the past few years, DL has yielded a new generation of SS supervised models with remarkable performance improvements, often achieving the highest accuracy rates on popular benchmarks and resulting in a paradigm shift in the field [58].

The performance gap between traditional machine learning and DL methods can, arguably, be attributed to the fundamentally different ways in which they process information. From a machine learning perspective, SS "faces an inherent tension between semantics and location: global information resolves *what*, while local information resolves *where*" [52]. As a consequence, algorithms must be able to "jointly encode location and semantics" and "combine deep, coarse, semantic information and shallow, fine, appearance information" [52].

NNs, differently from older machine learning models, feature innovative architectures that possess the capacity to capture both fine-grained local features and coarse global features through their hierarchical representation learning. This multiscale feature extraction ability has enabled DL-based approaches to effectively bridge the gap between semantics and

localization, ultimately leading to their outstanding performance in SS tasks.

Practically speaking, the objective of a NN solving the SS task is to derive from data the meaningful patterns that enable an informed prediction of segmentation masks. A NN is defined as a function $f_\theta(\cdot)$, fully determined by its hyperparameters $\theta$, which dictate how it generates predictions. The model, through an optimization process, seeks an optimal set of hyperparameters $\theta^*$ within a larger hyperparameter space $\Theta$. This is achieved by minimizing a non-negative loss function $\mathcal{L}$ over a training dataset $D_{\text{train}}$, which is a collection of input-target pairs $\{(x_m, y_m)\}_{m=1}^{M}$, where $x_m$ and $y_m$ respectively belong to the input and target sets $X_{\text{train}}$ and $Y_{\text{train}}$. The loss function quantifies the discrepancy between the network's predictions $f_\theta(x_m)$ and the training targets $y_m$. By reducing this loss, the network gradually improves its predictive accuracy over the training samples. Let $f_\theta(X)$ be the set of model predictions, formally:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \, \mathcal{L}\left(f_\theta(X_{\text{train}}), Y_{\text{train}}\right), \tag{2.1}$$

In the context of SS, the inputs are RGB images, and the targets are segmentation masks that have been manually annotated to provide pixel-wise ground-truth labels.

During training, the models optimize specifically for the training data. NNs, with their high capacity and flexibility, can learn not only the underlying patterns but also the noise and spurious trends present in the training set. If this is the case, the networks' training performance can overestimate their true generalization ability, leading to worse performance on new, unseen data, because the patterns they memorized do not generalize beyond the training examples. This phenomenon is called overfitting [73].

As a consequence, the training procedure is continuously monitored by evaluating performance on a validation dataset $D_{\text{val}}$, which the model does not see during training. Validation plays a key role in model selection, as it provides a more reliable estimate of the model's generalization capabilities. However, selecting a model based on validation performance can lead to overfitting on the validation set. Therefore, to obtain an unbiased estimate of true generalization performance, the final evaluation of the selected model is carried out on a test dataset $D_{\text{test}}$, which remains completely unseen during training and model selection [73].

### 2.1.2. Encoder-Decoder Models

This subsection provides notions regarding the encoder-decoder architecture used in the SS models adopted in this work.

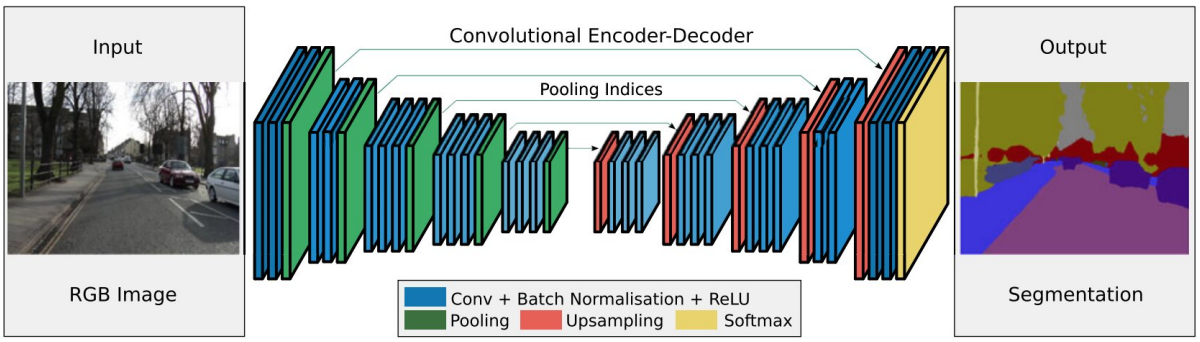



Figure 2.2: SegNet [6] is a fully convolutional model. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). Image from [58].

The encoder-decoder architecture is one of the most popular families of DL models for SS. Most of the DL-based segmentation works involve models belonging to this family [58]. This architecture consists of an encoder using convolutional layers, typically adopted from pre-trained visual encoders, and a deconvolutional network that takes the feature volume as input and generates a map of pixel-wise class probabilities [58]. The deconvolutional network is mainly composed of transpose convolutions and unpooling layers. Since encoder-decoder models output pixel-wise probability maps, the final segmentation mask is obtained by selecting the class with the highest probability for each pixel. An example of encoder-decoder architecture is shown in Figure 2.2.

In these architectures, the input image is encoded into a latent-space representation that condenses its semantic content. The decoder then reconstructs the output by progressively increasing the spatial resolution while reducing the number of channels in this representation [58].

Typically, encoder and decoder layers are linked through skip connections, enabling the decoder to reuse low-level spatial details from earlier layers to improve the accuracy of its upsampling operations [58].

### 2.1.3. Loss Functions

Loss functions play a central role in DL, defining how models learn and perform in different tasks. They measure the gap between predictions and actual labels, steering the optimization process to reduce errors. Choosing an appropriate loss function is essential, as it strongly influences model convergence, generalization, and overall effectiveness in various applications [22].

| Symbol | Description |
|---|---|
| $N$ | Number of pixels |
| $C$ | Number of target classes |
| $t_n$ | One-hot encoding vector representing the target class of the $n^{\text{th}}$ pixel. |
| $t_n^c$ | Binary indicator: 1 if the $n^{\text{th}}$ pixel belongs to class $c$, otherwise 0. |
| $y_n$ | Predicted class probabilities for $n^{\text{th}}$ pixel. |
| $y_n^c$ | Predicted probability of $n^{\text{th}}$ pixel belonging to class $c$. |
| $t_n \cdot y_n$ | Predicted probability for the target class of $n^{\text{th}}$ pixel. |

Table 2.1: Notation for SS losses, from [5].

Several years of research have produced many examples of successful loss functions in SS, able to guide the training of the neural networks to optimize different kinds of criteria. This subsection briefly illustrates the most relevant ones.

The following notions are sourced from [5], along with the formal notation used throughout the subsection and depicted in Table 2.1.

## Cross-Entropy Loss

The Cross-Entropy (CE) loss [40] quantifies the prediction error with a logarithmic measure of the discrepancy between the correct labels and their predicted probability, which favours convergence and reduces over-fitting. It is computed pixel-wise and then averaged over the entire image. Formally:

$$\mathcal{L}_{\text{CE}}(y, t) = -\frac{1}{N}\sum_{n=1}^{N}\log(t_n \cdot y_n). \tag{2.2}$$

As $t_n$ is a one-hot encoded vector, only the predicted probability of the target class affects the CE loss, and the batch-wise computation is performed by averaging over the batches. When dealing with imbalanced datasets, a common adopted strategy is to assign different weights to each class (often with inverse class frequencies).

Despite its popularity and simplicity, which allows treating SS as a pixel-wise classification problem, the CE loss presents some significant disadvantages:

- It does not acknowledge class imbalance, since pixels are considered separately. Hence, the majority classes tend to have a larger presence in the measure, pushing the model to favour over-represented classes and neglect under-represented ones.

This property may be more or less undesirable depending on the domain.

- It does not directly optimize overlap metrics, which are reasonable for this kind of region-based task.
- It penalises all errors equally, and it has no way to tune the penalty, unlike many other loss functions.

### Focal Loss

The Focal loss [49] was designed to downweight well-classified pixels and increase the presence of harder, misclassified examples. Built from the standard CE, it introduces a term that decays as the correct probability approaches 1, and the strength of this effect can be controlled by means of a hyperparameter. Formally:

$$\mathcal{L}_{\text{focal}}(y, t, \gamma) = -\frac{1}{N}\sum_{n=1}^{N}(1 - t_n \cdot y_n)^{\gamma}\log(t_n \cdot y_n). \tag{2.3}$$

The newly introduced term $(1-t_n \cdot y_n)^{\gamma}$ with $\gamma > 0$ (if $\gamma = 0$, the formula reduces to plain CE loss) smoothly brings to 0 the weight of the pixels which are confidently classified (i.e., with $t_n \cdot y_n \sim 1$), allowing the weights of confidently misclassified pixels (i.e., with $t_n \cdot y_n \sim 0$) to have a greater presence in the optimization process. The greater $\gamma$, the stronger the decay.

The Focal loss is frequently chosen in SS, as the property of dynamic, pixel-wise scaling often proves to be very effective, especially in multi-class scenarios, where some classes are more difficult to correctly predict than others (which may happen for reasons other than class imbalance). Additionally, the hyperparameter $\gamma$ allows for controlling the intensity of the scaling.

Regardless of these considerations, the focal loss comes with some disadvantages:

- The hyperparameter $\gamma$ has to be carefully tuned and validated to balance training stability and effectiveness.
- The constantly changing weights might cause instability, hinder the convergence process, and may lead to poor model calibration.
- The focal mechanism sometimes merely ends up acting as a class imbalance equalizer. In these cases, a fixed frequency-weighted loss might be a better choice.

### Dice Loss

The Dice loss [57] is a region-level loss function and deals with some issues of CE loss, expressing the overlap between the predicted segmentation and the ground truth. It is obtained through the average of a differential adaptation of the Dice coefficient across all classes. It can be written as:

$$\mathcal{L}_{\text{dice}} = 1 - \frac{1}{C}\sum_{c=0}^{C-1}\frac{2\sum_{n=1}^{N} t_n^c y_n^c}{\sum_{n=1}^{N}(t_n^c + y_n^c)}. \tag{2.4}$$

The Dice loss promotes maximisation of the overlap between the regions, and weighs all classes equally, regardless of their presence.

Despite the qualities that justify the wide adoption of this loss, it presents some undesirable properties:

- Classes with very low representation make the gradients numerically unstable, as a result of the excessively large gradients deriving from the very small denominator of the Dice coefficient.
- It is non-linear in the pixels and does not explicitly maximise likelihood, which leads to poor model calibration and an irregular surface.
- It might over-emphasise rare classes and under-emphasise frequent classes, leading to unstable convergence.

## 2.1.4. Evaluation Metrics

The evaluation of SS techniques, like many other tasks, relies on established evaluation metrics, crucial to provide objective and reliable measurements. This subsection presents the metrics leveraged for the analysis.

The following notions are adapted from [79, 86]. The formal notation used throughout the subsection and depicted in Table 2.2 is adapted as well from the same works.

### Pixel-wise Accuracy

The Pixel-wise Accuracy (PA) computes the ratio between the correctly predicted pixels and the total number of pixels. Formally:

| Symbol | Description |
|---|---|
| $N$ | Number of pixels |
| $C$ | Number of target classes |
| $t_n$ | Target class of pixel $n$ |
| $y_n$ | Predicted class of pixel $n$ |
| $\text{TP}_i$ | Number of pixels of target $i$ and predicted correctly |
| $\text{FP}_i$ | Number of pixels predicted wrongly with class $i$ |
| $\text{FN}_i$ | Number of pixels of target $i$ and predicted wrongly |

Table 2.2: Notation for SS evaluation metrics, adapted from [5].

$$\text{PA} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1} \left[ y_n = t_n \right] \tag{2.5}$$

This very simple metric reflects the per-pixel classification correctness of the model. It is strongly affected by class imbalance, since the formula does not provide any form of balancing. Hence, over-represented pixels will dominate the computation of the PA, while the accuracy of under-represented ones will have a more negligible weight. As a consequence, this metric does not faithfully capture the model's robustness in class-imbalanced scenarios. Regardless, the simplicity and straightforwardness of PA make it a widely used choice, typically used side-by-side with other, more expressive metrics.

### Mean Intersection over Union

The mean Intersection over Union (mIoU) [23] (also known as multi-class Jaccard Index) computes the ratio of intersection over the union between the prediction and the ground truths per class, then performs an averaging across the classes. Formally:

$$\text{mIoU} = \frac{1}{C} \sum_{i=1}^{C} \text{IoU}_i, \tag{2.6}$$

where the class-specific IoU (also known as Jaccard Index) is:

$$\text{IoU}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i}. \tag{2.7}$$

The mIoU is a widely adopted metric in SS, since it has some very desirable properties:

- By means of the macro-average, each class contributes equally to the overall com-

putation, so that it is not over-influenced by majority classes and under-influenced by minority classes.

- It is an interpretable measure of overlap that aligns well with human rationale.

On the other hand, depending on the context, it might deliver an under-estimated evaluation in cases of severe class imbalance, in which performance over severely under-represented classes might excessively lower the overall score.

### 2.1.5. Datasets

This subsection presents and describes the datasets involved in this work.

#### Pascal VOC 2012

Pascal Visual Object Classes 2012 (VOC2012) [23, 24] is a CV dataset providing images and ground truth annotations for five different tasks: classification, detection, semantic segmentation, action classification, and person layout. In this work, only the components related to SS will be considered.

This dataset contains a total of twenty classes, whose distribution is visualized in Figure 2.3, an unlabelled class, segmenting irrelevant objects and borders, and a background class, used to signal the absence of any of the main object classes.

The dataset comprises a total of 2,913 image-masks pairs, divided into a training set, containing 1,464 samples, and a validation set, containing 1,449 samples. Figure 2.4 displays a few of the available samples with predicted masks.

#### Microsoft COCO 2017

Microsoft Common Objects in Context 2017 (COCO2017) [48] is a CV dataset providing the visual annotation required for the tasks of object detection, panoptic segmentation, instance segmentation, keypoint detection, and image captioning. In this work, only the components related to SS (derived from instance segmentation) will be considered.

This dataset contains a total of eighty classes and a background class. The class instance distribution of the dataset is displayed in Figure 2.5, compared with VOC2012 [23, 24].

The dataset offers a total of 123,287 image-masks pairs, divided into a training set, containing 118,287 samples, and a validation set, containing 5000 samples. Figure 2.6 displays a few of the available samples.

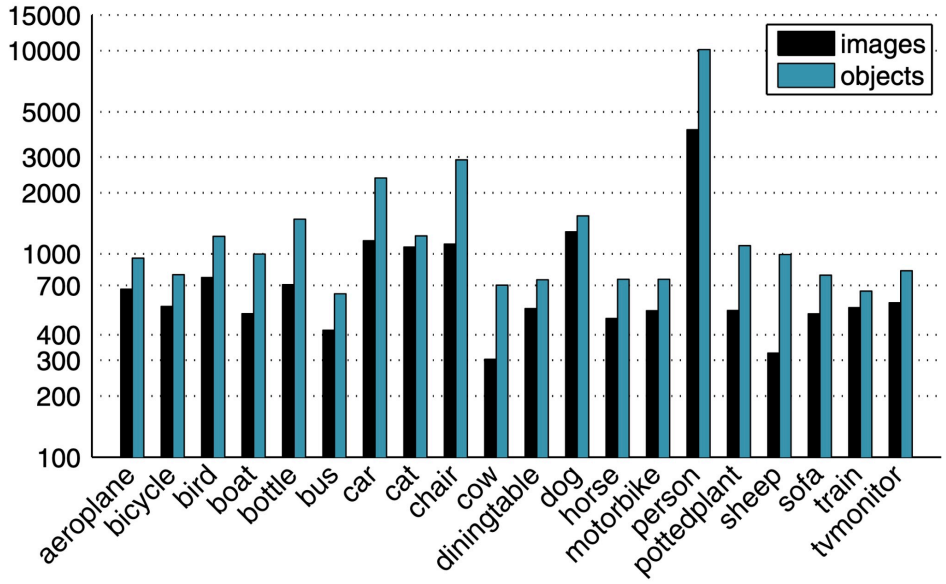



Figure 2.3: Histogram by class of the number of objects and images in VOC2012 containing at least one object of the corresponding class. It considers both training and validation images, and the vertical axis has a logarithmic scale. From [24].

## 2.2. Text Generation

This section provides information on the text generation concepts at the foundation of this work.

### 2.2.1. Language Models

This subsection briefly introduces the language modeling task and the related models. The following notions are adapted from the book [37].

Language Models (LMs) are models that predict upcoming words based on the preceding ones. They are inherently sequence-to-sequence models, since the language modelling task requires dealing with text pieces of variable size. The most successful language models are conditional and auto-regressive: they comprise a fixed vocabulary of words, referred to as tokens, predict a probability distribution over all the tokens of the vocabulary conditioned on the window of past words (hence their conditional nature), sample from this distribution, append the newly generated token to the window (hence their auto-regressive nature) and iterate this process to generate entire pieces of text.

Formally, the task of conditional language modelling, extensively studied by the field of NLP, is to model the probability distribution $P_V(w_{n+1})$ over a vocabulary $V$ for a token

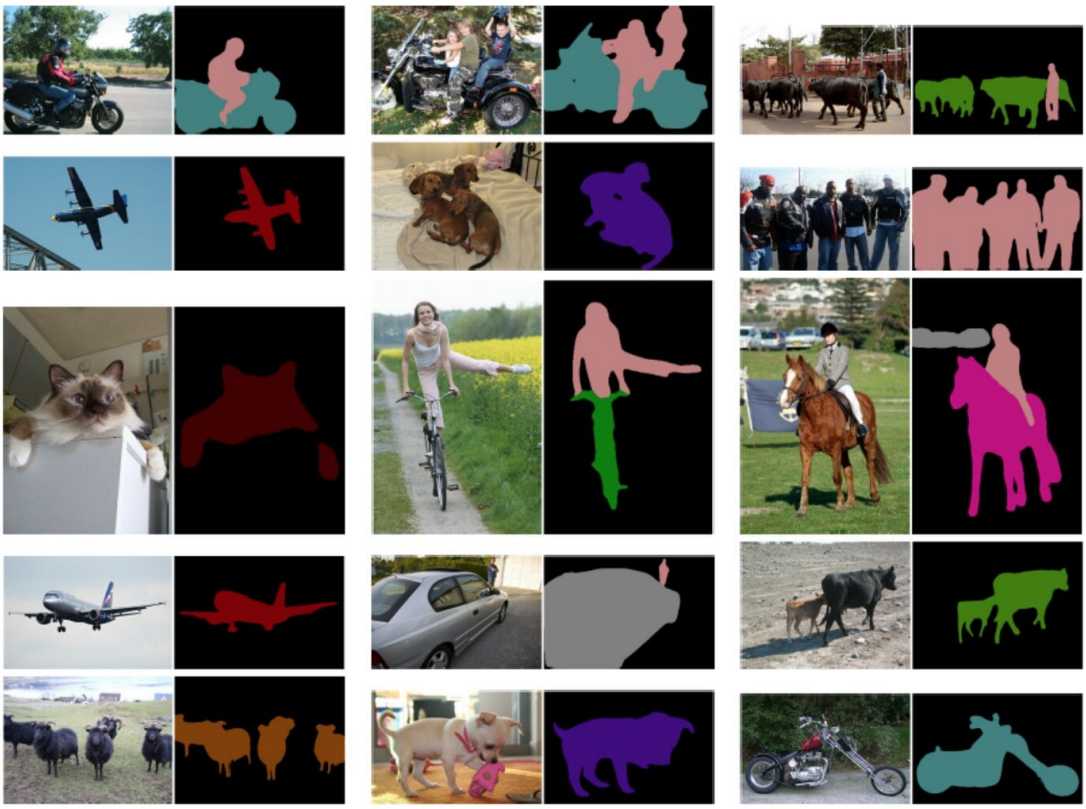


Figure 2.4: Examples of mask predictions in VOC2012 dataset. From [58].

at step $n+1$, based on the sequence of tokens from earlier steps. Formally:

$$P_V(w_{n+1}) = P_V(w_{n+1} \mid w_{<n+1}) = P_V(w_{n+1} \mid w_n, w_{n-1}, \ldots, w_2, w_1), \forall w \in V, \tag{2.8}$$

given $V$ of size $m_{\text{vocab}}$, and a context window of tokens, which is increasingly filled with sampled ones. Practically, for computational reasons, LMs are conditioned on a context window which can grow up to a fixed size $k$. Hence, their task becomes $P_V(w_{n+1}) = P_V(w_{n+1} \mid \{w_i\}_{n-k \leq i \leq n})$

Auto-regressive LMs decompose the text generation task into a next-token prediction objective, which allows them to be trained on large corpora of text to effectively shape the probability distribution they compute, which directly affects the correctness and diversity of the text they generate. Indeed, there are different ways to sample from the token distribution $P_V(w_{n+1})$ over the vocabulary $V$ at time-step $n$. The most frequently used ones are:

- Random sampling: $w_{n+1} \sim P_V(w_{n+1})$.
- Greedy decoding: $w_{n+1} = \text{argmax}_{w \in V} P_V(w_{n+1})$.

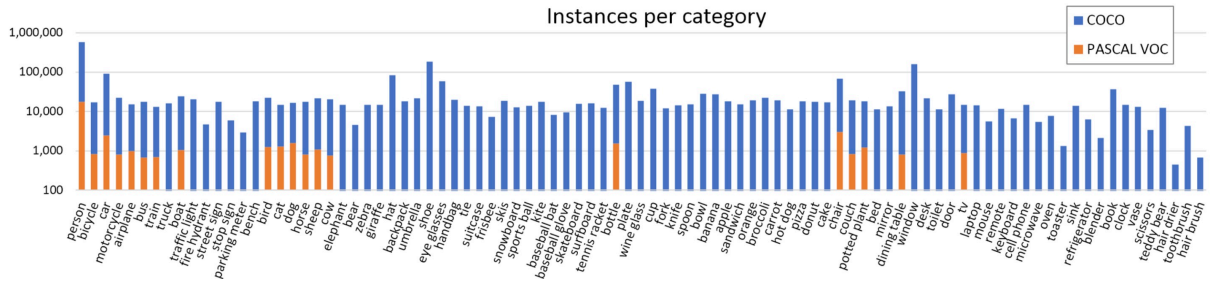



Figure 2.5: Histogram by class of the number of objects and images in COCO2017 containing at least one object of the corresponding class. It considers both training and validation images, and the vertical axis has a logarithmic scale. The visualization also provides a comparison with the VOC2012 dataset. From [48].

- Top-$k$ sampling: $w_{n+1} \sim P_{\tilde{V}}(w_{n+1})$ where $\tilde{V}$ only contains the most probable $k$ tokens (with re-normalised probability).
- Top-$p$ (nucleus) sampling: $w_{n+1} \sim P_{\tilde{V}}(w_{n+1})$ where $\tilde{V}$ only contains the most probable tokens whose probabilities are summed to $p$ (with re-normalised probability).

### 2.2.2. Word Embeddings

Modern LMs leverage word embeddings, which encode tokens as numerical vectors of fixed size, embedded into a shared vector space. This, as opposed to older methods leveraging sparse encoding techniques, allows for obtaining dense, meaningful word representations, which results in significant improvements in generalization capabilities and performance [3].

Word embeddings are derived through unsupervised pre-training stages, in which models learn to capture statistical patterns and linguistic regularities across large text corpora, enabling them to encode both semantic and syntactic properties of words based on their recurring contextual usage [3]. As vector representations, words are embedded into numerical spaces where algebraic operations, such as addition, subtraction, distance measures, and similarity measures, become meaningful [3]. Examples of this are shown in Figure 2.7.

In this context, vocabularies become collections of dense word vectors of fixed embedding dimensionality, whose information can be easily and effectively processed by models to tackle many kinds of tasks. In mathematical terms: $V = \{w_i \in \mathbb{R}^{d_{\text{embd}}}\}_{i=1}^{m_{\text{vocab}}}$, where $d_{\text{embd}}$ is the embedding dimensionality.

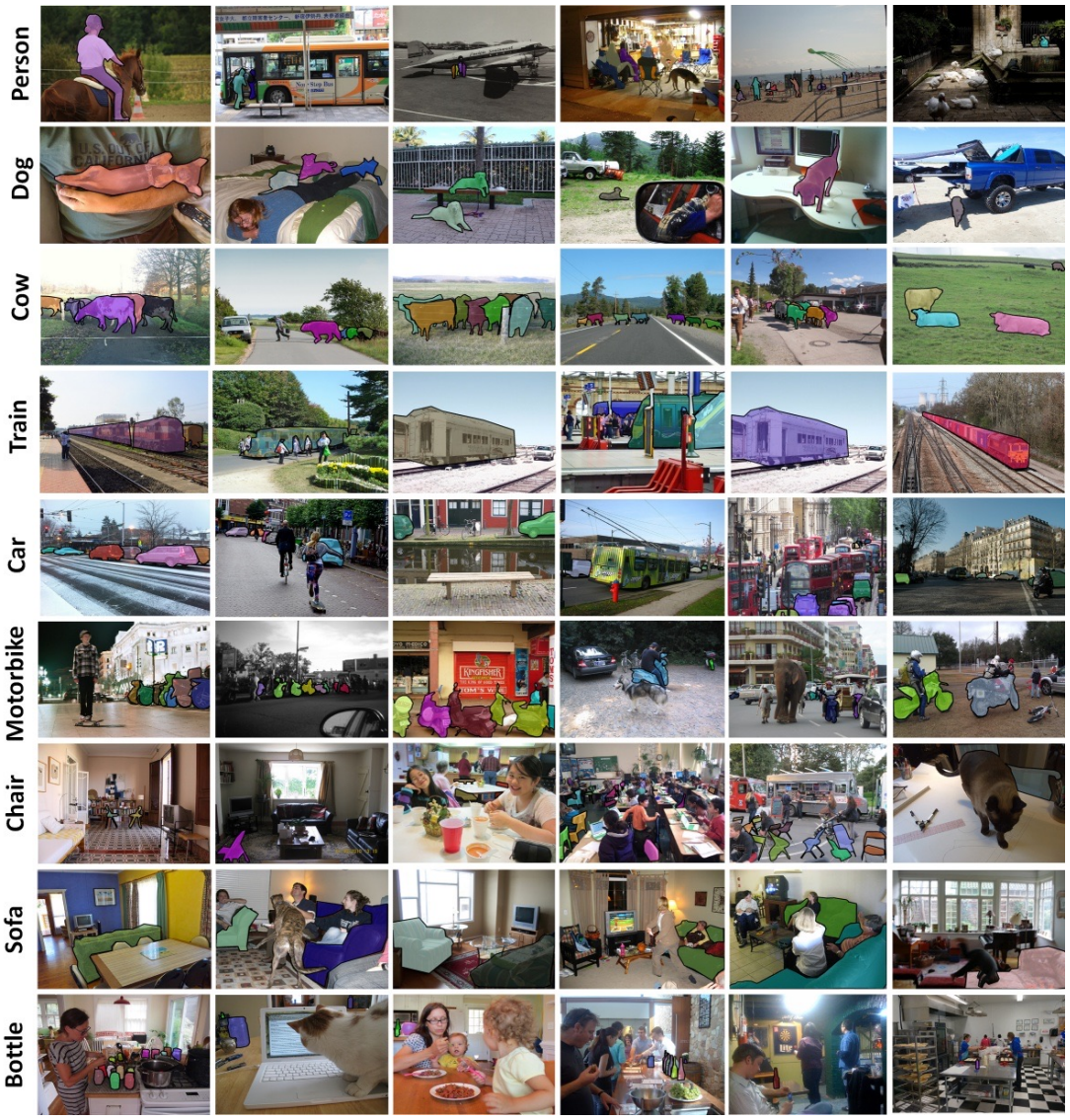
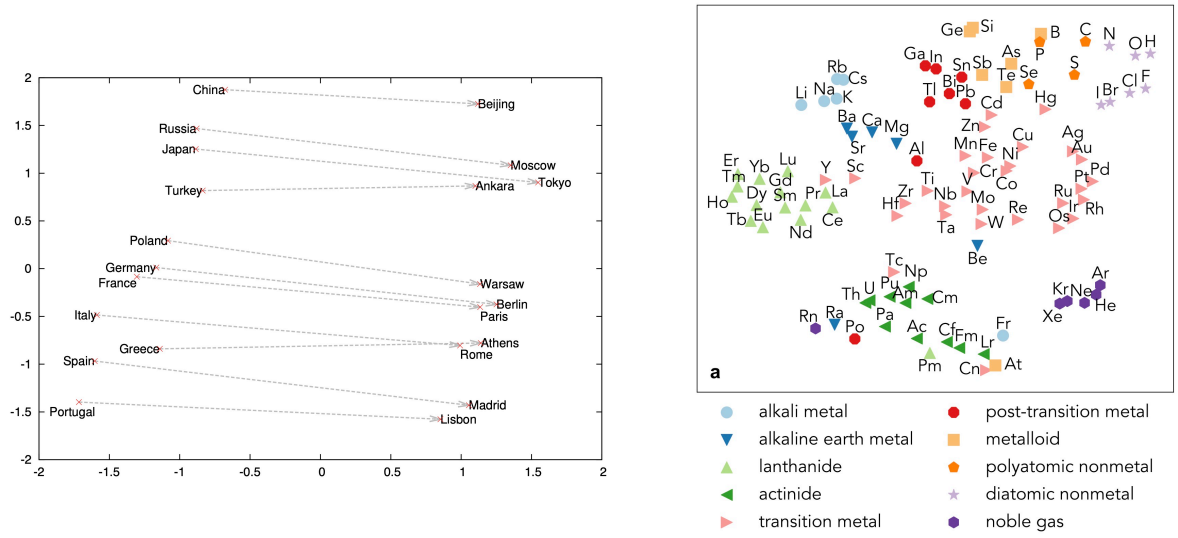



Figure 2.6: Samples of annotated images in the Microsoft COCO 2017 dataset. From [48].

### 2.2.3. Transformers and LLMs

As of current knowledge, empirical findings showed that only sufficiently large LMs achieved satisfactory results, revealing that language modelling performance is strongly scale-dependent, as a result of the complexity of the task.

All current state-of-the-art LMs implement the Transformer architecture [84], which has proven to scale effectively with the number of parameters, the depth of the network, as well as the number of training data points. With the rise of Transformers, model and dataset sizes continuously increased, leading to the definition of Large Language Models (LLMs), whose parameter count ranges from hundreds of millions to trillions. A clear illustration of this trend is shown in Figure 2.8, which depicts the scale-dependent loss




(a) (b)

Figure 2.7: Both figures show a two-dimensional projection of spaces learned by Word2Vec [55]. In (a), the space models the relationship between them without any supervised information about what a capital city means. From [56]. In (b), the space clusters together chemically similar elements, and the overall distribution exhibits a topology reminiscent of the periodic table itself. From [81].

curves of the GPT-3 LLM [11].

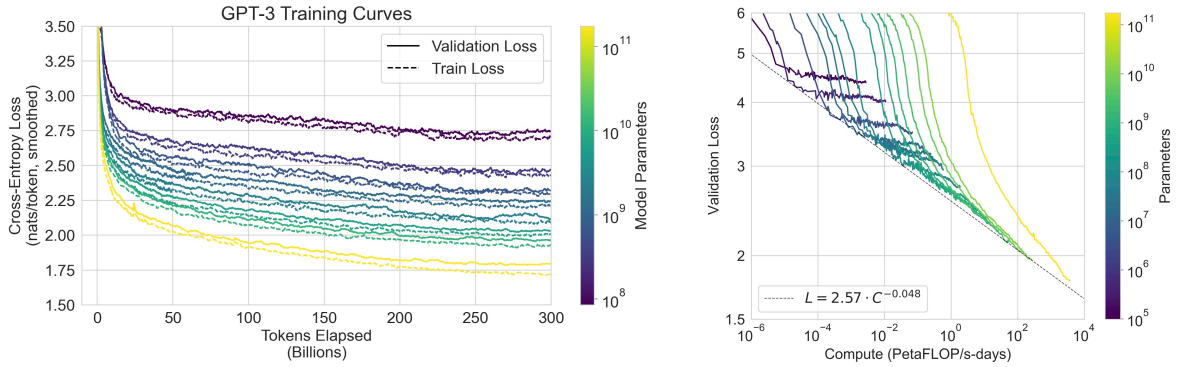
## The Transformer Architecture

The Transformer architecture [84] is fairly simple, and it presents a few key aspects:

- The reliance on embeddings, presented in Subsection 2.2.2 and, further on, in Subsection 2.3.1.
- A non-recurrent, stacked form, described in this subsubsection.
- A strong use of the attention mechanism, introduced in the following subsubsection.

The Transformer architecture is composed of an input projection layer, gathering the word embeddings of the tokens that populate the context window, a stack of $N$ transformer blocks, and an output projection layer, sharing the parameters of the input projection layer and mapping the transformed embeddings into the next token probability distribution.

The original architecture, shown in Figure 2.9, designed for the translation task, comprises encoder and decoder blocks, which enable the direct information flow from the encoded input text to the generated text. However, most modern implementations, tailored for




(a) (b)

Figure 2.8: The training and validation losses of the GPT-3 LLM exhibit consistent scaling with increasing parameter counts, dataset size (a), and computational resources (b). From [11].

text generation, are decoder-only, i.e., do not feature the encoder blocks and the layers connecting them to the decoder blocks.

The bulk of the computation happens in the stacked transformer blocks, with each one augmenting the residual stream with word vectors processed by means of scaled dot-product attention (which will be defined in Subsubsection 2.2.3) and a Multi-Layer Perceptron (MLP) [21, 84]. The output of each transformer block is fed as input to the next block.

Each transformer block, comprising multiple attention heads, extracts different kinds of meanings and patterns from tokens, and contributes to the enrichment of the residual path with information of diverse nature and value.

The operations performed in the Transformer block are not permutation-invariant. Therefore, a positional encoding, i.e., an additional vector encoding the word order information, is introduced to break the symmetry.

### The Attention Mechanism

The attention mechanism, once a complementary component in recurrent networks, now forms the central building block of Transformer models, allowing them to aggregate and process information from a varying number of elements efficiently and expressively [84].

The attention function featured in the Transformer blocks is the Scaled Dot-Product

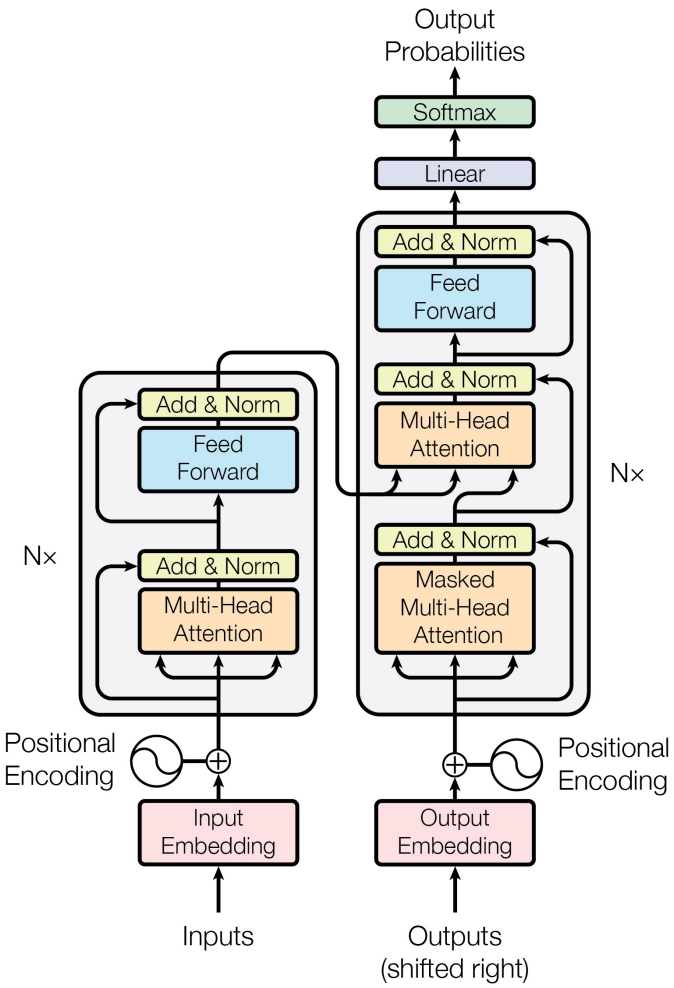



Figure 2.9: High-level view of the encoder-decoder Transformer architecture. The left half, displaying the encoder block, and the second Multi-Head Attention layer are not present in the decoder-only variant. The *masked* attribute of the attention indicates its auto-regressive nature. Image from [84].

Attention (SDPA), visualized in Figure 2.10a and defined as follows:

$$\text{SDPA}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.9}$$

where $Q, K, V \in \mathbb{R}^{n \times d_{\text{embd}}}$ are the query, key and value matrices, and the softmax function is applied row-wise, such that:

$$\text{softmax}(X)_{i,j} = \frac{\exp(x_{i,j})}{\sum_{k=1}^{\dots} \exp(x_{i,k})} \tag{2.10}$$

Each transformer block comprises a Multi-Head Attention (MHA), shown in Figure 2.10b,

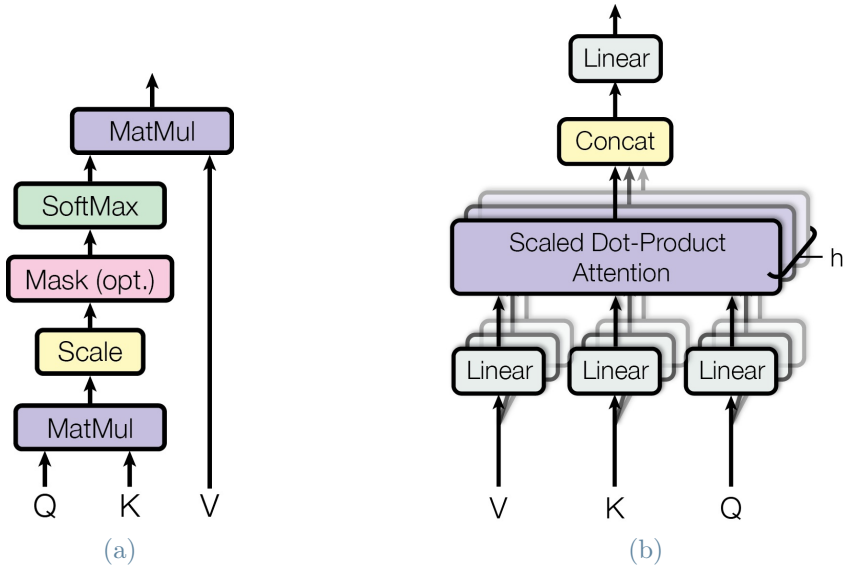



Figure 2.10: High-level visualization of (a) the sequence of operations included in the SDPA and (b) of the MHA. From [84].

in which each of the $h$ heads applies the SDPA on query, key and value matrices projected into a subspace of reduced dimensionality, and then maps them back into the original dimensionality $d_{\text{embd}}$:

$$\text{MHA}(Q, K, V) = \text{concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.11a}$$

$$\text{where head}_i = \text{SDPA}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.11b}$$

where $W_i^Q, W_i^K, \in \mathbb{R}^{d_{\text{embd}} \times d_k}$, $VW_i^V \in \mathbb{R}^{d_{\text{embd}} \times d_v}$, and $W_O \in \mathbb{R}^{d_{\text{embd}} \times d_{\text{embd}}}$ are projection matrices. Usually, $d_k = d_v = \frac{d_{\text{embd}}}{h}$ [84].

## 2.3. Vision-Language Models

Vision-Language Models (VLMs) are models at the conjunction between CV and NLP, and can reason on top of vision and text modalities. The main challenges faced in this context are providing transferable representations for data across different modalities and effectively fusing multi-modal data to achieve strong performance on downstream tasks [36].

This section presents and describes the current notions related to these challenges that have been crucial for the development of this work. It also introduces the field of Multi-Image Understanding as well as the Image Difference Captioning task.

### 2.3.1. Vision-Language Encoders

Vision-Language Encoders (VLEs) learn a unified representation space for images and text from large datasets of image-text pairs [97]. By encoding both modalities as numerical vectors in this shared space, the models can capture semantic relationships between them, so that similar concepts in images and text are mapped to nearby vectors. This alignment enables a range of cross-modal tasks, such as retrieving images based on a text query or performing open-vocabulary classification [67].

In recent years, models of this category have been trained by means of a family of scalable, effective, and flexible training objectives known as contrastive losses, which train models to match (positive) image-text pairs in the embedding space, making sure that unrelated (negative) image-text pairs are dissimilar [97]. This approach belongs to the broader field of contrastive learning.

A seminal work in this field has been developed by the authors of CLIP [67], who framed the objective as a multi-class classification problem and deployed a symmetric CE loss function, adapted for the vision-language context from the InfoNCE loss [61].

Given a batch of $N \times N$, CLIP loss aims to maximise the cosine similarity of the $N$ positive image-text embeddings, while minimising the cosine similarity of the $N^2 - N$ incorrect pairings. This batched computation is often visualized as in Figure 2.11

The authors of SigLIP [97], instead, framed the objective as a multi-label classification problem and adopted a sigmoid loss, disentangling the batch size from the loss and witnessing improved performance for both small and large batch sizes. Additionally, they proposed to further refine the image embeddings by means of an attention layer pooling the local image tokens together using the global image representation as key.

Despite the outstanding multi-modal capabilities of these models, which unlocked new state-of-the-art results on image-text retrieval and representation learning tasks, many consequent works argued that the very models struggle in compositional understanding, i.e., the ability to capture the compositional structure present in each modality and across the two modalities. This results in shallow understanding capabilities and spurious patterns [43, 80, 83, 96].

In particular, recent studies highlighted limitations on the fine-grained recognition capabilities: models tend to overly focus on coarse-grained categorical concepts, while overlooking attribute-level details and small features [8, 90, 91, 95, 100].

To address these limitations, several research works experimented with new architectures

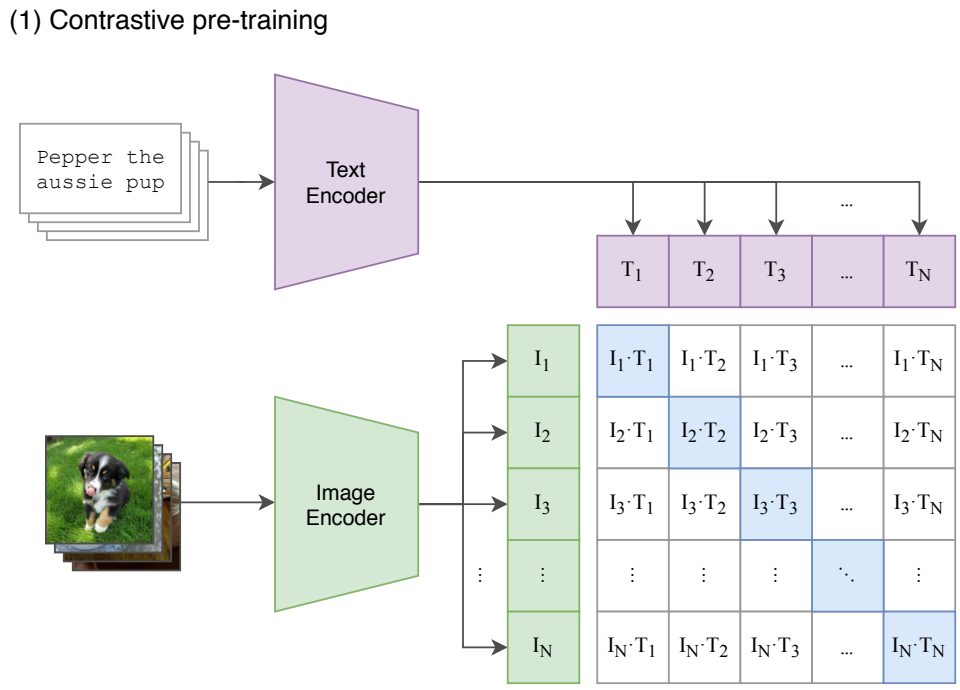



Figure 2.11: CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. Image from [67].

and training paradigms:

The authors of SimSeg [95] developed a training strategy to promote a locality-driven alignment of image and text features, improving the quality of the interaction between entities at different levels.

The authors of FG-CLIP [91] trained CLIP on a custom dataset of image crops and corresponding detailed, context-rich captions, augmented with a large number of hard fine-grained negative examples, to promote the fine-grained discriminative capabilities of the model.

The authors of FLAIR [90] proposed to provide the training contrastive objective with multiple positive sub-captions per image, each describing a localized visual portion. Additionally, they introduced a text-conditioned attention pooling layer to produce text-specific image representations. The model is trained by aligning the text features with the text-conditioned image representations as well as the standard, unconditioned features. An overview of the pipeline is shown in Figure 2.12, while a comparison of the architecture with CLIP and SigLIP is presented in Figure 2.13.

The text-conditioned attention pooling layer, which is a single MHA layer adopting the global text embedding as key and the local image patch embeddings as queries and values,

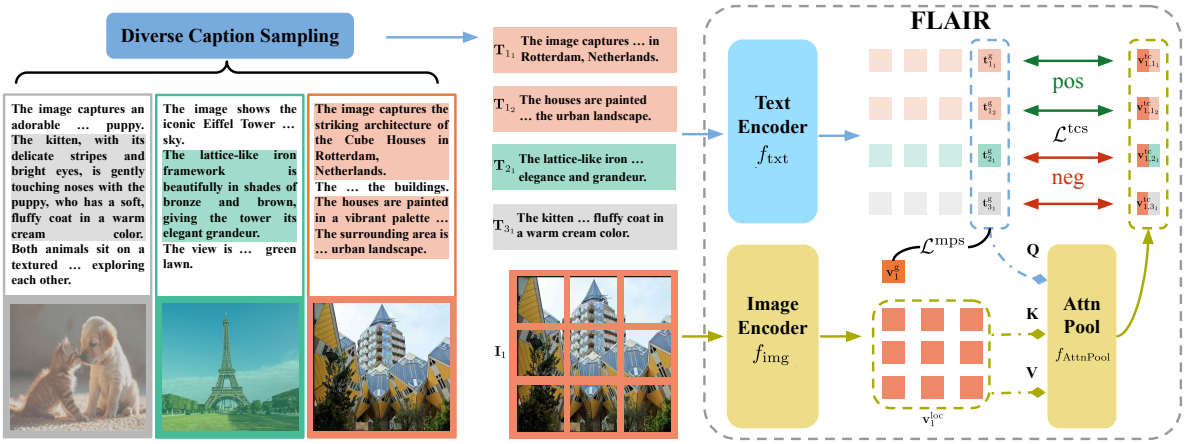



Figure 2.12: The FLAIR method samples a set of diverse positive and negative captions for an image. The text and image encoders then produce global text and image tokens, along with local image tokens. Using these global text representations, an attention pooling module creates fine-grained, text-conditioned image representations. A text-conditioned loss function aligns the text with these fine-grained image features, while a multi-positive loss function further improves the global alignment between the image and its captions. Image from [90].

pools the local image features into a global one conditioned on the text. The authors show that the attention maps at the heart of this computation can be effectively used as visualization maps for fine-grained visual grounding, as shown in Figure 2.14.

### 2.3.2. MLLMs

Multi-modal Large Language Models (MLLMs) are an extension of the LLMs that can handle multiple types of input beyond plain text. While a standard LLM generates text based solely on textual prompts, an MLLM can generate text conditioned on additional modalities, such as images, audio, and video. MLLMs can be seen as LLMs augmented with the ability to process and reason about non-textual information, while still leveraging their core language modeling capabilities.

Here, we will focus on their vision-based reasoning abilities, which enable a range of new applications. Vision-enabled MLLMs can tackle tasks that integrate visual understanding with language, such as image captioning (generating descriptive text for an image) and Visual Question Answering (VQA) (answering questions about an image) [50]. For instance, given a photograph of a city street, an MLLM could generate a caption like "A busy street with people walking and cars parked along the sidewalk," or respond to a question such as "How many people are visible in the scene?"

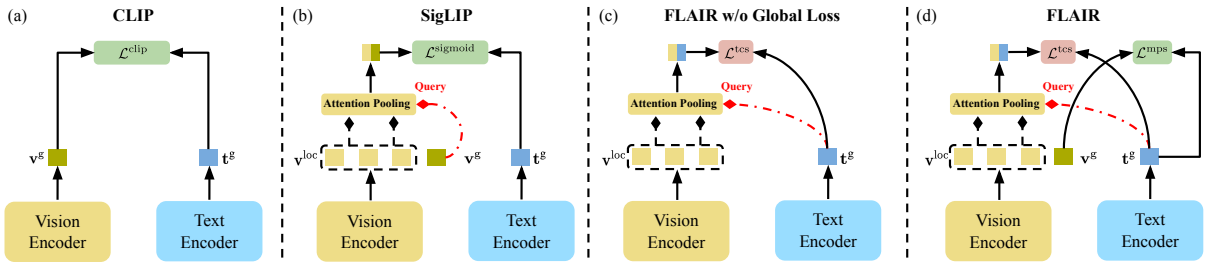



Figure 2.13: CLIP aligns global image and text features directly. SigLIP improves this by using a learnable global image token to pool local image features through cross-attention. FLAIR instead uses the text representation itself to guide attention pooling, making the visual features more language-aware. Finally, the method is further enhanced by adding an extra loss that strengthens global image-text alignment. Image from [90].

In contrast to text-only LLMs, image-grounded MLLMs can be conditioned on images, since their architectures, augmented with the vision encoders coming from VLEs (described in Subsection 2.3.1), allow local visual tokens coming from images to be included in the context window.

To generate these visual tokens, images input to MLLMs are typically first divided into a grid of smaller, fixed-size patches (typically, $16 \times 16$ or $32 \times 32$ pixels). Each patch is then flattened into a 1D vector through a small convolutional network that maps it into a latent embedding space of the same dimensionality as the text token embeddings. This process effectively converts each visual patch into a visual token, analogous to a word token in text, often augmented with positional embeddings to retain spatial structure. The resulting sequence of tokens, both text and visual, can be fed into a transformer-based MLLM, enabling cross-modal attention where the model can reason jointly over textual and visual context. An example of this is illustrated in Figure 2.15.

### 2.3.3. Multi-Image Understanding

To explore the applications of MLLMs discussed in this work, it is helpful to examine the field of Multi-Image Understanding (MIU), which assesses the models' capabilities across multiple images.

Despite the satisfactory applications of vision-grounded MLLMs, several significant research works proposed ad-hoc benchmarks to assess the MIU capabilities of MLLMs, which highlighted the fundamental inadequacy of these models when dealing with multiple images:

MIBench [51] showed that MLLMs tend to exhibit confusion when dealing with multiple

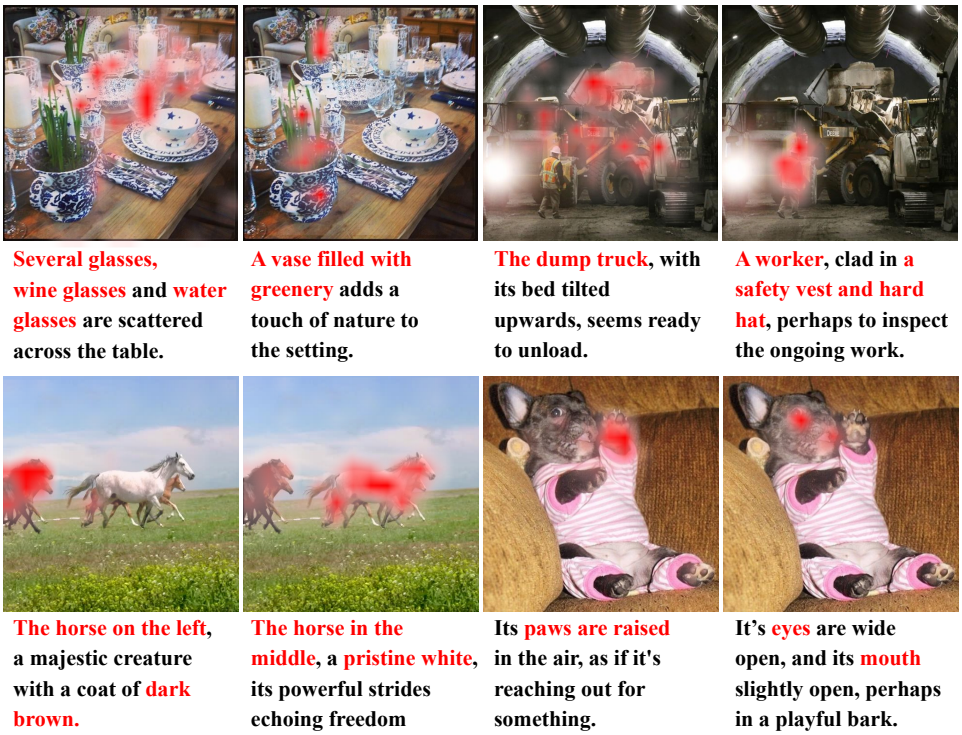



Figure 2.14: Visualization of attention maps in the attention pooling layer. Regions of high attention are highlighted in red. From [90].

images, witness limited improvements or even a deterioration in performance when increasing the number of demonstrations in few-shot learning, deliver insufficient results in fine-grained perception tasks, and, ultimately, have poor reasoning abilities across multiple images.

MIRB [99] revealed that eliciting reasoning capabilities of MLLMs with Chain-of-Thought prompting (which will be described in Subsection 2.4.1) results in minor improvements or sometimes even decreases in performance, highlighting the challenges in multi-image complex reasoning.

MMIU [54] underlined that MLLMs excel at capturing semantic content in high-level tasks, but struggle in comprehending temporal and spatial multi-image relationships, and have limited generalization abilities when facing unseen tasks.

MuirBench [85] highlighted that models are likely to hallucinate to unanswerable questions rather than abstaining, and state-of-the-art performance is significantly below the human baseline. Some qualitative results on this benchmark are presented in Figure 2.16.

ReMI [38] stated that MLLMs, despite significantly outperforming naive baselines, remain far behind human performance, and that different models perform well on different

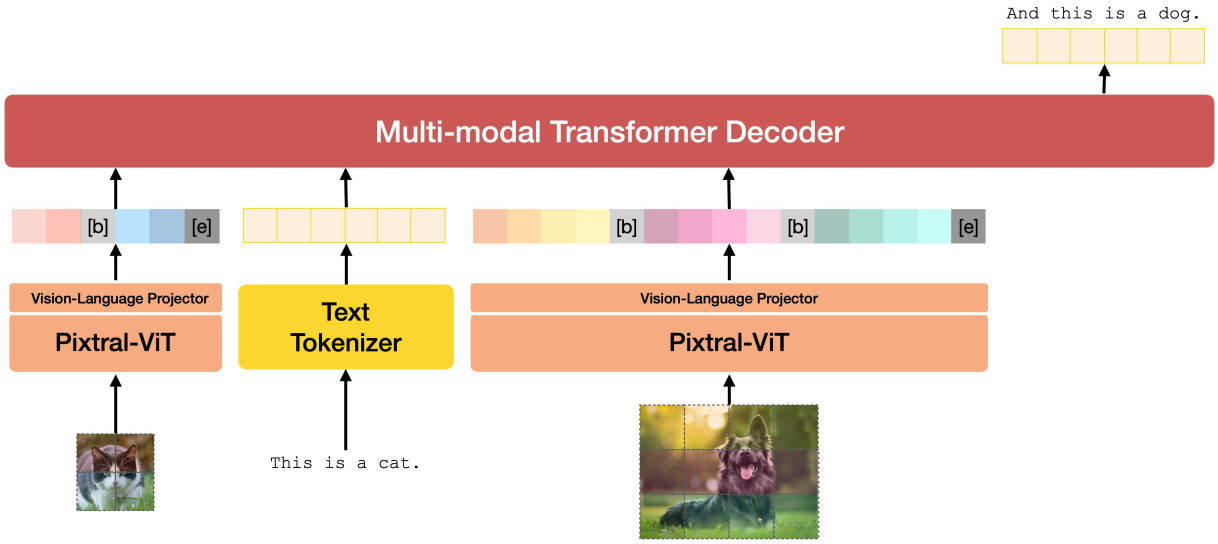



Figure 2.15: The Pixtral 12B MLLM [2] comprises a vision encoder that tokenises an arbitrary number of images in the context window, conditioning the multimodal decoder to predict the next text token. Image from [2].

tasks, suggesting that models may have different capabilities and limitations. Also, the authors observed that the most dominant cause of failures in their benchmarks is related to reasoning errors, followed closely by image reading errors.

Additionally, [51, 85] works stressed that models accepting multi-image inputs, even with fewer parameters, perform better than single-image input MLLMs, reasonably showing that generalizing from single-image training to multi-image inference is non-trivial, and that pre-training on interleaved image-text data and instruction tuning on multi-image data notably promote MIU capabilities.

On the other hand, the authors of [54] pointed out that, in some cases, single-image MLLMs can outperform ones trained with multiple images, effectively transferring single-image capabilities onto the multi-image context. In particular, models supporting high-resolution visual inputs can deliver competitive results when receiving multiple images concatenated into a single image.

Lastly, [38, 99] witnessed that models tend to perform better when fed with separated images, rather than concatenated, while [54] underlined that single-image MLLMs can benefit from concatenated images, arguably because of the better adherence with the training procedure.

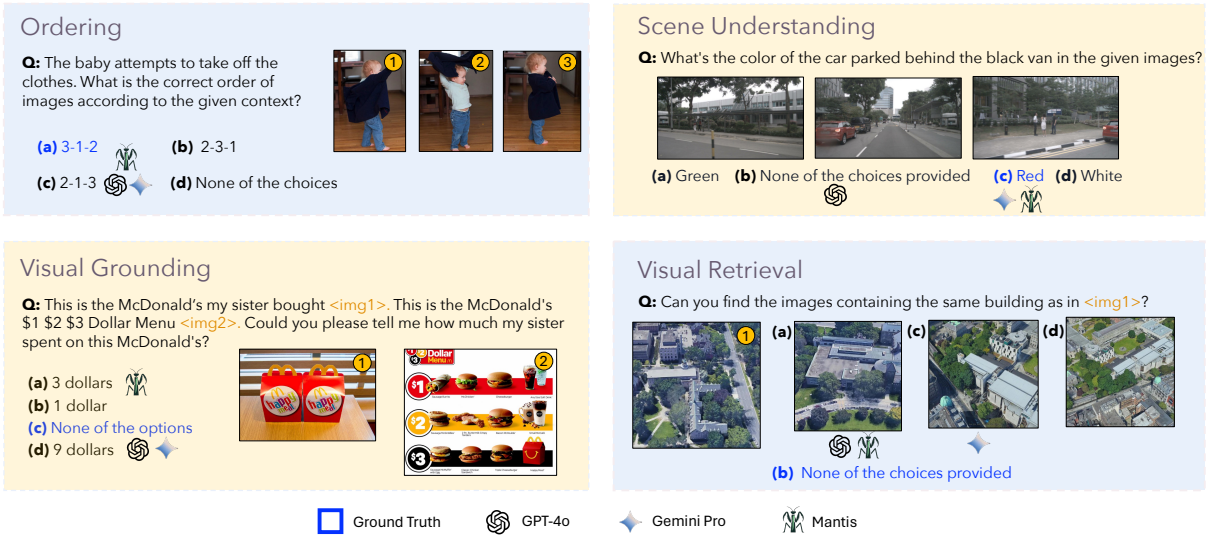



Figure 2.16: The tested MLLMs, identified by their symbol, tend to exhibit confusion in many fundamental MIU tasks within MuirBench [85]. From [85].

### 2.3.4. Image Difference Captioning

In this work, the process of generating natural language descriptions of the differences between images plays a critical role. This task, belonging to the broader MIU field (presented in Subsection 2.3.3), is known as Image Difference Captioning (IDC), which aims to describe the visual differences between two similar images with natural language (e.g., as in Figure 2.17). It is considered to be a challenging task since it requires fine-grained and compositional visual comprehension. Differently from the general image captioning task that describes a single image, IDC needs to perceive the fine-grained contents of two similar images to identify the differences, which may be of diverse types and magnitudes [94].

In recent years, many research works have proposed models, architectures, and training paradigms to tackle this task. Unfortunately, to the best of my knowledge, most ad-hoc IDC models are closely tailored to very specific domains such as satellite imagery (remote sensing, e.g., Chg2Cap [12], TIDSNet [44], SEIFNet [33]) or zoology (e.g., [94]). Therefore, these models lack the generality required to be competent in segmentation masks differencing, due to the limited, specific range of representations learned by these models, and the restrictedness of the vocabulary.

Some research works aimed at enhancing IDC networks' generality through the use of self-supervised training paradigms, multi-task learning (e.g., [94]), and synthetic datasets (e.g, CLEVR-Change [63]), but still, the amount of knowledge gathered this way is far

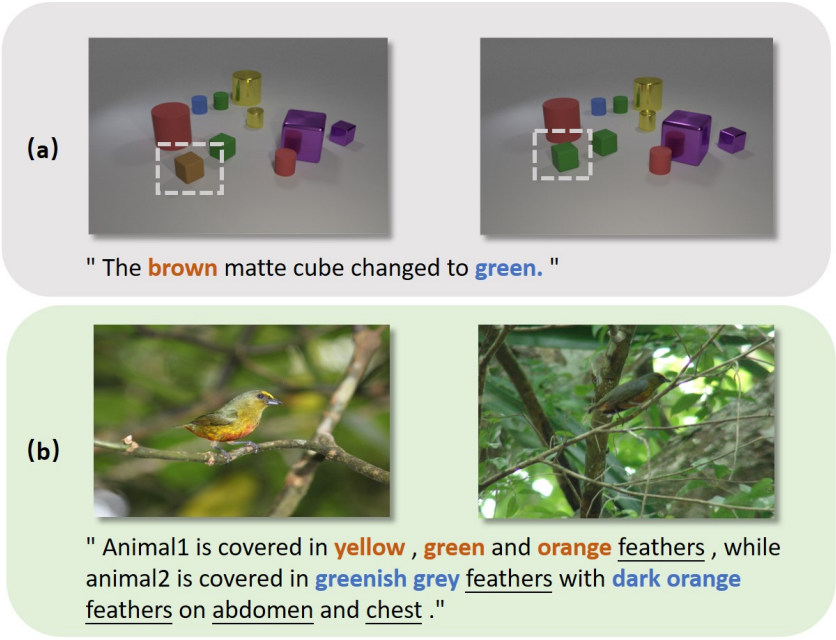



Figure 2.17: Image Difference Captioning examples, from [94].

from the extensive range of notions required to reason over segmentation masks.

Recently, research studies have explored how to adapt foundation MLLMs and VLMs for the IDC task (e.g., CLIP4IDC [25]), but this remains an under-researched field, and it has not been kept updated with the latest innovations regarding MLLMs. As a result, currently available general solutions still struggle to deliver satisfactory performance for the intended application.

## 2.4. Prompt Engineering

LLMs, by means of a large unsupervised pre-training stage, learn at training-time a broad set of skills and pattern recognition abilities, which they can use during inference to recognize and perform the desired task, based on the data it has been conditioned on. This rapid inference-time adaptation is referred to as In-Context Learning (ICL) [11].

As a consequence, through the careful selection of task-specific natural language instructions, specifications, or demonstrations, referred to as prompts, it is possible to steer the general, vast knowledge of LLMs to achieve a reasonable degree of competence over many tasks, without the need for additional training or fine-tuning. [11, 72].

### 2.4.1. Prompting Techniques

Prompting techniques provide systematic ways to design and structure prompts to maximise the effectiveness of LLMs in solving a target task. These techniques range from simple instruction-based queries to more elaborate strategies that incorporate demonstrations and reasoning steps.

This subsection illustrates the prompting techniques that had a primary role in this work.

#### Zero-Shot and Few-Shot Prompting

In zero-shot prompting, instead of extensive task-specific training procedures, users can rely on carefully crafted prompts that guide the model towards novel tasks without providing the model with labelled input-target pairs [11, 72].

In one-shot and few-shot prompting, the model is provided with examples encoded as input-target pairs to induce an understanding of the given task. The selection and arrangement of the examples can significantly influence model behaviour and biases, potentially improving performance on complex tasks [11, 72]. An overview of zero-shot, one-shot, and few-shot prompting is presented in Figure 2.18.

Empirically, in many tasks, model performance improves with the addition of a natural language task instruction and with the number of examples in the model's context. Few-shot learning also improves dramatically with model size: larger models can make increasingly effective use of in-context information, including both task examples and natural language task descriptions [11]. An example of this is displayed in Figure 2.19.

#### Chain-of-Thought Prompting

When solving complicated reasoning tasks, it is helpful to decompose the problem into intermediate, simpler steps and solve each of them before giving the final answer.

Chain-of-Thought (CoT) prompting consists of augmenting each shot of a one/few-shot prompt with a chain of thought, which hints to the model how to structure the reasoning required to arrive at the solution [72, 89]. A demonstration for this can be viewed in Figure 2.20.

Empirically, CoT prompting helps to achieve better performance on many complex reasoning problems and provides an interpretable window into the behaviour of the model [89].

However, manual creation of high-quality CoT examples can be both time-consuming and suboptimal. Indeed, with zero-shot CoT, it is possible to elicit reasoning traces from

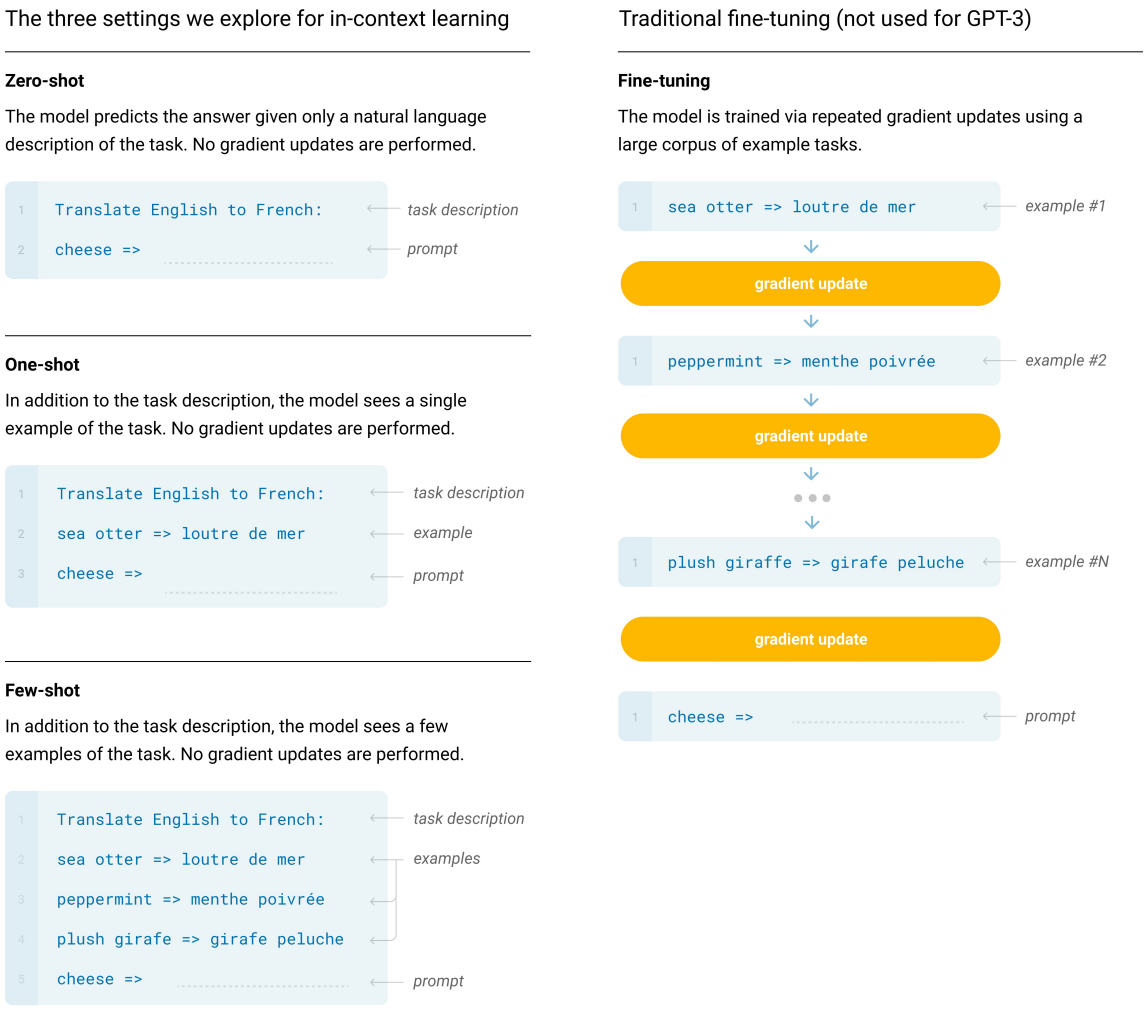



Figure 2.18: Zero-shot, one-shot, and few-shot learning, differently from traditional fine-tuning, allow to improve the performance of the LLM over new tasks only through forward passes, without expensive weight updates. Image from [11].

the LLM just by instructing it to generate reasoning chains without explicitly providing examples for the CoT [41, 72].

E.g., [41, 93] showed that simply prepending expressions like "let's think step-by-step" or "let's solve the problem" to the model's response improves the correctness of the model on mathematical tasks.

## Prompting MLLMs

With MLLMs able to process textual, audio, and visual inputs, the complexity of the multi-modal tasks has increased as well, and questions remain about how best to structure

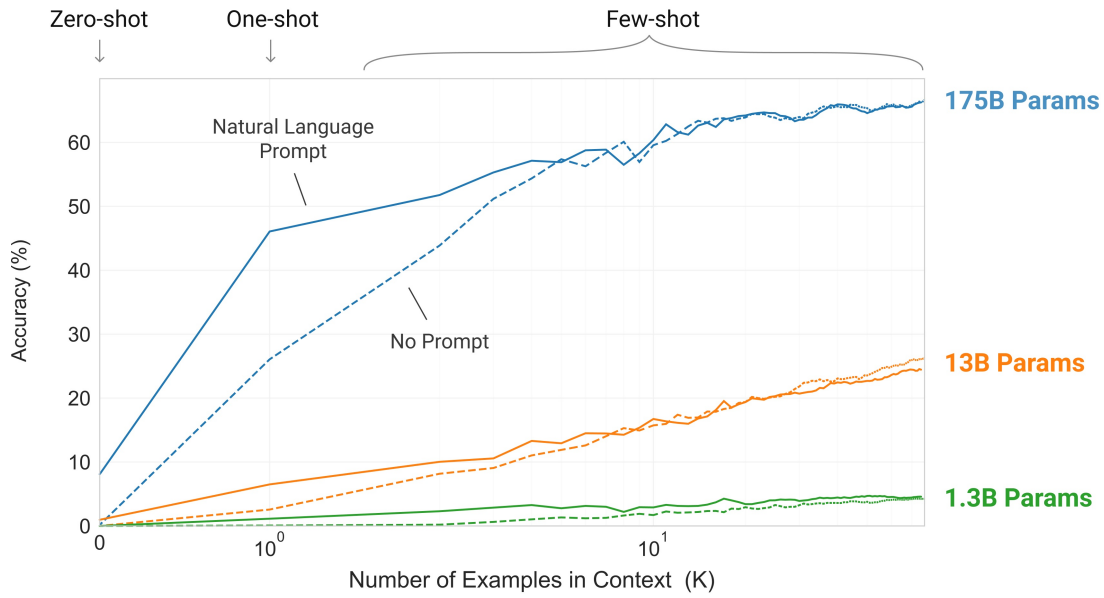



Figure 2.19: GPT-3 in-context learning performance on Symbol Insertion task. The accuracy curves demonstrate improved ability to learn a task from contextual information, especially as the model size grows. From [11].

the prompts to elicit optimal reasoning in such contexts [88].

Indeed, identifying the right prompt is a non-trivial task, which depends on many variables that can make a huge difference in performance. Also, multi-modal prompt engineering greatly benefits from having task-specific knowledge and knowing the model's underlying multi-modal mechanisms [102].

Starting from the way multiple images are arranged in the context, recalling what has already been said in Subsection 2.3.3, MLLMs trained to be multi-image usually benefit from receiving separated images, while it is not clear whether single-image MLLMs perform better with concatenated images or with separated images. Since each model may behave differently, their performance must be validated individually [38, 54, 99].

Furthermore, regarding the placement of elements within a prompt, recent findings demonstrate that both MLLMs exhibit a sensitivity to the sequential position of text and images. This positional bias is largely attributed to the operational mechanics of causal attention and relative positional encoding inherent in most MLLMs, which arguably is an artefact of their pre-training phase. This susceptibility to ordering is not confined to modalities but also applies to the sequence of instructions, which can substantially alter an LLM's performance. Consequently, regardless of the diverse ways in which LLMs can integrate different modalities, the ordering in which they are presented in the prompt can have a sig-

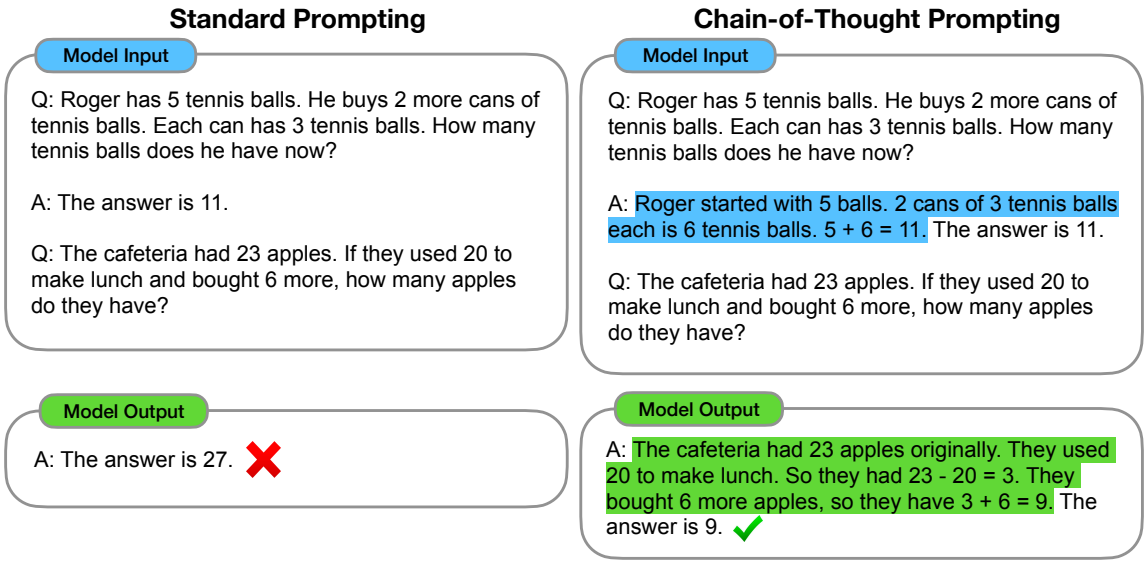



Figure 2.20: CoT prompting enables LLMs to tackle complex arithmetic, common-sense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted. From [89].

nificant impact on the model's behaviour. Therefore, prompt design and sequencing can still affect performance by influencing how multimodal information is integrated [87, 88].

In the context of a multi-image prompting, the authors of [88] have shown that different models show different performance trends when facing image-first, text-first, and image-text interleaving, depending on the multimodal fusion techniques and on the training data. The nature of the dataset plays a role in this, since datasets requiring reasoning traces and step-by-step problem-solving, on average, tend to favour prompts that interleave images with text, while other datasets favour images present before or after the text. Additionally, the optimal sequencing also depends on the specific structure of the individual question and of the task, and hence, is context-dependent. As a result, finding the optimal sequencing is not a one-size-fits-all approach, but depends heavily on the structural and logical flow required by the task.

Moving on from multi-image formatting to the multi-image reasoning, the authors of [99], recalling Subsection 2.3.3, acknowledged that applying zero-shot CoT, presented in Subsection 2.4.1, to tasks demanding complex reasoning, such as visual analogies or comparison, results in modest or even negative performance gains over the standard prompting (Figure 2.21).

To tackle this issue, [98] proposed Contrastive Chain-of-Thought, a prompting strategy that prompts the MLLM to discern and articulate the similarities and differences among

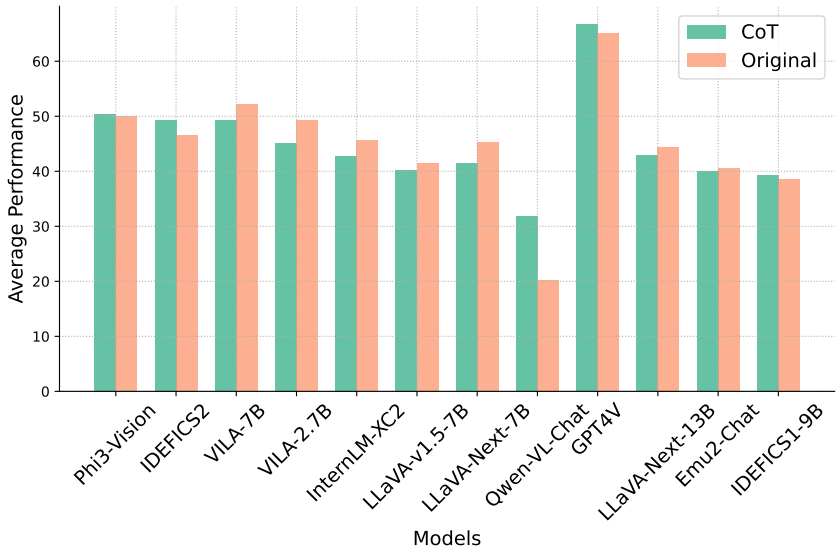



Figure 2.21: Among the MLLMs tested in [99], adopting zero-shot CoT prompting exhibits inconsistent effects. From [99].

various inputs, laying the groundwork for answering detailed, multi-image-based questions. This method aims to sharpen the models' focus, particularly on the distinctions between inputs, effectively facilitating causal reasoning and ensuring comprehensive capture of nuanced, question-relevant information.

### 2.4.2. LLM-as-a-Judge

As LLMs continue to evolve, evaluating their capacity on open-ended tasks becomes an increasingly pivotal matter. While manual evaluation is straightforward, it is expensive and can suffer from subjective inconsistency. Established evaluation metrics such as perplexity [35], BLEU [62], and ROUGE [46] are often inadequate in providing a semantic evaluation of the outputs. Also, the assumption of unique ground-truth reference responses is frequently invalid in many open-ended scenarios [32].

The improving capabilities of LLMs have created a new paradigm to face this problem: deploying LLMs as a judge to assess other LLMs (LLM-as-a-Judge). Indeed, given their inherent ability in reasoning with free-form text, LLMs have the potential to effectively evaluate how accurately LLMs adhere to open-ended instructions and meet user expectations [32].

Typically, an LLM is prompted with both the original instruction and the candidate response, along with evaluation criteria, such as relevance, correctness, completeness, and fluency. The model then produces a score or qualitative judgment reflecting how

well the response meets the expected standards. This can be done either through direct scoring, ranking multiple responses, or generating detailed feedback. By doing so, LLMs can provide a more semantic, flexible, and consistent evaluation than traditional metrics, especially in tasks with multiple valid answers or subjective nuances.

The effectiveness of LLM-as-a-Judge is highly dependent on the design of evaluation prompts, requiring careful prompt engineering to align automated assessments with human judgments [32]. Recent studies have shown that LLMs can achieve a high agreement rate with humans and provide nuanced, granular, and explainable evaluations that go beyond traditional metrics. However, ensuring consistent agreement with human evaluators remains a central challenge, highlighting the need for prompt designs that reliably capture human-like judgment while maintaining the scalability and robustness of automated evaluation [32].

## 2.5. Language Supervision

The main subject of this work is the supervision of the training process by means of natural language text. This research direction, although under-researched, has produced a few noteworthy results.

To my knowledge, one of the earliest works [19] on the subject proposed to leverage extra language supervision in the VQA task [4], by modulating Batch Normalisation (BN) layers [34] with text features (Figure 2.22a).

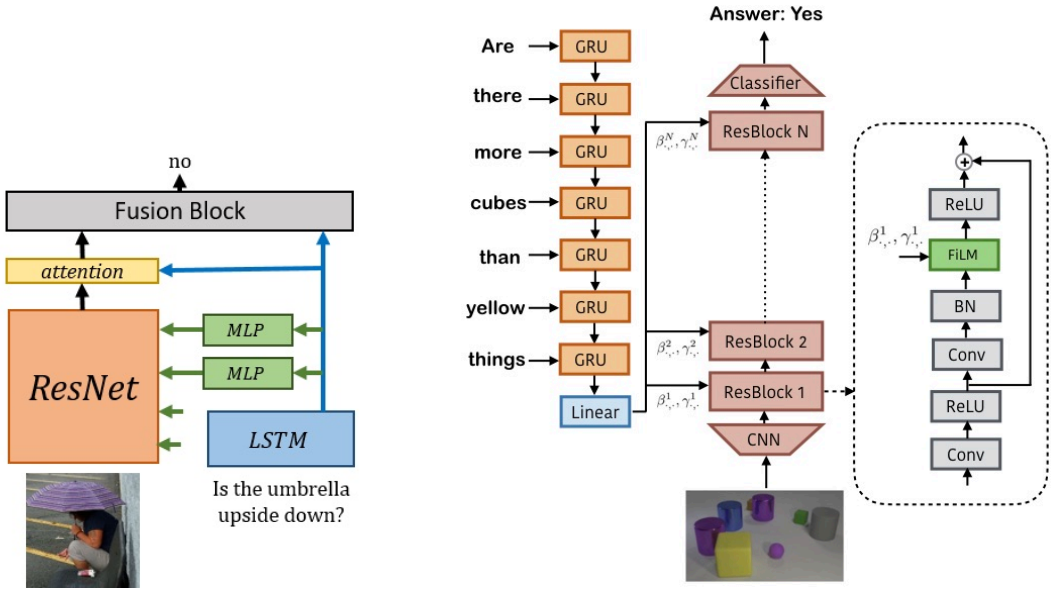
The text conditioning starts from a Long Short-Term Memory (LSTM) [30] with pre-trained word embeddings, which receives input visual questions and processes them, producing a sequence of hidden states. Then, the final hidden state of the LSTM is fed to an MLP, which projects them into parameter vectors that are used to shift the BN layers' parameters:

$$\begin{cases} \hat{\beta}_c = \beta_c + \text{MLP}(e_q)_\beta & \text{(2.12a)} \\ \hat{\gamma}_c = \gamma_c + \text{MLP}(e_q)_\gamma & \text{(2.12b)} \end{cases}$$

where $e_q$ is the VQA input question embedding resulting from the LSTM computation.

This form of conditioning is applied to every BN layer of the network, but the MLP is shared among them. This way, the additional trainable parameter count is reduced to that of a single MLP.

In the same VQA context, [64] explored how to apply a similar form of modulation by conditioning the feature representations through a linear mapping (FiLM block) defined




(a) (b)

Figure 2.22: In both (a) [19] and (b) [64], the text features, encoded by a recurrent model, directly condition a subset of the vision encoder layers: in (a) the BN layers, while in (b) the FiLM layers, which apply affine transformations. Images respectively from [19] and [64].

by a set of parameters learned starting from the input text (Figure 2.22b).

Here, the input is fed a Gated Recurrent Unit (GRU) [15] with pre-trained word embeddings, which outputs a final hidden layer embedding the text features, analogously to [19]. The text features are linearly projected into a multiplication vector and bias vector for each FiLM block, with each linear projection defined by different weights.

At this point, the learned vectors are used to define an affine transformation of the intermediate visual encoder's feature volumes:

$$\text{FiLM}(\mathbf{F}_{i,c}^{(n)} \,|\, \gamma_{i,c}^{(n)}, \beta_{i,c}^{(n)}) = \gamma_{i,c}^{(n)} \mathbf{F}_{i,c}^{(n)} + \beta_{i,c}^{(n)}, \tag{2.13}$$

referred to the $n^{\text{th}}$ FiLM block, the $i^{\text{th}}$ input sample and the $c^{\text{th}}$ 2D feature map.

The learned parameters are different for each block and for each feature map, but they are agnostic to spatial locations. This way, the number of trainable parameters is limited to $2 \times N \times C$ per FiLM block.

Despite the insightfulness of these two works, I argue that they are better seen as an advancement in multimodal fusion rather than in language supervision, since the text

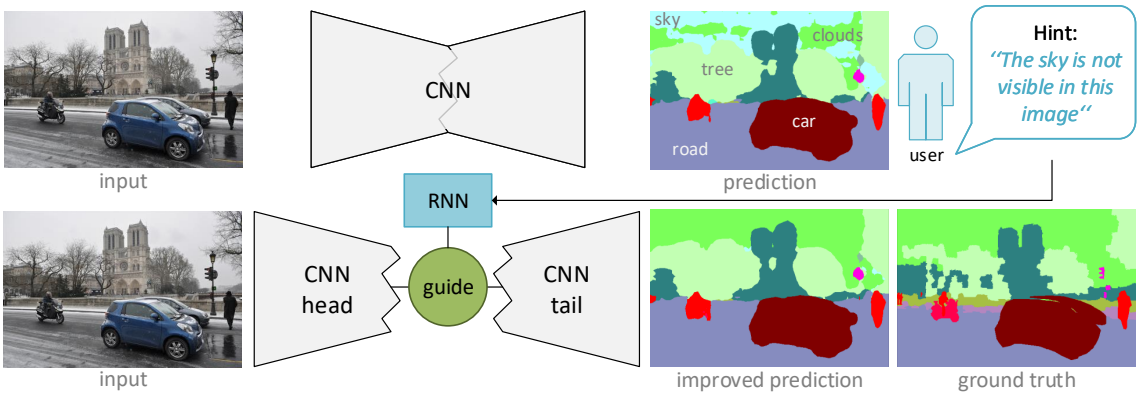



Figure 2.23: In [71], the text features, obtained by encoding a hint with a GRU, condition guiding blocks, which apply linear projections. The conditioning happens at inference time for a single image at a time. Image from [71].

used for extra supervision is the same one that is fed as input for the VQA task, meaning no new modality is introduced. Also, the contribution of the text in the pipelines is confined to a restricted number of layers, which have a limited role in the representations learned by the network. Therefore, the effects resulting from a wider and deeper usage of text supervision remain underexplored.

Later on, [71] investigated how to guide a trained, frozen neural network with natural language hints at inference time (Figure 2.23).

The authors proposed to insert a layer that is trained to modify the network's activations either through a recurrent model that translates natural language feedback to interaction weights. The user receives a first predicted mask, inputs a text query, and the network replies with an updated prediction. Hence, the user's input is optional; the network does not depend explicitly on human interaction but can be adjusted by it.

The integral piece of their work is the guiding block, which operates on a specific intermediate feature representation $h(x) = A \in \mathbb{R}^{H \times W \times C}$, scaling and shifting the activations by means of learned weight and bias vectors:

$$A'_{h,w,c} = (1 + \alpha_h + \beta_w + \gamma_c^{(s)})A_c + \gamma_c^{(b)}, \tag{2.14}$$

where $\alpha \in \mathbb{R}^H$, $\beta \in \mathbb{R}^W$ and $\gamma^{(s)} \in \mathbb{R}^C$ are respectively height-wise, width-wise and channel-wise multiplication vectors, while $\gamma^{(b)} \in \mathbb{R}^C$ is a bias vector. The multiplication vectors on different dimensional levels encourage localized changes across the whole dimensionality of the activation maps, while keeping a restricted parameter count of $H+W+C$. This computation can be interpreted as a residual update which plays the role of a filter

for the activation map $A_c$.

The computation of the guiding parameters $\alpha$, $\beta$, $\gamma$ takes place through a linear projection over the final hidden state of the text embeddings output by a GRU fed with the text hint, analogously to [19, 64].

Lastly, the guiding mechanism is assisted by rescaling the standard CE loss, by giving a full weight to the classes mentioned in the hint (to encourage changes described in the hint), giving wrongly predicted pixels a 0 weight (to prevent hints from being associated with other visual classes), and giving 0.5 weight to initially correctly classified pixels (to discourage corrupting correctly classified regions).

In this modified architecture, when the guiding happens, the original segmentation network is kept frozen, while the guiding block and the GRU (except for the pre-trained word embeddings) are trained from scratch for each sample. Notably, the guiding happens separately for each input image. Therefore, the learned parameters are fitted over a single image, and must be trained again from scratch if guiding over a different image.

Although this study explores an interesting method for language supervision, it presents some limitations. Indeed, it restricts the application of the language supervision to the inference phase, avoiding investigating its potential on the training procedure. Also, I can argue that, since the GRU and the guiding are trained from scratch for each sample (except for the word embeddings), they can hardly extract from text robust and meaningful representations able to supervise the model's training globally. As a consequence, it is likely that the whole guiding mechanism merely overfits each sample in its own way.

I argue that these limitations pose serious issues of generality and scalability and that, therefore, the proposed approach cannot be adequately adapted or extended to work at training time.

Afterwards, [65] studied how to supervise vision encoders with high-level language specifications to guide their spatial attention to learn task-relevant features instead of distractors (Figure 2.24).

The authors proposed to augment the loss function with an auxiliary term that optimizes the alignment between the attention maps computed by CLIP and the attention maps computed by the classifier. For a batch $X$ of $m$ images of binary targets $y$, the loss is computed as such:

$$\mathcal{L}_{att}(X, y, \mathcal{A}_i^{\mathrm{VL}}, \theta) = \frac{\lambda}{m} \sum_{i=1}^{m} \left\| \mathcal{A}_i^{f_\theta} (1 - \mathcal{A}_i^{\mathrm{VL}}) \right\|_1, \tag{2.15}$$

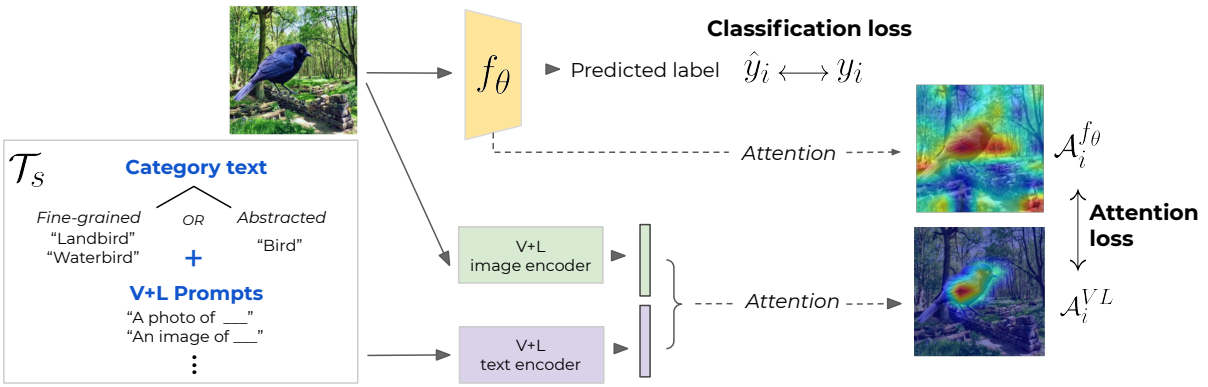



Figure 2.24: In [65], the vision encoder is trained with a loss term that promotes the alignment of the attention maps computed by the encoder and by CLIP. From [65].

where $\lambda$ is a hyperparameter that controls the strength of the attention supervision.

The CLIP attention maps $\mathcal{A}_i^{\text{VL}}$ are obtained by projecting into the shared latent space a set of CLIP-style prompts (e.g., "an image of `{class}`", "an image of `{class}`", etc.) formatted with the right class and the input images, and applying Grad-CAM [75] between the image-text similarity score and the feature maps after the last convolutional block in the image encoder. The classifier attention maps $\mathcal{A}^{f_\theta}$ are computed as the gradients of the output class with respect to the input image, i.e., $\mathcal{A}^{f_\theta} = \frac{\mathrm{d}y}{\mathrm{d}X_i}$.

The language supervision adopted in this work successfully guides the model learning process using text supervision, but it does not exploit the full expressive power of natural language. The textual representations are built on top of minimal, synthetic prompt formats and restricted to coarse-grained concepts. As with the works previously presented in this section, the potential benefits of a richer, deeper text supervision remain largely underexplored.

# 3 | Methods

This chapter describes the design choices and modules that act as foundations for the adopted methods.

Section 3.1 presents a framework for automatically generating diagnostic text that explains how segmentation predictions deviate from ground truth masks using MLLMs, describes the most important prompt design choices, and provides information on the approach leveraged for the evaluation.

Section 3.2 introduces the encoding of diagnostic text as a key component of the proposed pipeline, aiming to transform MLLM-generated diagnostic descriptions into embeddings that provide an additional supervisory signal during training. The section presents SDD, an image-text dataset built to improve vision–language encoders for the use case presented in this work, and provides an overview of the FLAIR fine-tuning stage.

Section 3.3 provides a thorough description of the proposed language-guided training pipeline, which augments standard segmentation learning with semantic feedback derived from language. The pipeline leverages a contrastive supervisory signal implemented by a Grouped SigLIP loss, which aims to improve training stability, and proposes a method to integrate synthetic textual negatives in the contrastive objective. The section also explains how to control the computational load of the pipeline through a custom data subsampling approach.

## 3.1. Diagnostic Text Generation

The investigated approaches revolve around textual data that encodes diagnostic information. Consequently, there is a strong need for robust and automated methods to generate textual information reliably. This task is naturally fulfilled by MLLMs, which are leveraged to generate text illustrating how the prediction mask deviates from the ground truth mask, effectively highlighting how and where the prediction is flawed, and judging the overall quality.

For each segmentation sample, the MLLM receives as input a class-specific prompt that

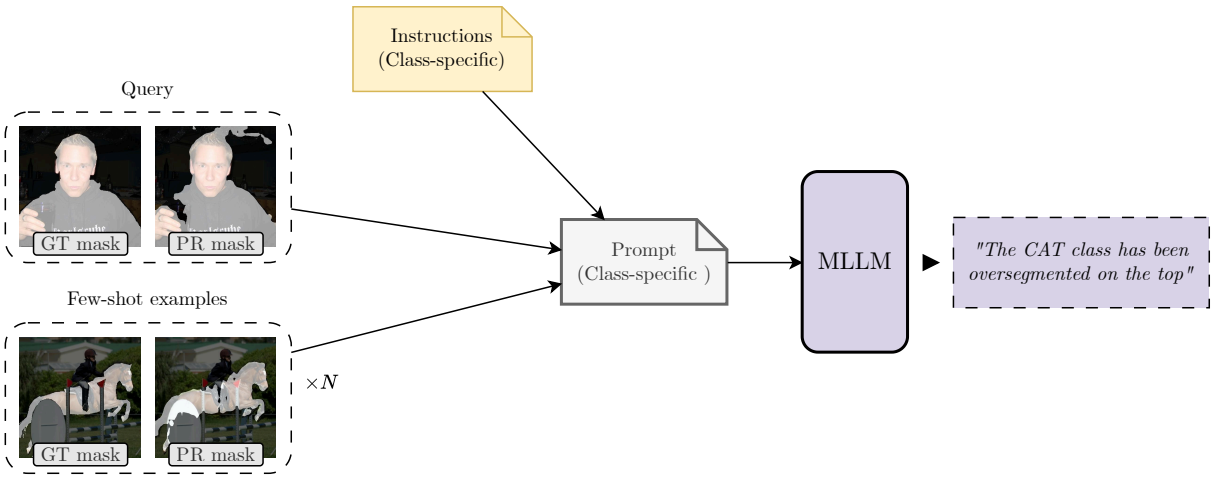



Figure 3.1: The prompt provided to the MLLM instructs it to produce diagnostic text based on the query ground-truth and predicted masks. It includes human-annotated examples that the model can leverage.

formats task instructions, the prediction mask generated by the segmentation model, and the ground-truth mask, and outputs the diagnostic text. The prompt features a set of demonstrations, aiding the model in its task. An overview of this is displayed in Figure 3.1.

The following subsections provide more information on the prompting strategy of the MLLM, on the mask formatting, and the adopted MLLM evaluation approach.

### 3.1.1. Class-split Prompting Strategy

The prompt fed to the MLLM must be complete and rich enough to allow the model to describe with expressiveness and consistency how the prediction compares to the ground truth, while still maintaining a reasonable computational efficiency. Therefore, in an attempt to strike a balance between the requirements, the prompting strategy features some fundamental non-trivial design choices.

One of these key choices is that the text generation process is class-specific: each prompt receives segmentation masks and instructions accounting for an individual class, with all other classes aggregated into a negative, unlabeled class. In the prompt formulation, the masks feature a binary class mapping, with the instruction explicitly referring to the positive class to enable the MLLM to provide a class-aware prediction. Since masks can, in general, comprise many class labels, they must be class-split into many binary masks, each one referring to a single class. An example of this is shown in Figure 3.2.

Class splitting offers several key advantages. It decomposes a complex multi-class task

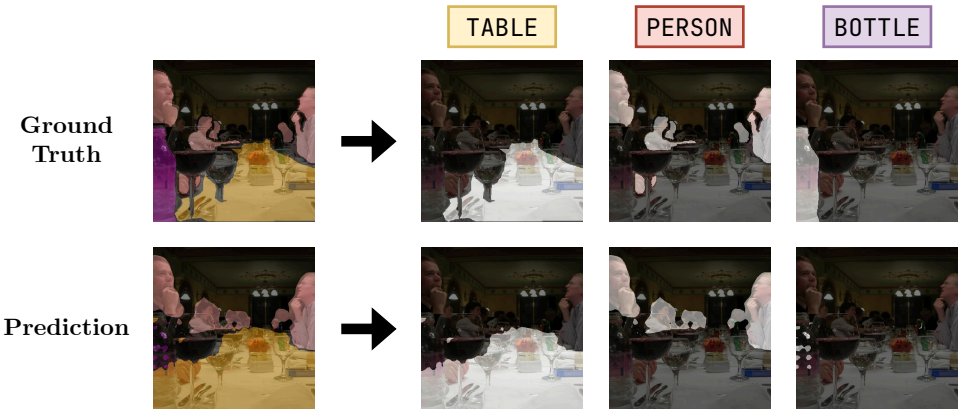



Figure 3.2: Class-splitting generates a binary mask for each class present in either the ground-truth or the prediction masks. Note that if a class is only present in one mask, the other mask is completely negative.

into multiple simpler predictions, each focused on a single class, and allows the MLLM to operate on cleaner, less cluttered inputs where all non-target classes are suppressed. Indeed, class-splitting disentangles the complexity of the answering task from the number of classes present in the masks. These benefits empirically lead to text outputs that are more accurate, expressive, and better grounded in the visual evidence.

Moreover, class splitting also positively impacts prompt design. When multi-class segmentation masks are used, classes must be encoded via a color map, which needs to be explicitly described in the prompt so the MLLM can interpret the masks correctly. Since there is no direct numerical encoding or textual representation of classes that the model could capture directly, the MLLM must visually inspect the masks and associate class colors with semantic labels.

This added abstraction increases both prompt and inference complexity: the prompt must carry auxiliary color-map information, and the model must rely on it to infer class identities from visual color cues. As a result of this additional responsibility, the MLLM becomes more prone to errors such as hallucinating absent classes, missing present ones, or confusing them.

Class-specific prompting avoids these issues. By providing binary (black-and-white) masks along with the explicit name of the target class, all necessary information is conveyed directly. The model no longer needs to reason over color mappings or infer class identities, resulting in a simpler prompt and higher-quality predictions.

On the other hand, class splitting also introduces notable drawbacks. Most importantly, it increases time complexity: given an $N$-class sample, instead of producing a single

prediction, the MLLM must generate one prediction per class, raising the computational cost from $\mathcal{O}(1)$ to $\mathcal{O}(N)$.

Furthermore, it leads to a variable number of inference passes, prompts, and generated texts, since each input sample must be replicated and processed separately for every class. This variability must be carefully managed at the scripting level to ensure that the original sample metadata is preserved and that batching and parallel operations remain effective.

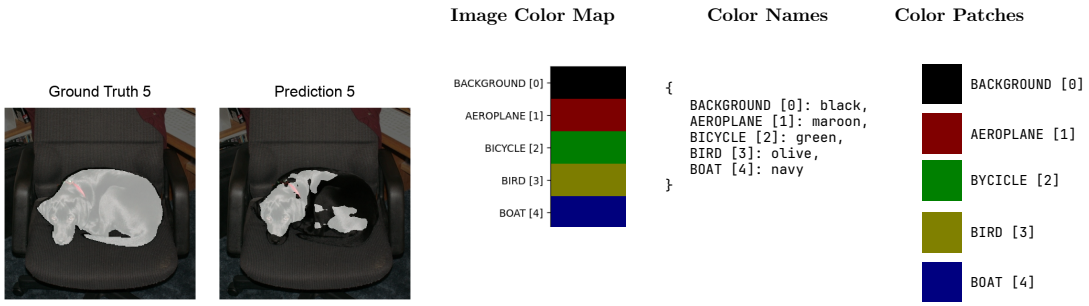
### 3.1.2. Mask and Color Map Formatting

Within the prompt, ground-truth and prediction masks are supplied as RGB images where pixels belonging to the target class are rendered in white and all other pixels in black. Prediction and ground-truth masks can be provided as separate images or concatenated in the same image object. An example of concatenated formatting is shown in Figure 3.3a. Optionally, the masks can be overlaid onto the corresponding background image to preserve visual context and enable context-aware predictions. This overlay is implemented via a simple convex combination of the mask and the background, with the blending coefficient $\alpha$ chosen to balance clear class separation against overall visual readability.

In the non-class-split scenario, the color map must be explicitly provided so that the LLM can interpret the classes encoded by the mask colors. The color map can be supplied in different formats, as illustrated in Figure 3.3b:

- **Image color map**: the class–color mapping is visually represented in a single image.
- **Color names**: each class is assigned the name of the closest matching color from the Python `webcolors` library, determined using the $L_2$ distance between the RGB vectors of integers in the range $[0, 255]$. This mapping is provided to the MLLM in textual form.
- **Color patches**: each class label is provided textually, while its corresponding color is given as an individual image patch. The patch size is set to be the visual token of the MLLM visual encoder.

The image color map format is the most straightforward, but it requires the MLLM to precisely localize class labels and colors, and correctly associate them to extract the mask colors. In contrast, the color names format forces the MLLM to correctly associate textual color names with the colors present in the mask. The color patches format lies between these two: it requires the MLLM to identify the class colors in the mask, match




(a) Example of concatenated masks pair.

(b) Image, names and patches color map formats.(b)

Figure 3.3: (a) visualises the concatenated masks formatting, (b) illustrates the color map formats tested in the non-class-split scenario.

them to the corresponding patches in the color map, and then map them to class names. Importantly, this formulation exploits the MLLM's ability to distinguish each patch as an independent visual token, and to enable a direct mapping to text without requiring fine-grained localization or complex compositional reasoning.

### 3.1.3. LLM-as-a-Judge Evaluation

The ability of the MLLM to successfully fulfill its task is crucial for the effectiveness of the subsequent methods. In particular, the model must accurately generate diagnostic text that describes the deviations between ground truth and predictions, capturing the most relevant elements with an appropriate level of detail, while minimizing hallucinations and omissions, and strictly adhering to the given instructions.

To validate and select the MLLM implementation and prompt settings most suitable for this open-ended answering task, traditional metrics are not appropriate since they cannot assess the semantic content of text pieces. As a result, I adopt an LLM-as-a-Judge approach (introduced in Subsection 2.4.2). The judge model is instructed to assess the correctness of predicted diagnostic texts with respect to ground truths, producing a binary decision ("correct" or "incorrect") based on predefined criteria. An overview of this is illustrated in Figure 3.4, with some examples shown in Figure 3.5.

In case of class-split answers, each class-specific answer is assessed, and sample-level accuracy is obtained as the average of its class-split answers. The overall accuracy is then computed as the average of these binary judgments among all samples.

For this evaluation, we manually annotated a total of 80 ground-truth diagnostic texts,

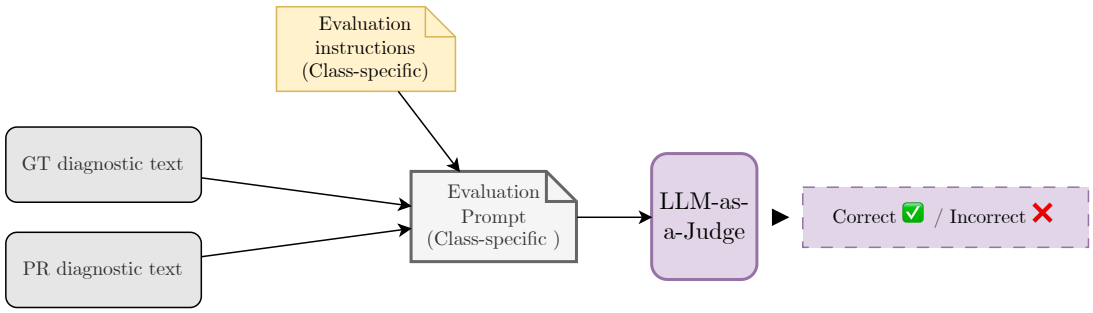



Figure 3.4: The evaluation prompt provided to the LLM-as-a-Judge instructs it to determine whether the prediction aligns with the ground truth.

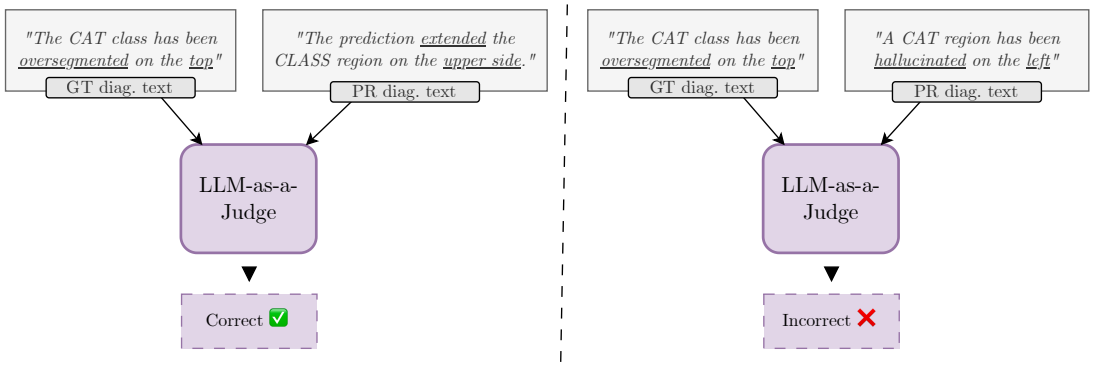



Figure 3.5: The LLM-as-a-Judge evaluates the prediction text by comparing its semantic content with that of the ground truth.

evenly split among annotators, with each receiving 20 samples. The annotated samples were randomly selected from the VOC2012 train–val split [24]. To ensure consistency, all annotators followed the same instructions provided to the LLM-as-a-Judge. Prediction masks were generated using LR-ASPP-MobileNetV3-Large [31], a pre-trained encoder-decoder segmentation model which operated on center-cropped images of size $520 \times 520$.

Given the inherent stochastic and subjective nature of LLMs-as-a-Judge, their usage requires a preliminary validation stage to assess the ability of the LLM to provide evaluations that are aligned with human judgments. To achieve this purpose, I manually annotated 20 ground-truth evaluations, built upon the first split of the 80 annotated text answers, based on diagnostic text predictions generated by Gemini 2.5 Flash [16], which was prompted in a non-class-split scenario, with separated images overlaid with background with an $\alpha$ factor of 0.6.

## 3.2. Diagnostic Text Encoding

The encoding of the diagnostic text generated by the MLLM plays a critical role in the language-guided pipeline. The text needs to be encoded in an embedding that can be integrated into the training to provide an additional supervisory signal, and the embedding must capture the diagnostic information capable of assisting the learning procedure.

Recalling Subsection 2.3.1, even state-of-the-art VLEs struggle to capture fine-grained, compositional information both in text and images. For this reason, I contribute to improving the performance gap by fine-tuning a checkpoint of the FLAIR VLE [90] on SDD, a synthetic dataset of diagnostic mask-text pairs. This additional fitting stage is aimed at enhancing the reasoning and grounding capabilities of the VLE with diagnostic visual and text difference information.

### 3.2.1. Synthetic Diagnostic Dataset

Synthetic Diagnostic Dataset (SDD) provides class-split diagnostic mask-text pairs and aims to improve the VLE contrastive capabilities for the use case presented in this work. The dataset is constructed by synthetically producing diagnostic masks and texts starting from all data points belonging to the VOC2012 and COCO2017 datasets, and it preserves the training and validation splits of the original datasets. All COCO2017 classes are standardized to VOC2012 using the official COCO2017 class mapping [48]. The VOC2012 UNLABELLED class is mapped to BACKGROUND. From both segmentation datasets, masks containing only the BACKGROUND class in both prediction and ground truth, either due to image resizing or the absence of VOC2012 classes in COCO2017 samples, are discarded.

The diagnostic samples making up the dataset are binary masks synthetically generated at the pixel level from each original sample, where pixels are set to white when the ground-truth and prediction masks disagree, and to black otherwise. Such binary masks are then overlaid with the background. The prediction masks were generated using LR-ASPP-MobileNetV3-Large[31], which operates on center-cropped images of size $520 \times 520$. These masks are then class-split following the approach introduced in Subsection 3.1.1. The diagnostic texts are generated by Gemma 3 12B QAT [78] following the same prompting procedure presented in Section 3.1. A few examples of SDD samples are visualized in Figure 3.6.

Gemma 3 12B QAT inference is powered by the LLM inference framework Ollama 0.8.0 [60] with Q4_K_M quants, and prompted with a set of parameters illustrated in Table 3.1.

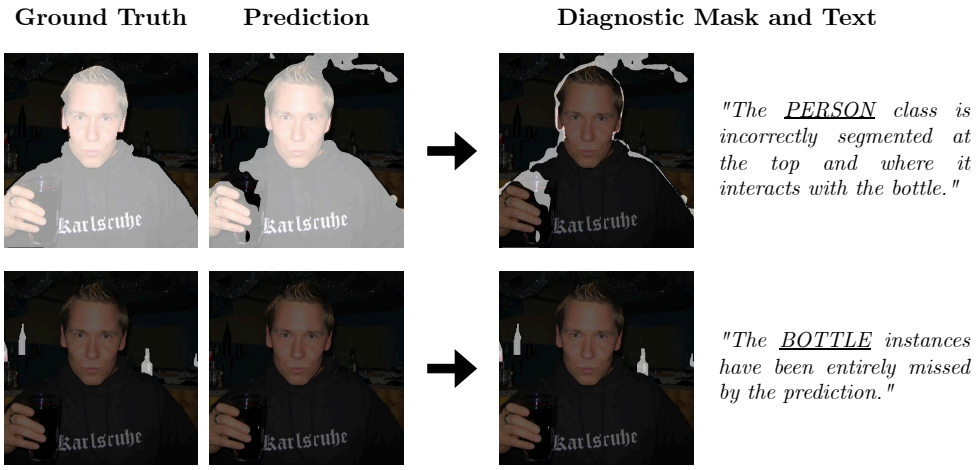



Figure 3.6: SDD diagnostic masks mark the regions where the ground truth and the prediction differ, while the textual descriptions explain the same discrepancy in natural language.

The dataset comprises a total of 184,017 training samples and 10,579 validation samples, with the splits further illustrated in Table 3.2. As shown in Figure 3.7, the class distribution of SDD mirrors that of the original VOC2012 [24] and COCO2017 [48] datasets. Consequently, SDD inherits a noticeable class imbalance: the PERSON (15) class is severely over-represented, accounting for about one third of the samples; CAR (7), CHAIR (9), and DININGTABLE (11) are moderately over-represented, while all other classes, except BOTTLE (5), present some degree of under-representation. In addition, most original samples contain only a small number of classes, and, as the class count increases, the number of samples comprising that number of classes decreases rapidly, with the average number of classes per original sample being slightly below two.

### 3.2.2. Fine-tuning FLAIR

To extract meaningful text features for subsequent stages of the pipeline, the FLAIR [90] text encoder was fine-tuned on SDD (see Subsection 3.2.1) to enhance its ability to capture relevant diagnostic information. The fine-tuning process followed an adapter-based approach [27, 76], where an adapter (hereafter referred to as text adapter) was trained from scratch to project the text encoder outputs into a new space of the same dimensionality, while keeping the original encoder parameters frozen. As highlighted in several key studies [14, 26, 42], this method often outperforms standard fine-tuning, as it preserves the encoder's core representations while introducing controlled, task-specific modifications, therefore reducing the risk of overfitting and catastrophic forgetting.

| Parameter | Value |
|---|---|
| **temperature** | 0.1 |
| **top-k** | 64 |
| **top-p** | 0.95 |
| **image size** | $224 \times 224$ |
| **overlay alpha** | 0.6 |
| **Masks formatting** | Separate |
| **one-shot VOC2012 UID** | 2007_000256 (class 1) |

Table 3.1: Gemma 3 12B QAT prompt and generation parameters for SDD predictions generation.

| Source Dataset | Training | Validation | Overall |
|---|---|---|---|
| **VOC2012** | 2,699 | 2,685 | 5,384 |
| **COCO2017** | 181,318 | 7,894 | 189,212 |
| **Overall** | 184,017 | 10,579 | 194,596 |

Table 3.2: Split sample counts of SDD, along with aggregated values along the rows and columns.

The loss guiding the training stage is the original FLAIR loss [90], obtained through the sum of a standard SigLIP loss term [97] measuring the alignment between the image and text feature vector (referred to as $\mathcal{L}^{\text{mps}}$ in the original work), and a second SigLIP loss term measuring the alignment between the text feature vector and the text-conditioned image feature vector (referred to as $\mathcal{L}^{\text{tcs}}$ in the original work). The loss is deployed in a single-positive setting, as configured by SDD.

## 3.3. Language-guided Pipeline

This subsection describes the training pipeline and the role of the diagnostic text generation and encoding modules. Overall, the pipeline can be seen as an enhanced version of a standard segmentation training procedure, in which additional language-based components guide the optimization of the segmentation model.

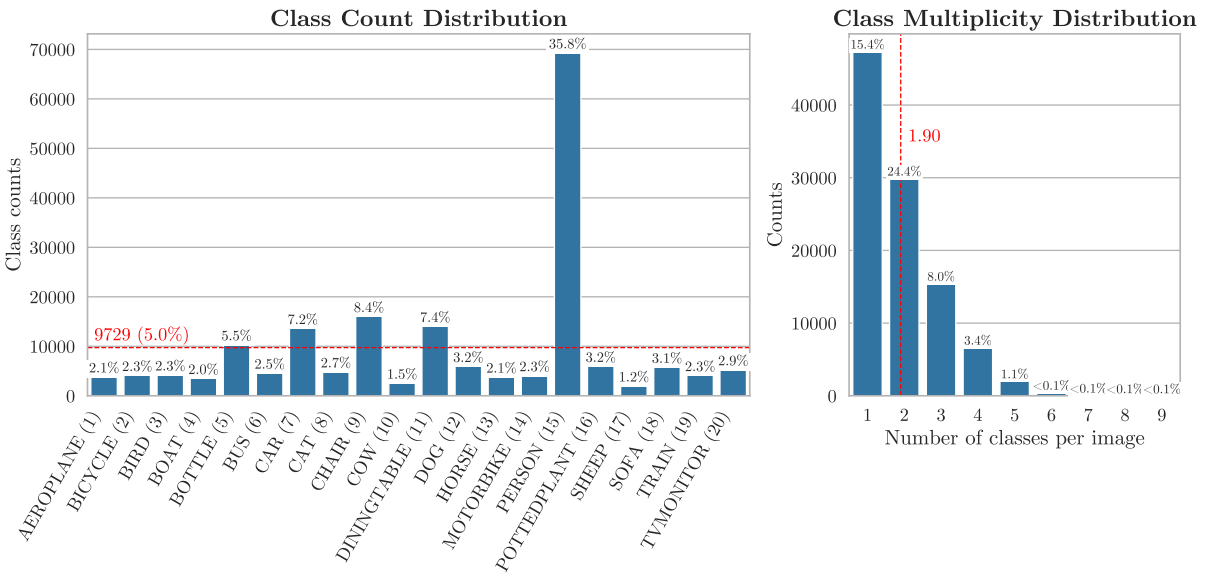



Figure 3.7: The SDD statistics are directly inherited from those of the original datasets. On the left, the class count distribution barplot witnesses the significant SDD class imbalance, while the barplot on the right shows the distribution of the number of classes per original sample.

### 3.3.1. Pipeline Overview

At a high level, the pipeline consists of a segmentation model augmented with a language-guided supervision signal. Alongside the usual mask prediction and pixel-wise supervision, the model leverages diagnostic textual descriptions derived from its own predictions and the corresponding ground truth. These descriptions are encoded and used to guide the alignment of visual features extracted from the segmentation model, encouraging the visual representations to become semantically meaningful and more effective for segmentation. A high-level overview of the pipeline is provided in Figure 3.8.

The central component of the pipeline is the segmentation model trained for the mask prediction task. During training, the predicted masks are supervised using a CE loss $\mathcal{L}_{\text{CE}}$, which measures the discrepancy between the predicted masks and the ground truth labels.

The same predicted masks and their associated ground-truth masks are then reused to construct the diagnostic text generation prompt, as described in Section 3.1. This prompt is fed to an MLLM, which generates diagnostic text strings following the predefined protocol.

The generated diagnostic text is subsequently processed by the VLE text encoder, which

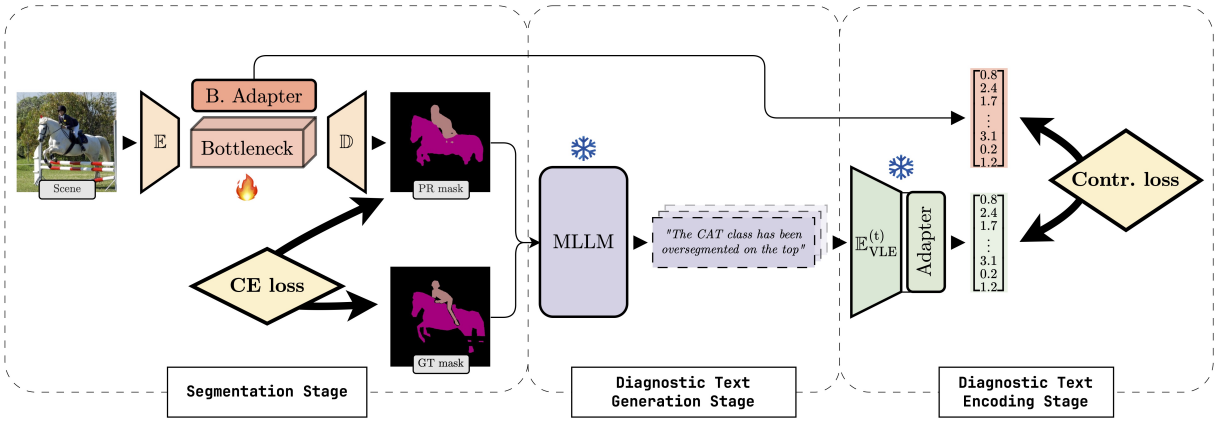



Figure 3.8: In the language-supervised pipeline, a frozen MLLM generates diagnostic textual descriptions from the predicted and ground-truth masks. These texts are encoded by a frozen vision–language text encoder into text feature vectors and aligned to adapted bottleneck feature vectors by means of a contrastive loss. The cross-entropy loss fully supervises the segmentation model, while its encoder is also optimized by the contrastive objective.

converts each text string into a numerical feature vector. The text encoder is kept frozen throughout training, as it provides reliable, ground-truth-like semantic representations of the diagnostic descriptions.

In parallel, during the forward pass of the segmentation model, the last feature volume produced by the segmentation encoder (hereafter referred to as the bottleneck) is extracted.

To enable interaction between visual and textual information, the bottleneck volume is first vectorized and, together with the text feature vector, mapped into a shared representation space. This mapping is required because the vectors of the two modalities are generally of different dimensionalities, which must be reconciled to allow proper alignment and interaction. These aggregating and mapping operations are performed by an additional trainable module (hereafter referred to as bottleneck adapter) , which can operate on both modalities and is trained from scratch. The bottleneck adapter modules analyzed in this work are further described in Subsection 3.3.5.

Once projected into the shared space, the visual and textual representations are aligned using a contrastive loss $\mathcal{L}_{\text{CL}}$. This language-guiding mechanism encourages the segmentation encoder to produce visual features that are consistent with the semantic content of the diagnostic text. While the text encoder remains frozen, the segmentation encoder is updated through both the contrastive loss and the CE loss.

The final training objective is defined as a multi-task loss that combines a CE loss $\mathcal{L}_{\text{CE}}$ with a contrastive loss $\mathcal{L}_{\text{CL}}$. This combination can be formulated in two different ways.

In the **auxiliary setup**, the loss is instead defined as

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{CL}}. \tag{3.1}$$

In this formulation, the CE loss remains the main optimization objective, while the contrastive loss acts as an auxiliary term whose influence is controlled by the weighting parameter $\lambda \in [0, 1]$. Increasing $\lambda$ strengthens the effect of the contrastive loss without diminishing the contribution of $\mathcal{L}_{\text{CE}}$. This setup is intended to assess whether the contrastive loss can serve as an auxiliary supervisory signal for the segmentation task.

In the **convex combination setup**, the loss is defined as

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{CL}}, \tag{3.2}$$

Here, the two loss terms are jointly weighted, yielding an explicit trade-off between the CE and contrastive objectives. As $\lambda$ increases, the contribution of the CE loss is proportionally reduced. This setup is designed to assess whether the contrastive loss can effectively replace, rather than merely assisting, the CE loss.

Ultimately, the pipeline trains the encoder to extract representations conditioned by the additional signal provided by the diagnostic text, so that the decoder can generate more meaningful and error-aware predictions. In the full pipeline, the CE signal supervises the entire segmentation model, while the contrastive signal supervises its encoder as well as the bottleneck adapter.

The additions introduced by this pipeline (i.e., the MLLM, the VLE, the bottleneck adapter, and the contrastive loss, tying everything together) serve the purpose of influencing the segmentation model during its training, but have no role at inference time, when the pipeline can be discarded, and the model can function on its own.

The pipeline is general enough to allow for the alignment of feature volumes at any encoder depth. I chose to consider the last one because it allows for the conditioning of the whole encoder, while considering an early, shallower feature volume would reduce the number of parameters supervised by the contrastive loss, thus harming its expressiveness. Also, the last feature volume tends to be the richest from a semantic perspective, and the most spatially compressed one, making it the best choice for the subsequent vectorizing operations.

Training a segmentation model from scratch initially produces highly noisy and incoherent masks. At this stage, these poor visual representations make it difficult for the MLLMs to generate meaningful and accurate text. For this reason, language-guided components are not introduced from the beginning but are added at a later training stage. This choice involves a trade-off: delaying language guidance improves stability, but it also limits how much influence language can have on conditioning the training process.

### 3.3.2. Grouped Contrastive Loss

The text-generation procedure described in Section 3.1 produces, for each sample, a variable number of diagnostic text strings. This number is determined at runtime and depends on the set of classes present in the predicted and ground-truth masks.

Consequently, applying a standard contrastive loss that independently aligns each positive bottleneck–text pair would introduce an imbalance: the contribution of each sample to the overall loss would scale with the number of associated class-specific texts. Samples generating many texts would disproportionately influence the loss, while those producing fewer texts would be underweighted.

To address this issue, the loss function must ensure that each sample contributes equally, independently of the number of class-split texts it generates. Following this principle, I adopt a custom variant of the SigLIP loss, referred to as Grouped SigLIP loss, which aggregates positive visual–text alignments at the sample level. Figure 3.9 displays an overview of the loss and shows an example computation, while the Python implementation is presented in Appendix C.1.

The Grouped SigLIP loss takes as input a batch of $P$ positive visual–text feature pairs (hereafter referred to as items) together with an index list of length $P$ specifying the group (which uniquely identifies the sample) to which each item belongs. Groups may contain an arbitrary non-zero number of items and may be singletons.

Operationally, the loss first computes the sigmoid-based loss for each positive and negative pair independently. Let $\ell_i^{(+)}$ and $\ell_{ik}^{(-)}$ denote the sigmoid-based loss contributions of the positive and $k$-th negative visual–text pairs associated with item $i$. Let $\mathcal{N}_i$ denote the set of negative pairs for item $i$.

Then, for each item $i$, it sums the corresponding positive term and all associated negative terms. In symbols:

$$L_i = \ell_i^{(+)} + \sum_{k \in \mathcal{N}_i} \ell_{ik}^{(-)}. \tag{3.3}$$

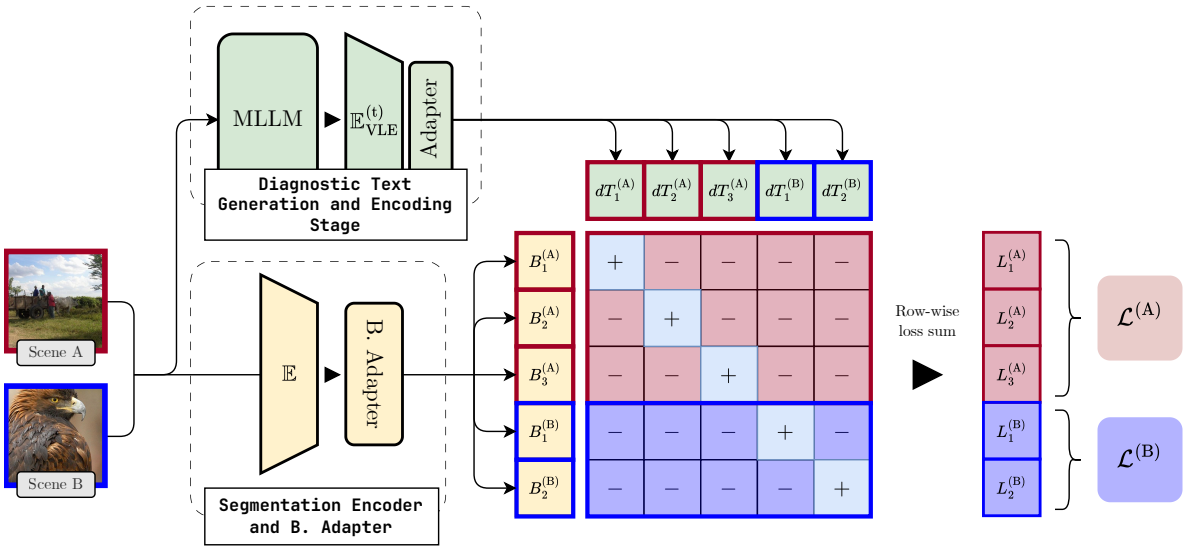



Figure 3.9: In the Grouped SigLIP loss, the sigmoid losses computed for each bottleneck-text vector are aggregated by bottleneck through summation, and the resulting values are averaged together by sample (group). This asymmetric behavior improves sample-level stability without weakening the contrastive component of the loss. In this figure, the notation is adapted to ease the visualization: items and loss terms are indexed by group and group position, rather than using a single flat index $i$.

The aggregated loss terms are normalized by the size of their group to which each item belongs, producing a group-normalized loss. Let $G_i$ be the group to which the item $i$ belongs, mathematically:

$$\tilde{L}_i = \frac{L_i}{|G_i|}. \tag{3.4}$$

Finally, the loss is obtained by averaging these normalized terms across all groups in the batch. Let $\mathcal{G}$ be the set of groups, formally:

$$\mathcal{L}_{\text{CL}} = \frac{1}{|\mathcal{G}|} \sum_{i=1}^{P} \tilde{L}_i. \tag{3.5}$$

This formulation treats the group-normalized terms as the fundamental units of optimization rather than the individual items. As a result, it removes dependencies on group size, allowing samples with variable numbers of associated texts to contribute equally to the final loss value.

It is worth noting that the proposed loss does not normalize the negative loss contributions at the group level. For each sample $i$, negative terms are simply summed and contribute

independently, regardless of their group membership. This design choice is motivated by the observation that normalizing negative contributions would weaken the discriminative component of the contrastive objective, thereby reducing the effectiveness of implicit hard negative mining.

The proposed loss is designed for class-split texts while offering flexibility in how bottleneck information is used. Since bottlenecks capture class-holistic patterns, it is not trivial to integrate them effectively in the class-split alignment process. In practice, there are two strategies for this.

The simpler strategy assigns the same sample vectorized bottleneck to all associated class-split texts (through unrolling). Thanks to the loss's grouped logic, the weight of each sample in optimization remains unaffected by the number of class-split texts. This method is straightforward and avoids extra complexity, but it limits expressiveness, as the alignment for each sample is determined by aggregating the class-split alignments. As a result, it leads to worse loss optimization and higher loss magnitudes.

The more advanced strategy tailors the bottleneck to each class-specific text, typically through some form of class-conditioning. This avoids per-sample aggregation and enables more precise, fine-grained alignment, better capturing the unique characteristics of each class-specific text. However, it introduces additional architectural complexity, extra learnable parameters, and a higher risk of overfitting.

Ultimately, the choice represents a trade-off between simplicity and expressiveness, and directly follows from the implementation of the bottleneck adapter, introduced in Subsection 3.3.1 and further explained in the upcoming Subsection 3.3.5.

Lastly, analogously to the standard SigLIP implementation [97], the Grouped SigLIP loss produces values whose magnitude depends on the number of negatives. Since the number of negatives is directly tied to the batch size, loss values are numerically comparable only when computed over batches of the same size.

### 3.3.3. Synthetic Contrastive Negatives

Negative loss terms play a crucial role in the behavior of contrastive objectives, as they provide the discriminative component that prevents collapse of the latent representation space.

The original implementation of the Grouped SigLIP loss relies on in-batch samples as negative examples. An alternative variant, whose Python implementation is presented in Appendix C.2, investigates the impact of using synthetic, per-image textual negatives

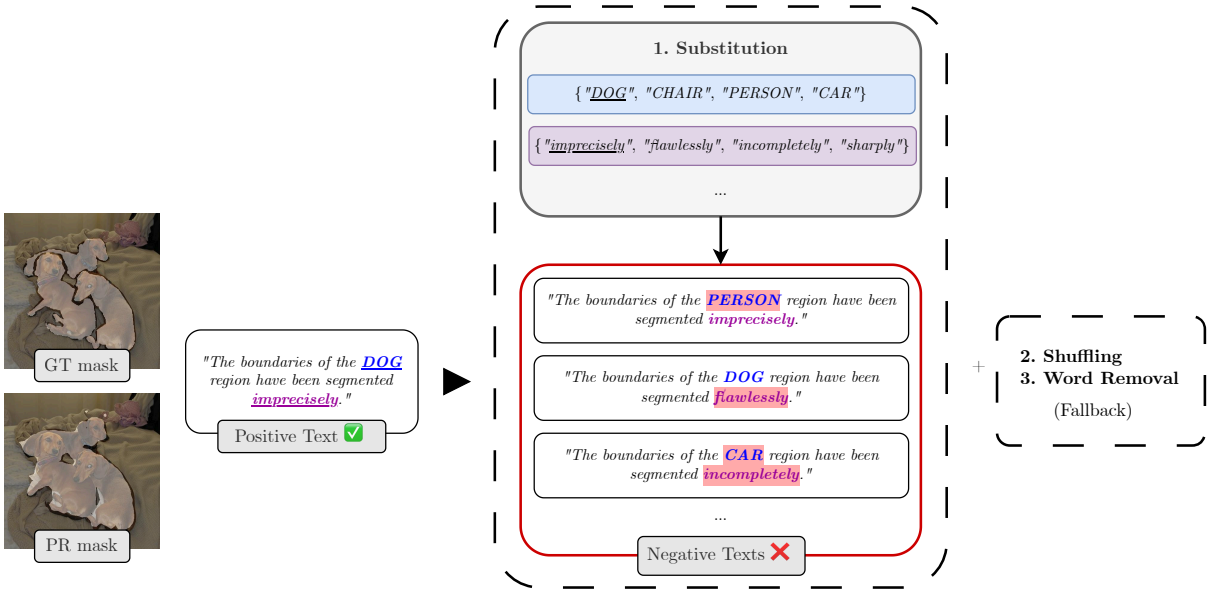



Figure 3.10: The synthetic negatives algorithm generates a fixed number of synthetic textual negatives from the corresponding positive text. It first applies random word substitution using predefined word pools, stochastically replacing eligible words. If this does not yield enough distinct negatives, it falls back to randomly shuffling the sentence words. Finally, if needed, it generates additional negatives by randomly removing a subset of words from the sentence.

during training. In practice, each positive item is associated with its own unique set of negatives, although the rest of the loss computation procedure remains the same.

Synthetic negative generation offers two main advantages: it can introduce harder negatives, and it allows explicit control over the number of negatives contrasted with each positive pair. This can lead to more effective loss optimization and enables the comparison of loss values computed with different batch sizes, provided that the same number of negatives is used.

Specifically, for each bottleneck visual feature, an algorithm generates a fixed number of synthetic negatives by perturbing the corresponding positive diagnostic text through a three-stage sequential procedure consisting of substitution, shuffling, and removal. A demonstration of the approach is presented in Figure 3.10.

In the first stage, random word substitution is applied. A set of predefined word pools is constructed, where each pool contains words that can be meaningfully substituted for one another while preserving grammatical correctness. For each positive text, the algorithm identifies all words belonging to any word pool. If a word appears in multiple pools, the first pool according to a fixed ordering is selected. Each eligible word is then

independently substituted, with some probability, by another word drawn uniformly from the same pool and different from the original. All substitutions are performed in a single pass, ensuring that replaced words are not modified again. Owing to the stochastic nature of this process, multiple distinct negatives can be generated from the same positive text. Mathematically, if a sentence comprises $N$ words contained in word pools of length $M$ (including the original word), then this substitution procedure can generate a maximum of $(M-1)^N$ synthetic negatives.

This substitution strategy, however, does not guarantee the generation of the desired number of negatives, as the number of substitutable words may be limited, and stochastic sampling can produce duplicate sentences. To address this, a second stage applies random shuffling of the sentence tokens, producing syntactically incorrect and semantically distinct negatives. Mathematically speaking, starting from a sentence of $N$ distinct words, reshuffling can generate up to $N! - 1$ synthetic negatives.

If the target number of negatives is still not reached, a final removal stage is applied. In this stage, an integer is sampled uniformly between 1 and half of the sentence length, and this corresponding number of words is randomly selected and removed from the sentence. From a mathematical point of view, starting from a sentence of $N$ distinct words, this removal approach can generate up to $\binom{N}{\lfloor \frac{N}{2} \rfloor}$ synthetic negatives.

The effectiveness of the negatives generation procedure depends primarily on the first substitution phase. Coherent and semantically meaningful replacements lead to higher-quality negatives, whereas poorly designed substitutions can degrade their usefulness. Consequently, the quality of the predefined word pools has the most direct impact on the final negatives, while their number and size also influence how many distinct negatives can be generated. In addition, the substitution probability controls the entropy of the process, balancing diversity against preservation of structure. For these reasons, careful design and tuning of the word pools and substitution parameters are essential to meet the desired trade-off between negative sample quality and quantity.

Conversely, the subsequent stages of random word shuffling and word removal mainly act as fallback mechanisms to ensure that the required number of negatives is obtained, since the negatives they produce tend to be less meaningful. For this reason, these stages are best applied sparingly, complementing the substitution phase by guaranteeing the target number of negatives while preserving an overall balance in negative sample quality.

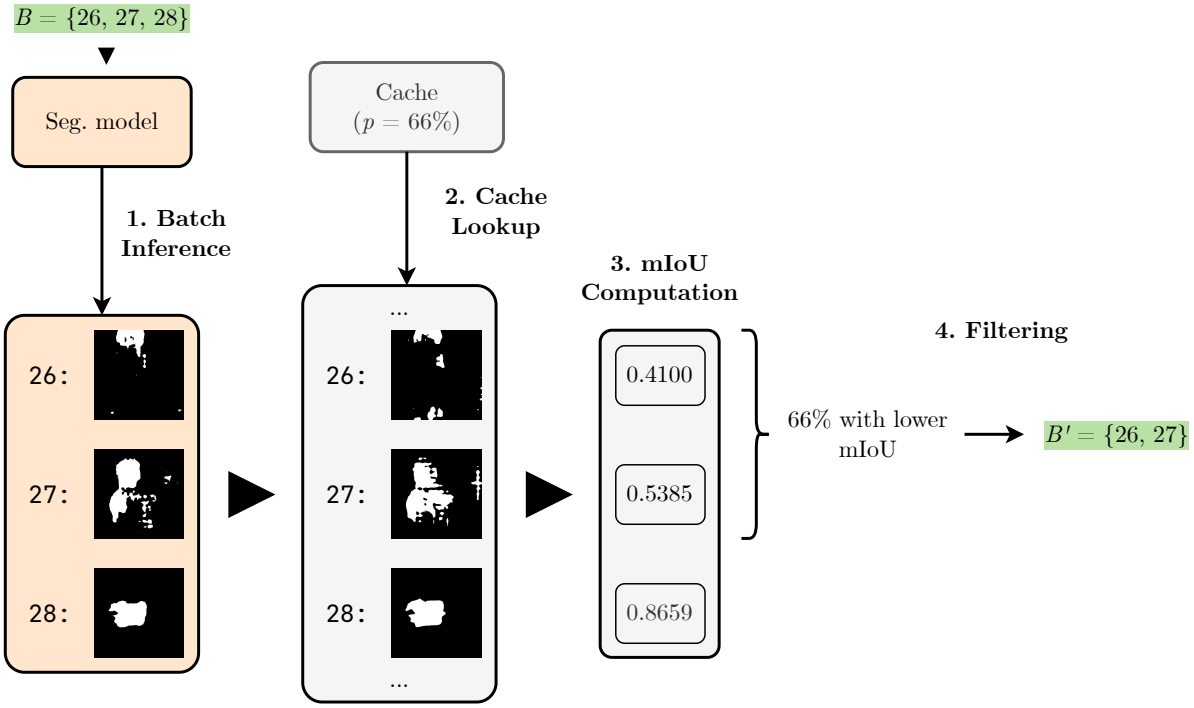



Figure 3.11: In the data subsampling approach, the masks of the batch samples are fetched from the cache and compared with the newly predicted masks. The top $p$ samples with the lowest mIoU (i.e., those that changed the most in the last training epoch) are retained for the subsequent diagnostic language conditioning process.

### 3.3.4. Data Subsampling Strategy

Given the high computational cost of MLLMs, controlling both computational and temporal complexity during training is critical. To this end, I introduce an input subsampling strategy that selectively determines which data samples are used to generate diagnostic text and participate in the text-guiding process. An overview and example of the subsampling process is visualized in Figure 3.11.

First, for each sample, only the class-specific samples corresponding to classes present in the ground truth are considered, while classes appearing exclusively in the predictions are discarded. This choice not only reduces the number of processed samples but also promotes training stability: ground truth classes provide consistent and reliable semantic anchors, whereas prediction-only classes vary across iterations and tend to be increasingly noisy when the segmentation model is still inaccurate. Restricting the pipeline to ground-truth-aligned information ensures that the supervision signal remains meaningful and well-conditioned throughout training.

Among the ground truth classes, only those whose predicted masks exhibit the greatest change over recent iterations are retained. This behavior is implemented via a cache-like memory that stores class-specific binary prediction masks. When a new batch of predictions is produced, the corresponding class masks are compared against their cached counterparts, and only a fixed percentage of masks with the lowest mIoU are retained.

This heuristic is deliberately designed and motivated by a clear intuition: prediction masks that vary substantially over time are those that most strongly influence model training and are therefore highly informative for guiding learning. In contrast, masks exhibiting only minor changes typically provide weaker supervision and contribute less to the optimization process, and might even encourage overfitting. By prioritizing samples with higher temporal variation, the method concentrates language-guided supervision on the most impactful regions of the learning space. After selection through batch subsampling, the newly chosen masks are then used to update the cache, ensuring that the stored representations remain aligned with the model's evolving state throughout training.

Notably, many standard segmentation training pipelines (e.g., [13, 31, 39, 52, 69, 74]) rely on regularization strategies that deliberately inject stochasticity into the training process. These include data augmentations such as random cropping and horizontal flipping, as well as model-level sources of randomness like dropout [29]. While beneficial for generalization, this induced variability is detrimental to the subsampling strategy: it causes segmentation masks to fluctuate due to extrinsic randomness rather than genuine changes in the model's predictions. As a result, the measured dissimilarity between masks may reflect augmentation noise rather than meaningful learning dynamics, therefore weakening the connection between mask temporal variation and informativeness. As a consequence, without carefully controlling or accounting for these sources of randomness, the sample temporal dissimilarity estimates become less representative of the actual samples' informativeness.

Furthermore, to avoid a cold start, training should begin with a cache that is already aligned with the selected checkpoint. If the cache is empty, it will not contain prediction masks, and the initial subsampling operations will not leverage the dissimilarity criteria. A simple way to address this issue is to prefill the cache before starting training. This can be done by running a forward pass over the entire training dataset once, using the chosen checkpoint, and collecting all the corresponding prediction masks. By doing so, the cache is properly initialized and consistent with the model state, ensuring correct and stable behavior from the very first training iteration.

It is important to note that the proposed subsampling strategy effectively alters the batch

size used in the contrastive loss. As discussed in Subsection 3.3.2, Grouped SigLIP loss values are directly comparable only when computed over batches of the same size; therefore, the same retention percentage $p$ must be used as well. This has important implications for comparing training and validation contrastive losses: to ensure a faithful evaluation, the validation loss should not be subsampled (to avoid approximations), and the training loss should also be computed without subsampling. In contrast, the loss variant with synthetic negatives, as outlined in Subsection 3.3.3, can operate with different batch sizes, and thus different $p$ values, provided that the number of negatives generated per image remains the same. However, this variant is unsuitable for faithful validation contrastive loss computation, as the stochastic generation of synthetic negatives can introduce variability that distorts evaluation. Using only true in-batch negatives provides a consistent and reliable measure of model performance.

Overall, this sampling strategy is designed to enhance the efficiency of the language-guided pipeline while striving to maintain training effectiveness. Reducing the number of considered samples proportionally enhances the temporal complexity but may negatively affect the performance and stability of the contrastive loss optimization, and, consequently, of the language-guiding mechanism. In particular, contrastive learning benefits from large batch sizes due to the increased number of negative samples. Subsampling effectively lowers the batch size, potentially resulting in noisier and less stable optimization. Therefore, the subsampling ratio must be carefully tuned to balance the gains in computational efficiency against the acceptable loss in robustness.

### 3.3.5. Bottleneck Adapters

As outlined in Subsection 3.3.1, the interaction between the bottleneck feature volume and the diagnostic text feature vector is mediated by a bottleneck adapter. This module is responsible for compressing the 3D feature volume into a 1D representation by reducing its spatial dimensions, and for projecting both modalities into a shared embedding space.

The implementations adopted in the experiments, illustrated in Table 3.3, are grouped into three categories:

- **Linear-based**: `GAP+linear (text-side)`, `GAP+linear`, and `GAP+linearX2` modules first pool the spatial dimensions with non-learnable Global Average Pooling (GAP) layers, then adapt the dimensionality of the resulting vectors through linear layers.
- **Convolution-based**: `convX2+GAP` module first adapts the dimensionality of the feature volumes through point-wise convolutions, then pools the spatial dimensions

| Module Name | Block | Param. Count | Spatial Processing | Learnable Pooling |
|---|---|---|---|---|
| `GAP+linear (text-side)` | (bn.-side) GlobalAveragePooling<br>(text-side) Linear($D_\mathrm{t} \to D_\mathrm{bn}$) | $D_\mathrm{bn}(D_\mathrm{t}+1)$ | | |
| `GAP+linear` | GlobalAveragePooling<br>$\to$ Linear($D_\mathrm{bn} \to D_\mathrm{t}$) | $(D_\mathrm{bn}+1)D_\mathrm{t}$ | | |
| `GAP+linearX2` | GlobalAveragePooling<br>$\to$ Linear($D_\mathrm{bn} \to D_\mathrm{t}$)<br>$\to$ BatchNorm1D<br>$\to$ ReLU<br>$\to$ Linear($D_\mathrm{t} \to D_\mathrm{t}$) | $(D_\mathrm{bn}+D_\mathrm{t}+3)D_\mathrm{t}$ | | |
| `convX2+GAP` | Conv1x1($D_\mathrm{bn} \to D_\mathrm{t}$)<br>$\to$ BatchNorm2D<br>$\to$ ReLU<br>$\to$ Conv1x1($D_\mathrm{t} \to D_\mathrm{t}$)<br>$\to$ GlobalAveragePooling | $(D_\mathrm{bn}+D_\mathrm{t}+3)D_\mathrm{t}$ | ✓ | |
| `conv+attnPool` | Conv1x1($D_\mathrm{bn} \to D_\mathrm{t}$)<br>$\to$ MHA($d=D_t$, $h=8$) | $(D_\mathrm{bn}+4D_\mathrm{t}+7)D_\mathrm{t}$ | ✓ | ✓ |
| `conv+attnPoolByText` | Conv1x1($D_\mathrm{bn} \to D_\mathrm{t}$)<br>$\to$ MHA($d=D_t$, $h=8$) | $(D_\mathrm{bn}+4D_\mathrm{t}+6)D_\mathrm{t}$ | ✓ | ✓ |

Table 3.3: The table lists each bottleneck adapter module, detailing its constituent layers and the parameter count in terms of the bottleneck and text vector dimensionalities, $D_\mathrm{bn}$ and $D_\mathrm{t}$. It also highlights the module's capabilities for spatial processing and learnable pooling.

with non-learnable GAP layers.

- **Attention-based**: `Conv+AttnPool` and `Conv+AttnPoolByText` modules adapt the dimensionality of the feature volumes through point-wise convolutions, then pool the spatial dimensions through learnable attention pooling layers, following the implementation in [90].

From a conceptual standpoint, linear-based modules first reduce the spatial dimensions and then adapt the feature dimensionality. This design has the lowest expressive power, as spatial information is discarded immediately and no spatial processing is performed. In contrast, convolution-based and attention-based modules process spatial information before pooling, resulting in higher expressive power since they can operate on the spatial structure.

Both linear-based and convolution-based modules rely on non-learnable pooling operations. In this work, I adopt GAP [47] layers to remain consistent with standard image classification pipelines, where GAP layers are commonly used to vectorize convolutional feature maps. In contrast, attention-based modules, while also using convolutions to process spatial information and adjust dimensionality, perform pooling via learnable parameters.

Within the attention-based approaches, the key distinction between `conv+AttnPool` and `conv+attnPoolByText` modules lies in the choice of the query used for pooling. The former employs a query vector learned from scratch to aggregate visual tokens, whereas the latter uses external query vectors. Following the FLAIR text-conditioned attention pooling strategy [90], introduced in Subsection 2.3.1, I use the class-split diagnostic text feature vectors as queries to generate text-conditioned bottleneck representations. Unlike the other methods, it introduces explicit bottleneck class-conditioning: by pooling with a query corresponding to each class-specific text, the adapter produces a distinct bottleneck representation conditioned on each class, resulting in the effects discussed in Subsection 3.3.2.

Among all the modules, `GAP+linear (text-side)` is the only one that processes text features, as it adapts the dimensionality of the text vectors. All other modules operate exclusively on the visual bottleneck, leaving the text representations produced by the VLE unchanged. This preference for bottleneck-sided adaptations is based on the foundational assumption that, in the pipeline, diagnostic text serves as ground-truth-like information and should not be further processed during the segmentation model training. Additionally, the dimensionality mapping is always one-sided to minimize the number of additional learnable parameters.

Finally, it is worth noting that all modules introduce learnable parameters optimized through the contrastive loss, which supervises their training. Consequently, the contrastive loss is not solely optimized by the segmentation encoder. This may reduce the impact of the language guidance on the segmentation model and complicate the analysis of its actual effects. On one hand, this issue tends to grow stronger as the number of parameters, i.e., the module's complexity, increases. On the other hand, more complex modules improve the effectiveness of the dimensionality mapping and the pooling operation.

# 4 | Experiments

This chapter presents the experiments conducted in this work, detailing the experimental setup and the results.

Section 4.1 covers the preliminary experiments on the MLLM.

Section 4.2 presents the preliminary experiments on FLAIR.

Section 4.3 details the experiments and ablations on the language-guided pipeline.

### Experimental Setup and Reproducibility

All experiments were conducted locally on multiple Nvidia RTX 1080 Ti GPUs with 12 GB of VRAM and Nvidia Quadro RTX 6000 GPUs with 24G of VRAM. Further details are provided in each section. The code used for the experiments is publicly available at `https://github.com/Luca-Olivieri/training-text-description-loss`.

## 4.1. Preliminary Experiments on the MLLM

This section reports the results related to prompt design and MLLM validation and selection, which precede the full pipeline experimentation.

### Experimental Setup

The MLLM generation parameters are displayed in Table 4.1. Temperature was kept low to reduce text variability but different from 0 to avoid generation collapse. For each model, top-$p$ and top-$k$ were assigned the default values configured by their manufacturers. Gemini 2.5 Flash [16] was accessed through the Google AI Studio API, while all the other MLLMs were hosted locally by the LLM inference framework Ollama 0.8.0 [60] with Q4_K_M quants. Quantization was employed to reduce memory usage and inference time, while optimally preserving model capabilities. Local models received images at a smaller size to further reduce the inference temporal complexity.

| MLLM | Temperature | Top-p | Top-k | Image size |
|---|---|---|---|---|
| Qwen2.5VL 7B [7] | 0.1 | 1.00 | all | $224 \times 224$ |
| Gemma 3 12B QAT [78] | 0.1 | 0.95 | 64 | $224 \times 224$ |
| Mistral S 3.1 24B [59] | 0.1 | 1.00 | all | $224 \times 224$ |
| Gemma 3 27B [78] | 0.1 | 0.95 | 64 | $224 \times 224$ |
| Gemini-2.5-flash [16] | 0.1 | 0.95 | 64 | $520 \times 520$ |

Table 4.1: The table shows the generation parameters used for the experiments.

For prompt construction, each mask was overlaid on the original image with $\alpha = 0.6$ to ensure a balanced trade-off between visual clarity and mask–background separation (see Subsection 3.1.2). The prompts include three few-shot examples selected from VOC2012 [24], corresponding to UIDs [2007_000256, 2007_000039, 2007_000332], respectively of classes 1, 20, and 13 in class-split experiments.

### 4.1.1. Evaluation Prompt Selection

Before introducing the inference prompt and MLLM selection phase, this subsection outlines the evaluation prompt experiments carried out to align the LLM-as-a-Judge with human judgments. Specifically, multiple evaluation prompts were tested in both non–class-split and class-split settings by means of 20 ground-truth evaluations (as introduced in Subsection 3.1.3). In all experiments, Gemini 2.5 Flash [16] is employed as the LLM-as-a-Judge.

#### Non-class-split Scenario

In the non-class-split scenario, the evaluation prompts were created by augmenting a base prompt with a variety of instructions that introduce diverse evaluation criteria. Different combinations of these instructions can be selected to steer the automated evaluation process. The base evaluation prompt considered in this work, shown in the following frame, is adapted from [82] to suit the considered scenario. In the frame, parameters in '`[·]`' are setting-dependent, while parameters in '`{·}`' are sample-dependent.

**LLM-as-a-Judge Evaluation Prompt**

You are an intelligent chatbot designed for evaluating the correctness of AI assistant predictions for question-answer pairs. Your task is to compare the predicted answer with the

ground-truth answer and determine if the predicted answer is correct or not. Here's how you can accomplish the task:

---

## INSTRUCTIONS:
- `[ADDITIONAL_INSTRUCTIONS]`

Please evaluate the following answer:

Ground truth correct Answer:

`{GT_ANSWER}`

Predicted Answer:

`{PR_ANSWER}`

Provide your evaluation as a correct/incorrect prediction along with the score where the score is an integer value between 0 (fully wrong) and 5 (fully correct). The middle score provides the percentage of correctness. Please generate the response in the form of a Python dictionary string with keys 'pred', 'score' and 'reason', where value of 'pred' is a string of 'correct' or 'incorrect', value of 'score' is in INTEGER, not STRING and value of 'reason' should provide the reason behind the decision.

Only provide the Python dictionary string. Escape properly quotes within the 'reason' string. For example, your response should look like this: {'reason': reason, 'score': 4, 'pred': 'correct'}.

The evaluation prompt variations were built by introducing different combinations of the following statements in the instructions list.

- General assessment rules:
  - $\mathbf{G_1}$: *"Focus on the correctness and accuracy of the predicted answer with the ground-truth."*.
  - $\mathbf{G_2}$: *"Focus on the correctness, completeness and accuracy of the predicted answer with the ground-truth."*
- Guidelines regarding error types:
  - $\mathbf{T_1}$: *"Be critical of significant inconsistencies of error types."*
  - $\mathbf{T_2}$: *"Be critical of significant inconsistencies of error types and spatial locations."*
- Other guidelines:

- **D**: *"Consider predictions with less or more specific details (as long as they show some consistency with the ground truth) as correct evaluation."*
- **S**: *"Be strict with your evaluations."*
- **P**: *"Expect precision from the predicted answer; it is not enough for it to roughly capture the essence of the ground truth; the prediction has to overlap sufficiently with the ground truth to be considered correct."*

The evaluation prompt asks the LLM-as-a-Judge to output a decision ("correct" or "incorrect"), a score in the $[1, 5]$ range, and a reason. Only the decision was retained, while the score and reason were used solely to support it. Performance was then measured by computing the average accuracy between the predicted and the ground-truth decisions among all the available samples.

A set of possible instruction combinations was tested, and the results, shown on the left side of Figure 4.1, indicate that the tested prompts require careful tuning through effective combinations of instructions to achieve competitive performance. The largest accuracy improvements were obtained by incorporating notions of precision, error categorization, and spatial awareness. This suggests that the base evaluation prompt is overly permissive and vague, and must be made more critical and analytical to yield meaningful assessments. The combination <G2, D, S, P, T2> achieved the highest accuracy of 85% and was therefore selected as the optimal non-class-split evaluation prompt.

### Class-split Scenario

In the class-split scenario, accuracy results from experiments using non-class-split evaluation prompts can still be applied, but it is important to carefully evaluate the pertinence of automated class-split assessments. For class-specific diagnostic text predictions, evaluations should focus solely on the positive class and avoid being influenced by statements related to other classes. This is particularly crucial in this assessment phase, where the ground truths to which predictions are compared are not class-specific.

A straightforward and effective way to evaluate the pertinence of automated assessments is to verify that the text includes the intended positive class while excluding the other classes. Restricting the MLLM to only mention classes in uppercase makes it easy to check for their presence. This method, however, serves only as a heuristic: it does not account for variations such as lowercase letters, plurals, or irregular forms, nor does it distinguish between a class mentioned as a reference or one that is actually being evaluated. Despite these limitations, it provides a useful estimate of the expected evaluation pertinence.

To address the challenge of constraining the LLM-as-a-Judge to focus on specific classes, the previously selected non–class-split evaluation prompt was extended with three additional statements:

- **Positive class inference**: prepending *"The predicted answer is class-specific: for your evaluation, only consider the class involved in the predicted answer, ignore statements in the ground truth that are related to other classes not present in the prediction."* to the instruction list, to let the LLM-as-a-Judge infer the positive class from the class-specific predicted answer.
- **Positive class specification**: prepending *"The predicted answer only covers the class* `{POS_CLASS}`*: ignore all the statements in the ground truth that consider other classes. Your evaluation should only be based on the statements concerning the* `{POS_CLASS}` *class."* to the instruction list, to explicitly define the positive class.
- **Positive class specification (recency)**: appending *"### Important: \n The predicted answer only covers the class* `{POS_CLASS}`*: restrict your focus exclusively on the* `{POS_CLASS}` *class, you must ignore all the statements in the ground truth that consider different classes. If the ground-truth answer does not mention the class* `{POS_CLASS}`*, assume it is saying that "Both masks have no* `{POS_CLASS}` *in them, so there is no deviation.""* at the end of the prompt, to explicitly define the positive class, leveraging the recency bias [20].

In this context, an automated evaluation was considered correct if it mentioned only the positive class and no other classes. Pertinence was measured as the proportion of correct evaluations over the total number of evaluations. The results, shown on the right side of Figure 4.1, indicate that the considered LLM-as-a-Judge setup succeeds in producing pertinent class-specific evaluations. While positive class inference alone yields reasonably high pertinence, the best performance is achieved by explicitly instructing the LLM to focus on the specified positive class. Further reinforcing this instruction through recency bias leads to a maximum pertinence of 100%. Consequently, this final prompt was selected as the optimal class-split evaluation prompt. An example of a fully formatted evaluation prompt built using these settings is provided and visualised in Appendix A.1.

### 4.1.2. Inference Prompt and MLLM Selection

By leveraging the LLM-as-a-Judge evaluation prompts selected in the previous subsection, the diagnostic text generation capabilities of the MLLM can be systematically evaluated. The quality of the generated text depends both on the model's suitability for the task and

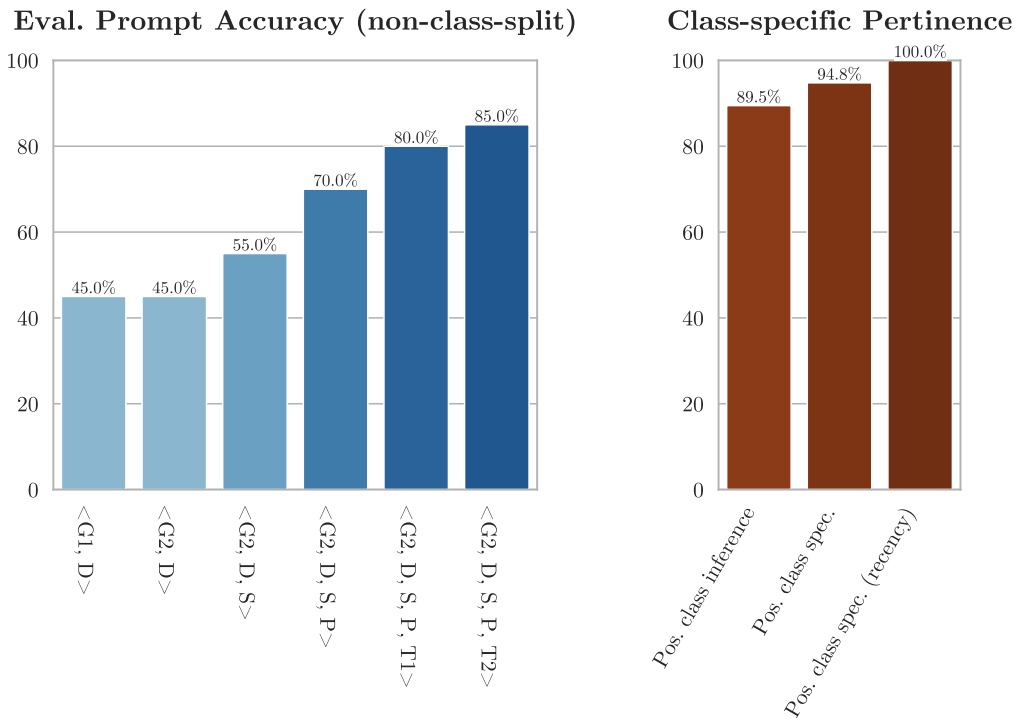



Figure 4.1: On the left, the bar plot shows the accuracy of the LLM-as-a-Judge under non–class-split evaluation prompts with different instruction sets. On the right, the plot shows the pertinence scores associated with the statements used to augment the non-class-split evaluation prompt, making it class-specific.

on the effectiveness of the inference prompt. This subsection describes the experiments conducted on the inference prompts and on the MLLM itself, and analyzes the resulting performance. As in Subsection 4.1.1, Gemini 2.5 Flash [16] is adopted as the MLLM-as-a-Judge, using either the non-class-split or class-split evaluation prompts depending on the experimental setting.

Text generation with MLLMs poses inherent challenges. Overly restrictive prompts may harm generation quality by constraining the model with excessive requirements, while overly generic prompts can result in inconsistent and highly variable outputs. Ideally, generated texts should be complete and expressive, yet sufficiently consistent and standardized. This balance is particularly important in a training pipeline, where stable but meaningful textual representations are required.

To address this trade-off, a simple inference prompt with a limited number of explicit instructions was adopted, relying on few-shot examples to implicitly convey the desired output standard. The following experiments evaluate the effectiveness of this design and analyze the impact of different factors, including class-split vs. non-class-split scenarios, mask formatting, the few-shot examples count, and the choice of the MLLM.

The base prompt template is shown in the following frame. The few-shot examples section is repeated for each few-shot example, and omitted if there are no shots. Format parameters in '`[·]`' are setting-dependent, while those in '`{·}`' are sample-dependent.

**MLLM Inference Prompt**

I am in a binary semantic segmentation context, and I want to compare a **prediction** mask with a **ground truth** mask, both segmented over the same **scene**.

`[SCENARIO_INSTRUCTIONS]`

`[MASK_FORMATTING_INSTRUCTIONS]`

## Instructions

Your task is to find where and how the prediction deviates from the ground truth. Assume the ground truth to be correct. If there are no significant deviations, simply say it.

To help you, I will give you a set of example images, each associated with an ideal answer, which might mention classes whose names are irrelevant to your problem.

EXAMPLE `[SUPPORT_SET_INDEX]`.

Input:

`[SHOT_GT_PR_MASKS]`

Output:

"The ground truth AEROPLANE region has been segmented quite well by the prediction, but the boundaries are imprecise and less defined."

---

`[{POS_CLASS_REFERENCE}]`

Input:

`[{QUERY_GT_PR_MASKS}]`

Output:

The structure below defines the substitutions for each setting-dependent parameter.

- `[SCENARIO_INSTRUCTIONS]`:
  - **Non-class-split**: *"In the masks, the classes are mapped to a specific color as defined by a color map, which is shared by both masks. What follows is a mapping between class names, class identifiers and colors. `[COLOR_MAP]`"*
  - **Class-split**: *"In both masks, a color-class mapping is applied: the white color is mapped to the `{POS_CLASS}` class, while the black color refers to unlabelled classes."*
- `[MASK_FORMATTING_INSTRUCTIONS]`:
  - **Separate**: *"I will give you two images: the first image is the ground truth mask, the second image is the prediction mask. Both images are overlaid with the scene to support your analysis."*
  - **Concatenated**: *"I will give you an image which concatenates horizontally the ground truth and the prediction, which are displayed side-by-side. The ground truth and the prediction are marked by the corresponding title above them. In the whole image, the image on the left is the ground truth, and the image on the right is the prediction. Both images are overlaid with the scene to support your analysis."*
- `[SHOT_GT_PR_MASKS]`/`[{QUERY_GT_PR_MASKS}]`:
  - **Separate**: *"Ground Truth. `{GT_MASK}` \n Prediction. `{PR_MASK}`"*
  - **Concatenated**: *"Ground Truth and Prediction. `{CONCAT_GT_PR_MASKS}`"*
- `[POS_CLASS_REFERENCE]`:

- **Non-class-split**: omitted from the prompt.
- **Class-split**: *"Now, I ask you to generate the output based on the following input. Remember that the considered class is the* `{POS_CLASS}` *class; reference it explicitly in the answer."*

The first set of experiments contrasts the class-split and non-class-split scenarios. In the non-class-split setting, the representation of the color map, provided either as an image, a list of class names, or a list of color patches, was also tested. Diagnostic text predictions were generated using Gemini 2.5 Flash [16] and compared against the first split of 20 text targets of the 80 manually annotated for the purpose (see Subsection 3.1.3). Performance was measured as the average accuracy across the separate and concatenated mask formatting configurations.

The results, shown in Figure 4.2, indicate that diagnostic text generation is challenging in both non-class-split and class-split scenarios. All non-class-split evaluated configurations achieve at best a moderate accuracy of 66.5%, whereas the class-split scenario attains a substantially higher accuracy of 74.0%. Within the non-class-split setting, representing the color map as a list of patches yields better performance than image-based or name-based formats, likely due to improved separation of class-specific color information, although remaining markedly inferior to the class-split results. Consequently, all subsequent experiments focus exclusively on the class-split scenario.

The next experiment investigates the impact of mask formatting choices, few-shot prompting, and the selection of MLLMs. As discussed in Subsection 2.4.1, both the arrangement of input images and the number of few-shot examples provided to an MLLM can substantially influence its performance. To analyze these effects, multiple MLLMs with different parameter counts were evaluated using both separate and concatenated mask formatting strategies, and with a number of few-shot examples ranging from 0 to 3, while preserving the fixed ordering of example UIDs. Since, as observed in Subsection 2.4.1, each MLLM exhibits distinct sensitivities to these prompt configurations, all combinations of model, mask formats, and few-shot count were tested. The whole set of 80 manually annotated text answers (see Subsection 3.1.3) was employed for this experiment.

The results of this experiment, reported in Table 4.2 and summarized through aggregated analyses by model, mask formats, and few-shot count in Figure 4.3, indicate that these design choices have a critical and non-trivial impact on task performance. Models from the Gemma 3 family [78] achieve, on average, solid performance in both the 12B and 27B configurations. Mistral S 3.1 24B [59] exhibits moderate average performance, but with large variability across different settings. In contrast, Qwen2.5-VL 7B [7] yields inadequate

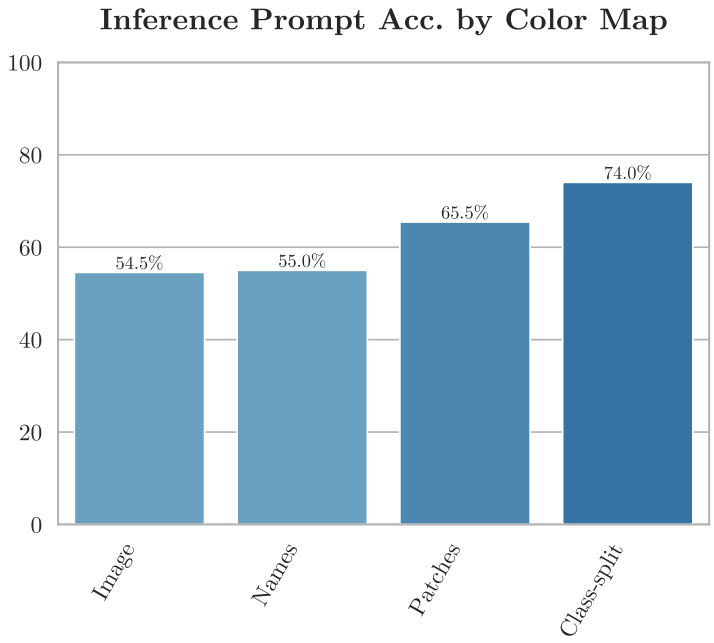



Figure 4.2: The plot shows the accuracy computed by the LLM-as-a-Judge in the non-class-split scenario, by color map format, and in the class-split scenario.

results in most evaluated configurations. The separate and concatenated mask formatting strategies result in comparable performance overall. Finally, few-shot prompting generally improves performance over the zero-shot setting, although using between one and three shots leads to similar accuracy levels.

| | **Separate** | | | | **Concat.** | | | |
|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** |
| **Qwen2.5VL 7B** [7] | 29.9 | 43.7 | 36.8 | 33.7 | 30.9 | 36.5 | 50.4 | 39.3 |
| **Gemma 3 12B QAT** [78] | 58.2 | **75.1** | 65.3 | 67.0 | 57.6 | 68.3 | 53.1 | 66.3 |
| **Mistral S 3.1 24B** [59] | 36.4 | 38.3 | 36.8 | 56.9 | 46.4 | 55.5 | 59.9 | 69.7 |
| **Gemma 3 27B** [78] | 58.5 | 70.0 | 68.3 | 72.1 | 67.3 | 66.0 | 70.5 | 63.9 |

Table 4.2: The table gathers all the accuracy values computed by the LLM-as-a-Judge over all the combinations of the tested MLLMs, mask formats and few-shot count.

Among the evaluated MLLMs, Gemma 3 12B QAT [78] offers the best trade-off between performance and computational cost. The aggregated results of the final experiment (Figure 4.4) indicate that the model benefits from the use of separate masks, while the number of shots has an inconsistent impact. Based on these findings, Gemma 3 12B

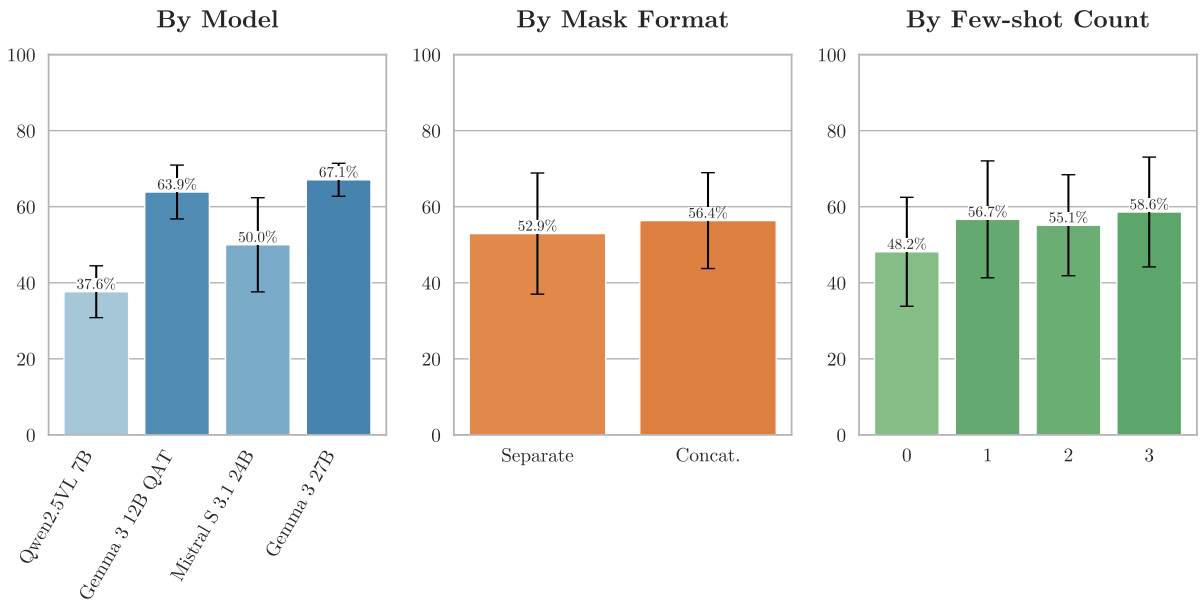



Figure 4.3: The plots show the accuracy obtained by aggregating the values in Table 4.2 by model, mask format, and number of few-shot examples.

QAT [78] was selected as the MLLM for subsequent experiments, using separate masks and a one-shot prompting setup with the mask of UID '2007_000256' (class 1). The setting, illustrated in Table 3.1, has also been adopted to generate the diagnostic texts for SDD, introduced in Subsection 3.2.1. An example of a fully formatted prompt built using these settings is provided and visualised in Appendix A.2.

## 4.2. Preliminary Experiments on FLAIR

This section presents the results of fine-tuning FLAIR [90], conducted before the full pipeline experiments, as motivated in Section 3.2. As described in Subsection 3.2.2, fine-tuning was performed by training an adapter module from scratch on top of the frozen text encoder outputs, keeping the original parameters fixed. To better adapt text representations to the smaller target data distribution while maintaining stable representations, the adapter was implemented as a single linear layer, limiting the number of trainable parameters and computational cost. The vision encoder, instead, was kept frozen, as it is not involved in the text-guided pipeline.

The model was fine-tuned on SDD (introduced in Subsection 3.2.1) using bfloat16 precision. The training protocol, shown in Table 4.3, largely mirrors the FLAIR training setup [90], except for batch size, number of epochs, and warmup steps. These hyperparameters were reduced to accommodate the shorter fine-tuning phase on a smaller dataset.

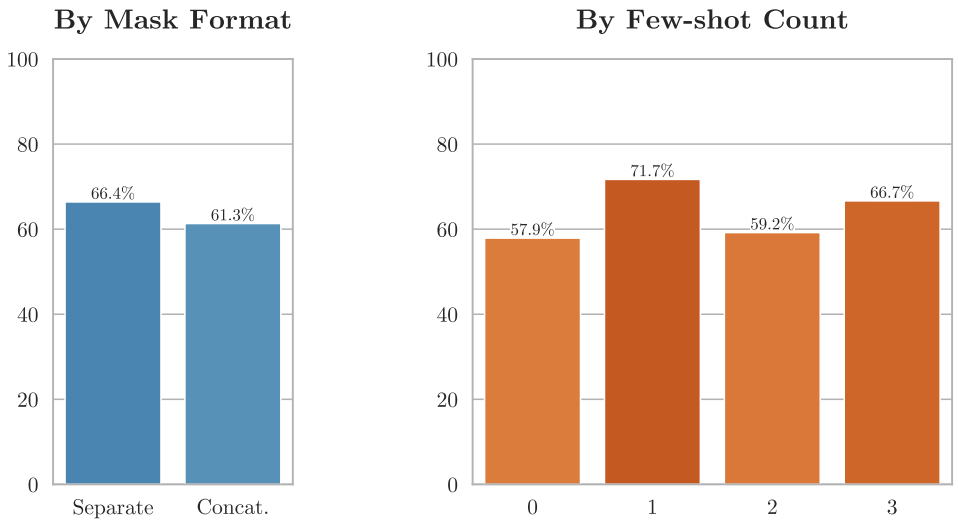



Figure 4.4: The plots show the accuracy obtained by aggregating the Gemma 3 12B QAT [78] values in Table 4.2 by mask format and by few-shot count.

Besides the optimizer's weight decay, no other data augmentation or regularization technique was adopted.

Model capability in encoding diagnostic text was evaluated through training metrics as well as qualitative inspection of attention maps, generated following the method proposed by the FLAIR authors [90] (see Subsection 2.3.1) on a subset of samples from SDD.

| Parameter | Value |
|---|---|
| **Base checkpoint** | CC3M-recap |
| **Image size** | $224 \times 224$ |
| **Epochs** | 16 |
| **Batch size** | 256 |
| **Optimizer** | AdamW($\beta_1 = 0.9\,, \beta_2 = 0.98\,, \varepsilon = 1 \times 10^{-8}$) |
| **Weight decay** | 0.5 |
| **Base LR** | $5 \times 10^{-4}$ |
| **LR schedule** | Cosine decay (100 warm-up steps) |
| **Max grad. norm** | 10.0 |

Table 4.3: FLAIR fine-tuning training protocol.

The training loss trends, shown in Figure 4.5, indicate that contrastive performance improves significantly during the initial epochs before rapidly plateauing and converging to

a final value of 5.1833. The validation loss exhibits a similar pattern but converges to a higher minimum of 5.3576, suggesting the presence of mild overfitting. This behavior is consistent with the adapter-based fine-tuning strategy, which helps limit overfitting while simultaneously restricting the model's capacity to further adapt. Notably, the contrastive loss values, recorded in Table 4.4, demonstrate a remarkable improvement compared to the pretrained model. This result confirms that the fine-tuning procedure significantly enhances the contrastive capabilities of the text encoder and, consequently, its ability to produce meaningful representations from the diagnostic texts.

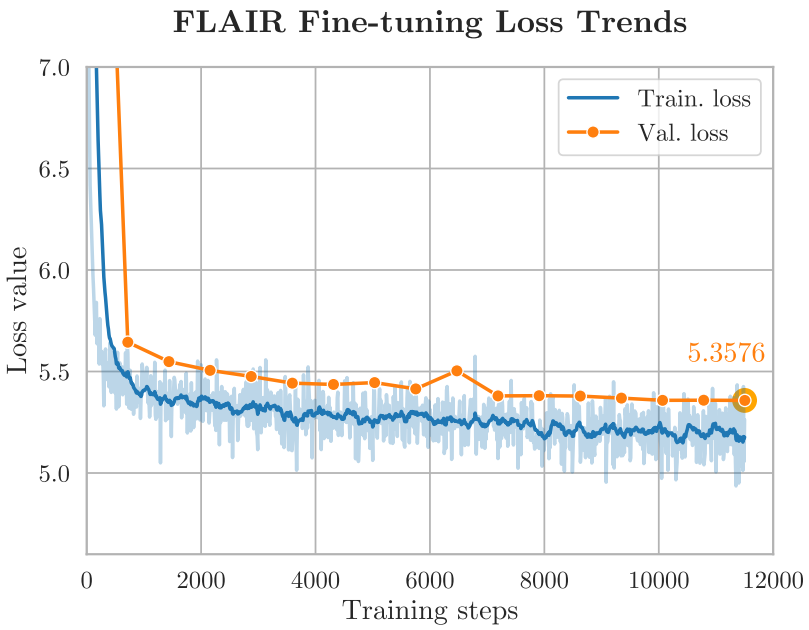



Figure 4.5: The plot displays the trends of FLAIR training and validation losses observed during fine-tuning.

| Ckpt | Train. loss | Val. loss |
|---|---|---|
| Pre-trained | 11.0826 | 11.0529 |
| Fine-tuned | 5.1833 | 5.3576 |

Table 4.4: The training loss and validation loss computed on the respective SDD splits, by model.

Overall, the attention maps, available in the public repository of this work (specified at the beginning of this chapter), with some examples visualized in Appendix A.3, tend to exhibit improved precision and visual clarity, although in some samples the improvement is less apparent.

## 4.3. Language-Guided Pipeline Experiments

This section presents the experimental setup and the experiments of the language-guided pipeline, and analyze the results.

## Experimental Setup

The pipeline experiments assessed the training procedure of encoder-decoder segmentation models on the VOC2012 dataset [24], within the language-guided pipeline described in Section 3.3. The official training and validation splits were employed. Following the VOC2012 training protocol [23, 24], the UNLABELLED class was ignored in the CE loss computation, as the segmentation models are not designed to predict it.

As discussed in Subsection 3.3.2, the segmentation models first underwent a pre-language training phase without the language-guided mechanism. For a fair comparison, this phase was tailored for each model to ensure comparable starting validation mIoU performances. To avoid cache cold start issues, as outlined in Subsection 3.3.4, the cache was prefilled at the beginning of training. The model-dependent training parameters are displayed in Table 4.5, both for the pre-language and language-guided settings. Each model was trained using center-cropped images matching its input resolution.

| Seg. Model | Image size | Epochs | Batch Size | Base LR |
|---|---|---|---|---|
| LR-ASPP-MobileNetV3-Large [31] | $520 \times 520$ | 4/50 | 8 | $10^{-4}$ |
| | $520 \times 520$ | 25 | 32 | $10^{-4}$ |
| FPN-ResNet18 [39] | $512 \times 512$ | 2/50 | 8 | $10^{-4.5}$ |
| | $512 \times 512$ | 25 | 32 | $10^{-4}$ |
| DeepLabV3-ResNet18[13] | $520 \times 520$ | 2 | 8 | $10^{-4.5}$ |
| | $520 \times 520$ | 25 | 32 | $10^{-4.5}$ |
| FPN-ResNet34 [39] | $512 \times 512$ | 1/50 | 8 | $10^{-4}$ |
| | $512 \times 512$ | 25 | 32 | $10^{-4}$ |
| DeepLabV3-ResNet34[13] | $520 \times 520$ | 1/50 | 8 | $10^{-4}$ |
| | $520 \times 520$ | 25 | 32 | $10^{-4}$ |

Table 4.5: The table lists both pre-language (in the topmost row) and language-guided (in the bottommost row) training settings. The pre-language epochs indicate how many epochs have been completed out of the total number.

The model-agnostic training setting parameters are displayed in Table 4.6. As discussed in Subsection 3.3.4, to maintain the effectiveness of the subsampling strategy, I opted to use dropout (with each model using its original dropout rate) as the sole regularization technique during training, in addition to the optimizer's weight decay. No data augmentation strategy has been employed. The experiment investigating the impact of regularization techniques on cached mask variability is presented in Appendix B.

For each segmentation model, the baseline is trained from the pre-language checkpoint using the same settings but without language guidance. This setup omits the MLLM, FLAIR [90], the bottleneck adapter, and the contrastive loss, retaining only the segmen-

tation model with the standard CE loss, as in a typical segmentation scenario.

| Parameter | Value |
|---|---|
| **Regularizers** | Dropout |
| **Optimizer** | AdamW($\beta = (0.9, 0.999)$, $\varepsilon = 1 \times 10^{-8}$) |
| **Weight decay** | $10^{-0.5}$ |
| **LR schedule** | Cosine decay (200 warm-up steps) |
| **Max grad. norm** | 10.0 |
| **Multi-task setup** | Convex comb. |
| **Contr. neg.s** | In-batch |
| **Bn. adapter** | `GAP+linear (text-side)` |
| **Cache retention perc.** | 0.4 |

Table 4.6: The table lists the training setup for the pipeline experiments.

For the MLLM, I used Gemma 3 12B QAT [78], prompted according to the settings outlined in Subsection 4.1.2 and summarized in Table 3.1. For the VLE, I employed the FLAIR [90] fine-tuned checkpoint selected in Subsection 4.1.2. Both modules are executed in the experimental setups respectively explicited in Subsection 4.1 and Subsection 4.2. All the segmentation models and FLAIR operated at bfloat16 precision.

As configured by the segmentation task, the primary evaluation metric is the mIoU. Additional assessed metrics include the CE and contrastive loss values, their $\lambda$-normalized ratio as an indicator of balance between the two objectives, and the gradient norm magnitude as a measure of training stability. For each experiment, the analysis covers segmentation and contrastive metrics, their interplay, gradient norm trends, and concludes with a discussion of the results. Unless stated otherwise, contrastive metrics were not computed on the validation set to improve computational efficiency.

The training metrics shown for each experiment are computed using simple exponential smoothing with a factor of 0.9 applied to the batch metrics from each training step. Validation metrics are measured at the start of training and after each epoch, then aligned with the training steps for visualization. In the experimental plots, solid lines represent novel results, while dotted lines show previous results for comparison. Some plots use a logarithmic scale to facilitate visual comparison between runs.

### 4.3.1. Analysis of Language-Guided Pipeline

The experiments presented in this subsection evaluate the pipeline's behavior across different scenarios by analyzing segmentation and contrastive trends as the newly introduced training parameters vary.

## Lambda Analysis

This experiment investigates the effect of the contrastive weighting factor $\lambda$ on the training process. In this multi-task convex combination scenario, increasing $\lambda$ amplifies the weight of the contrastive loss while correspondingly reducing the weight of the CE loss.

The segmentation metrics, shown in Figure 4.6, indicate that as $\lambda$ increases, the progression of both CE and mIoU deteriorates, although the overall trends remain similar. Consequently, the language-guided model fails to match the baseline performance. Moreover, the baseline model exhibits clear signs of overfitting, with a noticeable gap between training and validation metrics. However, the introduction of the language-guided pipeline does not effectively mitigate this behavior: although it lowers the training performance, it also leads to a comparable degradation in validation metrics. As a result, no tangible generalization improvement is observed.

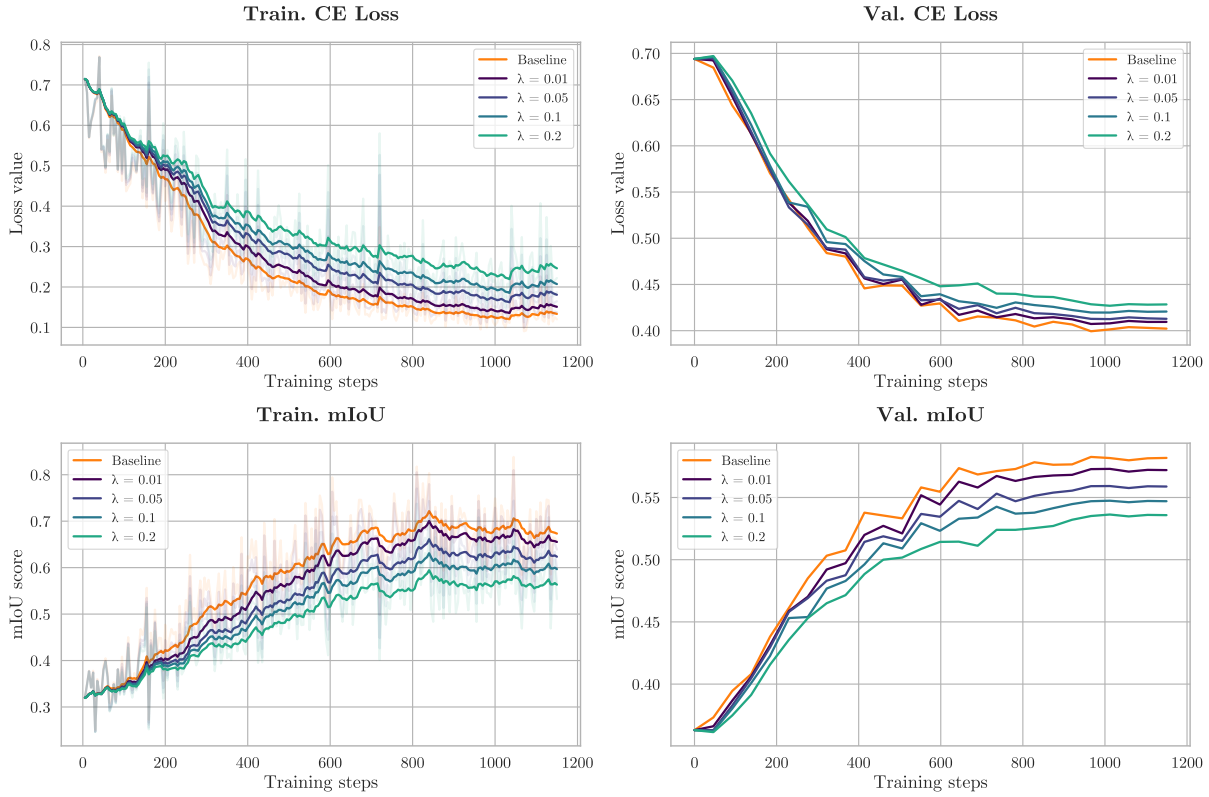



Figure 4.6: The plot illustrates how the segmentation metrics vary with different values of $\lambda$.

The contrastive metrics, presented in Figure 4.7, behave as expected: the contrastive loss is better optimized for higher $\lambda$ values. Its trend converges earlier than the CE loss, and the unnormalized values remain significantly larger, suggesting that the alignment task is much more challenging than the segmentation one, in this setting. The ratio of

the two losses stabilizes quickly during training, indicating a consistent balance between them. However, the gradient norm increases sharply with $\lambda$, pointing to instability and irregularities in the optimization process.

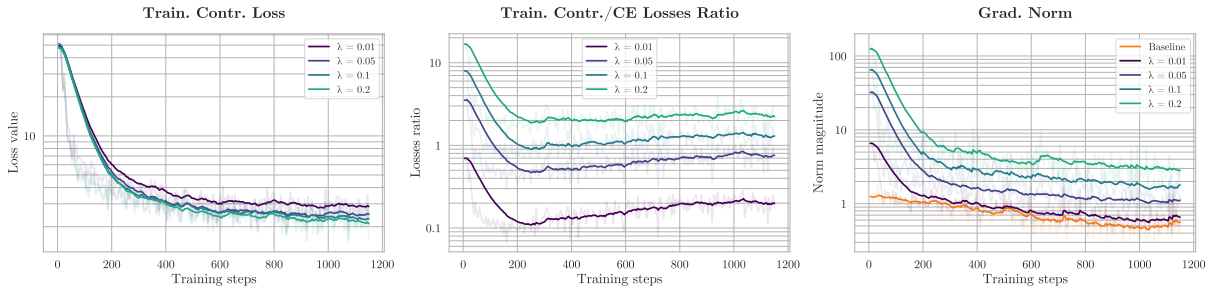



Figure 4.7: The plot illustrates how the contrastive metrics and the gradient norms vary with different values of $\lambda$.

Overall, these results suggest that the contrastive loss struggles to complement the CE loss and may introduce instability and dissonance in training as its influence grows.

## Synthetic Negatives Analysis

This experiment examines the impact of synthetic contrastive negatives (see Subsection 3.3.3) on the training process, with different values of $\lambda$. The word pools adopted in the first substitution stage of the synthetic negatives generation algorithm are visualized in Appendix A.4. To ensure a fair comparison with experiments using in-batch negatives, the number of generated negatives was matched to the number of in-batch negatives in the same setting. The results are directly compared with those of the prior experiment, which are shown as dotted lines in the plots.

The segmentation metrics, shown in Figure 4.8, indicate that introducing synthetic negatives into the contrastive loss degrades both the CE losses and the mIoU, compared to using in-batch negatives. Moreover, this negative effect becomes more pronounced as $\lambda$ increases. Overall, all metrics exhibit a trend similar to that observed with in-batch negatives.

Similar observations hold for the contrastive metrics, shown in Figure 4.9, which display a noticeable deterioration. In particular, the contrastive loss is considerably higher, indicating that the synthetic negatives make contrastive optimization more difficult. Consequently, the loss ratio increases, though it remains comparable to previous results, while the gradient norm shows a significant increase, suggesting that synthetic negatives amplify the instability of the training optimization.

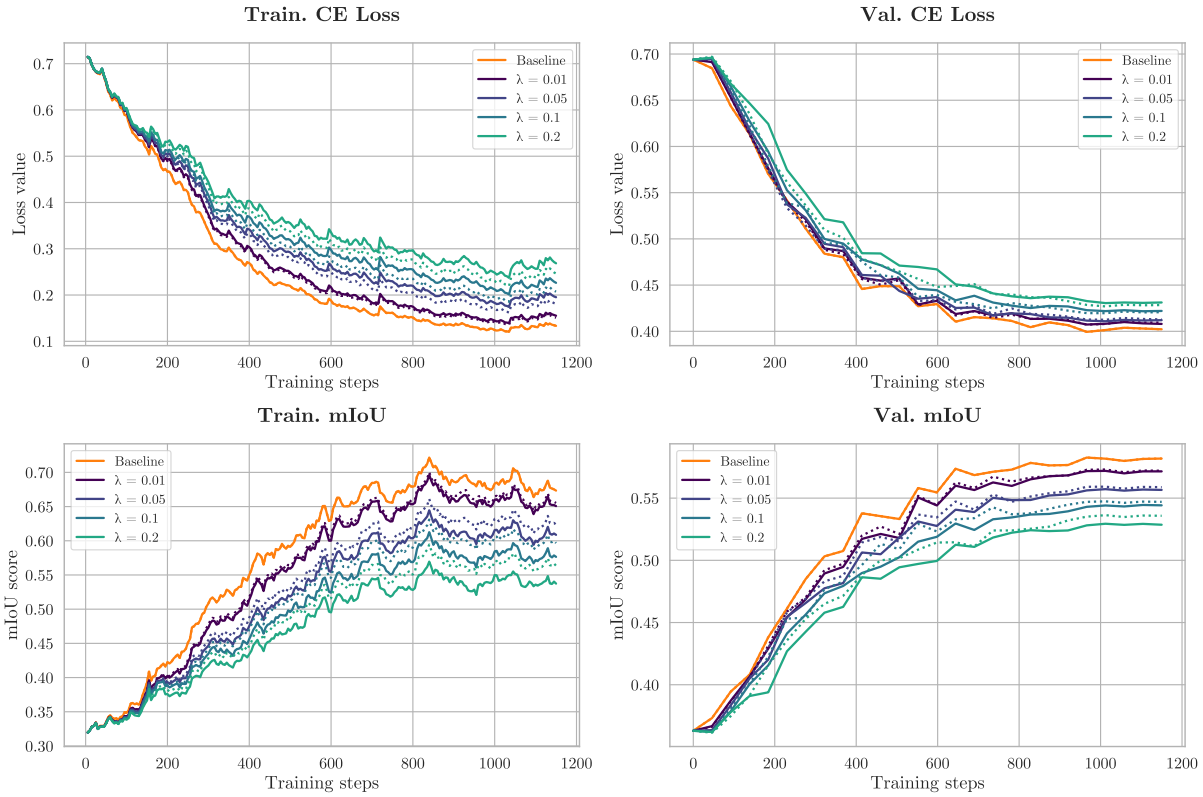



Figure 4.8: The plot illustrates how the segmentation metrics vary using synthetic contrastive negatives. The dotted lines represent the in-batch counterpart results from the prior experiment.

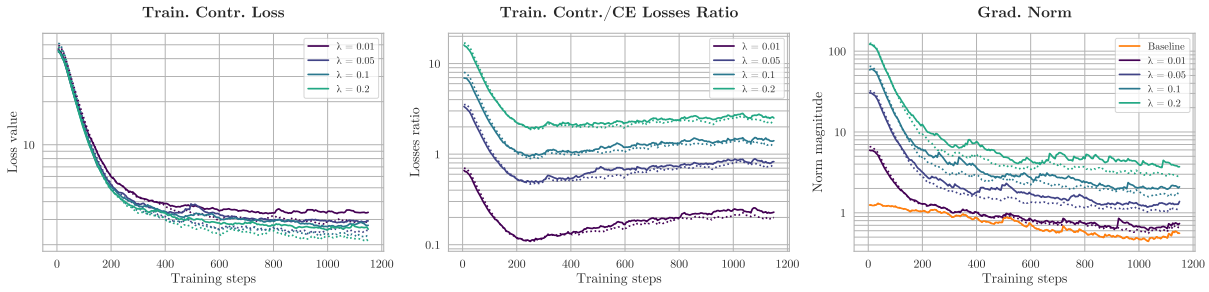



Figure 4.9: The plot shows the contrastive metrics and the gradient norm trends using synthetic contrastive negatives. The dotted lines refer to the in-batch counterpart results from the prior experiment.

The results indicate that the contrastive optimization process becomes more challenging when synthetic negatives are used. Since these negatives are deliberately constructed to be hard, i.e., more similar to the positive samples, they are inherently more difficult to distinguish in the embedding space. As a consequence, the model struggles to separate positives from negatives, leading to a higher contrastive loss and increased optimization difficulty. This additional challenge amplifies training instability and ultimately degrades the segmentation performance.

### Bottleneck Adapters Analysis

This experiment investigates how different bottleneck adapter designs (see Subsection 3.3.5) affect training, using in-batch negatives and $\lambda = 0.1$.

As shown by the segmentation metrics in Figure 4.10, the adapter choice has a clear impact on performance. Linear-based adapters achieve the lowest scores and behave very similarly to each other. In contrast, both convolutional and attention-based adapters yield a consistent improvement across all segmentation metrics, approaching the baseline performance. Among the language-guided configurations, attention-based adapters tend to perform best overall, while convolutional adapters follow closely, with a slight gap in mIoU.

The contrastive metrics in Figure 4.11 show that the `conv+AttnPoolByText` module optimizes the contrastive loss much more effectively, and continues improving it until the end of training, even after the other modules have converged for several epochs. Its loss ratio remains constant and significantly lower than that of the other adapters, indicating a more balanced and effective optimization of both losses, driven by a superior contrastive learning process.

The remaining adapters exhibit similar behavior overall. Among them, the `convX2+GAP` and `conv+AttnPoolByText` modules perform slightly better than the linear-based adapters in optimizing the contrastive loss, but they display a higher loss ratio, indicating an optimization process more skewed toward the contrastive objective.

The linear-based modules exhibit the largest gradient norms, while the convolution-based module shows the smallest. Attention-based modules fall in between, despite achieving the most effective contrastive loss optimization, likely due to their higher module complexity.

This experiment highlights adapter complexity as a key factor in optimization efficacy for both segmentation and contrastive objectives. Exploiting spatial structure in the bottleneck adapters consistently improves performance across both tasks, whereas class-

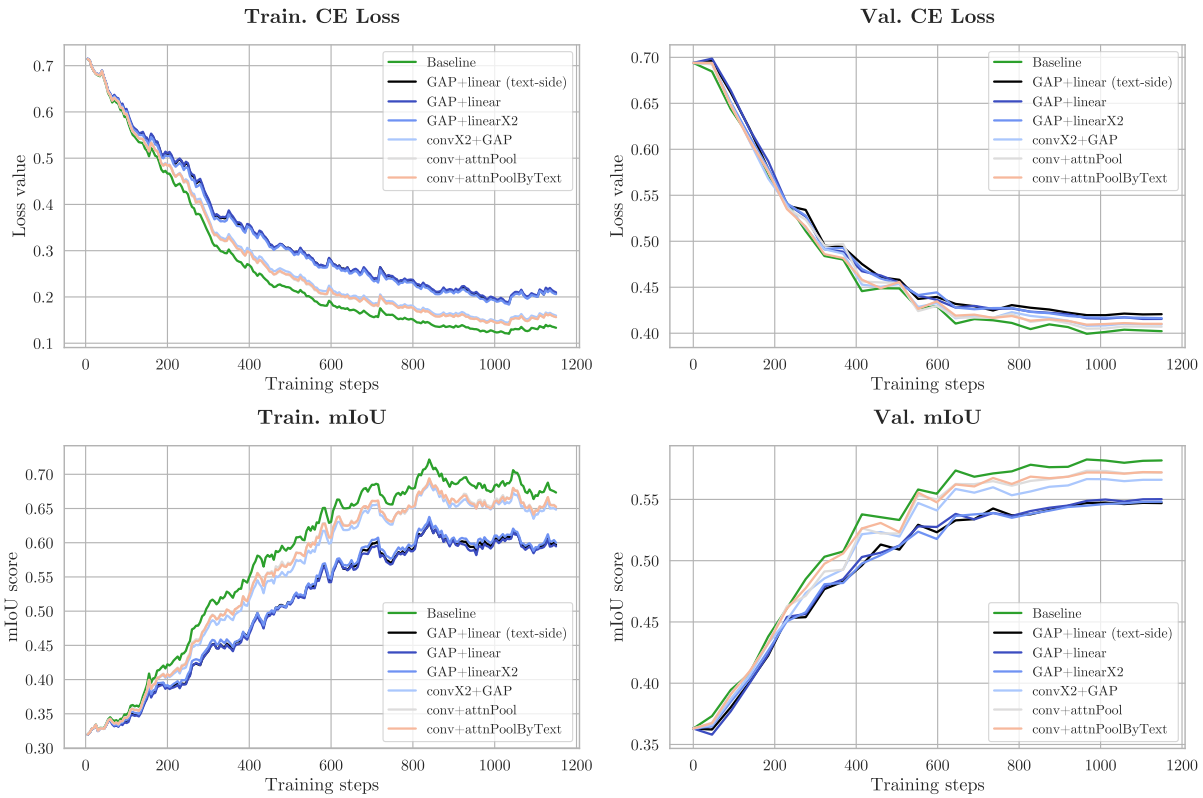



Figure 4.10: The plot illustrates how the segmentation metrics vary with different bottleneck adapter modules.

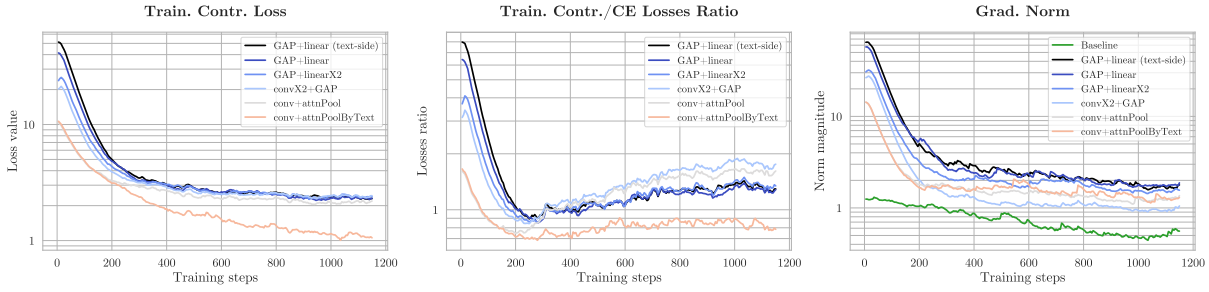



Figure 4.11: The plot shows the contrastive metrics and the gradient norm trends using different bottleneck adapter modules.

conditioned bottlenecks substantially enhance contrastive loss optimization with more modest gains in segmentation metrics. Gradient norm magnitude, however, shows a non-trivial behavior: it appears to improve with better contrastive optimization but degrades as adapter complexity increases, while remaining higher than the baseline overall.

## Segmentation Models Analysis

This experiment explores how various encoder-decoder segmentation models respond to the language-guiding mechanism during training. The models were evaluated in an auxiliary multi-task setup, with the $\lambda$ value increased to 0.5 to emphasize the impact of language-guided conditioning. A `conv+AttnPoolByText` module implements the bottleneck adapter. The resulting metrics, based on the optimal values obtained during training, are presented in Table 4.7.

| | CE Loss | | mIoU | | Contr. Loss |
|---|---|---|---|---|---|
| | **Train.** | **Val.** | **Train.** | **Val.** | **Train.** |
| LR-ASPP-MobileNetV3 [31] | 0.1360 (+0.046) | 0.4239 (+0.024) | 73.25 (-10.58) | 54.84 (-3.42) | 0.478 |
| FPN-ResNet18 [39] | 0.0509 (+0.022) | 0.5122 (-0.024) | 85.84 (-5.80) | 52.88 (+0.72) | 0.346 |
| DeepLabV3-ResNet18 [13] | 0.0627 (-0.065) | 0.3877 (-0.063) | 92.86 (+0.19) | 59.77 (-1.00) | 0.390 |
| FPN-ResNet34 [39] | 0.0330 (+0.013) | 0.4674 (+0.002) | 89.74 (-5.07) | 58.74 (+0.07) | 0.345 |
| DeepLabV3-ResNet34 [13] | 0.1402 (+0.030) | 0.4326 (-0.016) | 90.99 (-2.59) | 62.24 (-1.86) | 0.779 |

Table 4.7: The table gathers the optimal CE loss, mIoU, and contrastive loss values of the segmentation models tested in the language-guided pipeline, highlighting the change compared to their baseline. Green → better, red → worse.

As illustrated by the segmentation metrics in Figure 4.12, the models generally exhibited worse trends in both CE loss and mIoU compared to their baselines, with varying degrees of severity. Both training metrics mostly reflected a performance decline, with occasional minor improvements. Regarding validation mIoU, LR-ASPP-MobileNetV3 experienced the largest drop (-3.42), followed by DeepLabV3 models with ResNet34 (-1.86) and ResNet18 (-1.00) encoders. In contrast, FPN models with ResNet18 and ResNet34 encoders showed very modest gains (+0.72 and +0.07, respectively).

Regarding the contrastive loss trends shown in Figure 4.13, DeepLabV3-ResNet34 exhibited the poorest optimization, followed by LR-ASPP-MobileNetV3, while the other models all achieved similar loss values.

The experimental results suggest that the language-guiding mechanism generally degrades or preserves the segmentation performance of the models, with the effect varying across models. The relationship between architectural characteristics and the effectiveness of

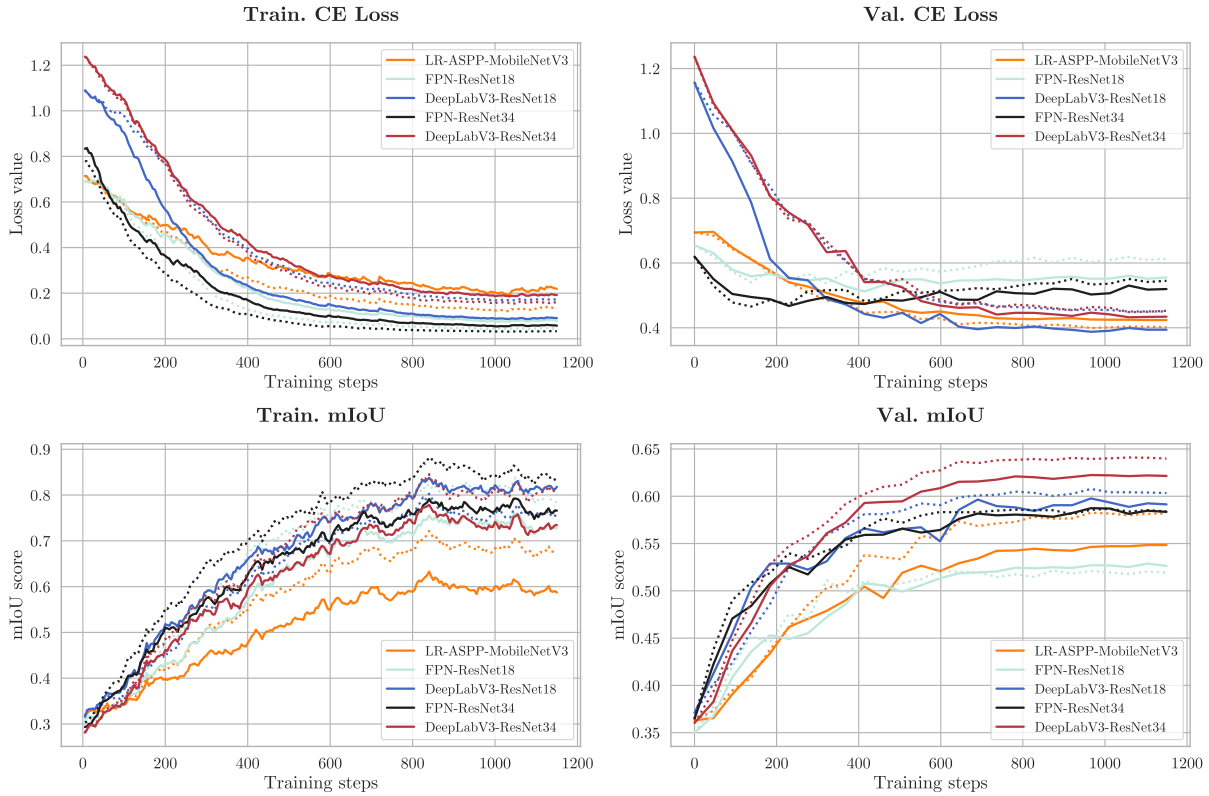



Figure 4.12: The plot illustrates the segmentation metrics of different language-guided segmentation models.

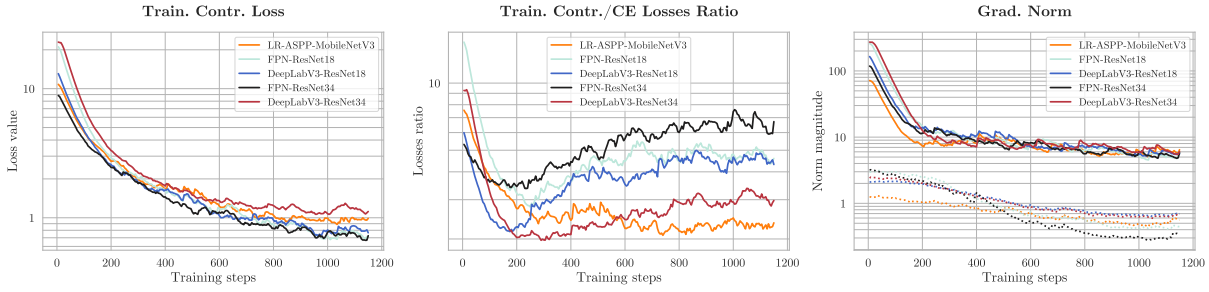



Figure 4.13: The plot illustrates the contrastive metrics and the gradient norm trends of different language-guided segmentation models.

contrastive optimization remains inconclusive.

## Data Subsampling Analysis

This experiment analyzes the effect of the subsampling strategy on the training procedure, varying the subsampling percentage $p$. As discussed in Subsection 3.3.4, to ensure a fair comparison across runs, synthetic contrastive negatives were employed, with each run employing the same number of negatives as in the case of in-batch negatives, ensuring that loss values are directly comparable. The weighting factor $\lambda$ was set to 0.1.

Across the tested range of $p$, lower percentages, corresponding to more aggressive subsampling, consistently led to worse performance in both segmentation and contrastive metrics, as shown in Figures 4.14 and 4.15. While the overall trends are similar across metrics, the performance degradation increases more than linearly as $p$ decreases, whereas higher values of $p$ tend to converge to comparable performance levels.

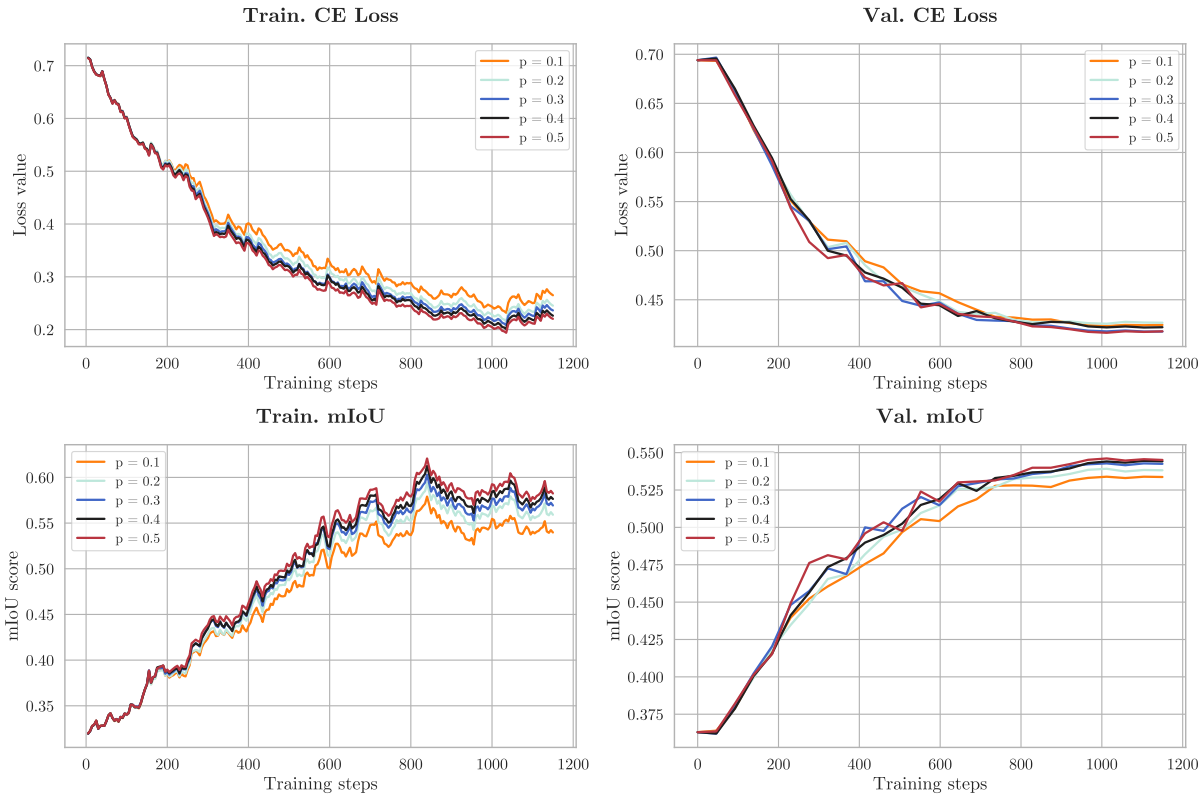



Figure 4.14: The plot illustrates how the segmentation metrics vary with different subsampling percentages.

Notably, very low values of $p$ result in significantly less stable gradient norms, characterized by larger magnitudes and more irregular behavior. A similar instability is observed

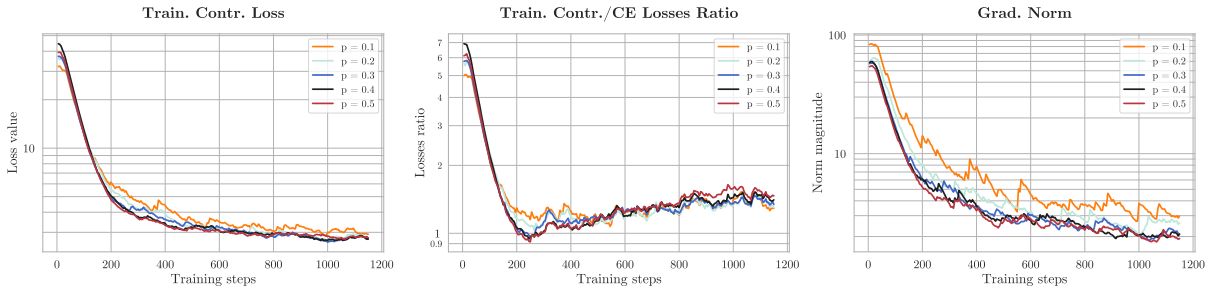



Figure 4.15: The plot shows the contrastive metrics and the gradient norm trends using different subsampling percentages.

in the contrastive loss, whereas the segmentation metrics do not reflect the same level of variability.

Overall, the experiment shows that more aggressive subsampling incurs reduced performance and increased training instability, suggesting a strong correlation between these two phenomena.

### 4.3.2. Ablation Studies

The experiments in this subsection evaluate the pipeline's behavior by performing ablations on the modules of the pipeline, analyzing both segmentation and contrastive trends.

#### FLAIR Fine-tuning Ablation

This experiment ablates the impact of the SDD fine-tuning of FLAIR [90] (see Section 4.2) during training. The losses were aggregated in an auxiliary multi-task setup, with $\lambda = 0.1$. The bottleneck adapter is implemented via a `conv+AttnPoolByText` module.

As shown by the segmentation metrics in Figure 4.16, fine-tuning yields slightly improved performance, although the gains are very modest. Also, fine-tuning appears to contribute to more stable training behavior.

In contrast, the contrastive metrics in Figure 4.17 indicate that fine-tuning negatively affects contrastive optimization. Specifically, it causes the optimization trend to converge earlier and at a higher value, while producing lower-magnitude but more stable gradient norms.

These results pinpoint that fine-tuning substantially harms the contrastive objective while slightly benefiting the segmentation task, suggesting an inverse correlation between the two.

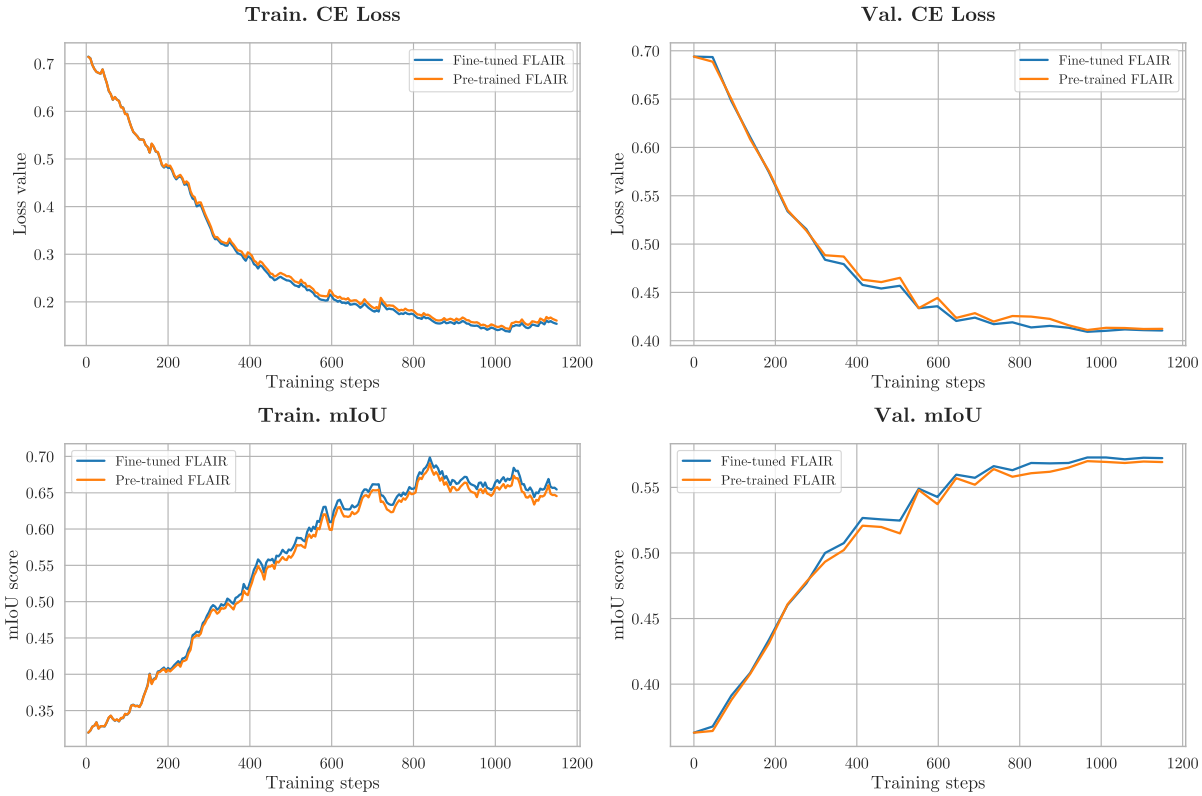



Figure 4.16: The plots illustrate how the FLAIR fine-tuning affects the segmentation metrics trends.

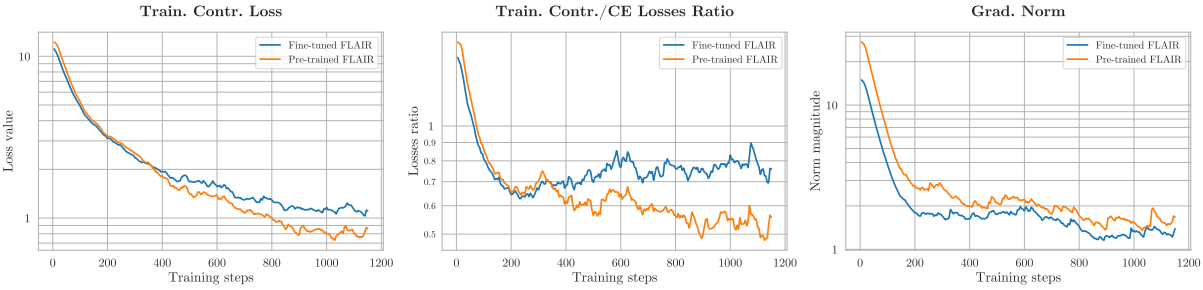



Figure 4.17: The plots show how the FLAIR fine-tuning affects the contrastive metrics and the gradient norm trends.

### Subsampling Ablation and Validation Contrastive Loss Analysis

This experiment investigates the impact of the subsampling strategy on the training process by disabling subsampling entirely, setting the cache retention percentage to $p = 1.0$. Consequently, all samples are used for the language guidance. The experiment also evaluates the validation metrics of the contrastive loss, which is faithfully computed in this scenario.

The runs were performed in an auxiliary multi-task setup with the $\lambda$ factor set to 0.1. The bottleneck adapter is implemented using a `conv+AttnPoolByText` module, and in-batch negatives are employed. The unsampled run ($p = 1.0$) is compared against the baseline and against a run with a subsampling retention percentage of $p = 0.4$ (the default).

In this context, as described in Subsection 3.3.4, the unsampled training and validation contrastive losses are directly comparable. However, training contrastive metrics are not directly compared across runs with different values of $p$.

The segmentation metrics, shown in Figure 4.18, indicate that removing subsampling slightly narrows the already small gap with the baseline, although performance remains slightly lower than the baseline.

The gradient norms without subsampling, displayed in Figure 4.19, are slightly but consistently lower than those for $p = 0.4$, yet they remain considerably higher than the baseline. The contrastive metrics indicate that the pipeline experiences low overfitting, even when using the strongest bottleneck adapter implementation, and that the training and validation contrastive losses follow similar trends. In contrast, the validation loss ratio is significantly lower than the training loss ratio: while the contrastive losses are comparable, the validation cross-entropy loss is much higher than the training one. Moreover, the trends of the two ratios diverge, with the validation loss ratio steadily decreasing, whereas the training loss ratio gradually increases.

This experiment suggests that using moderate subsampling retention percentages $p$ provides an effective trade-off between performance and computational efficiency, substantially reducing time complexity while preserving most of the performance. It also indicates that the alignment task is challenging for the pipeline: even with the strongest bottleneck adapter, overfitting is mild, and the contrastive loss remains considerably higher than the CE loss.

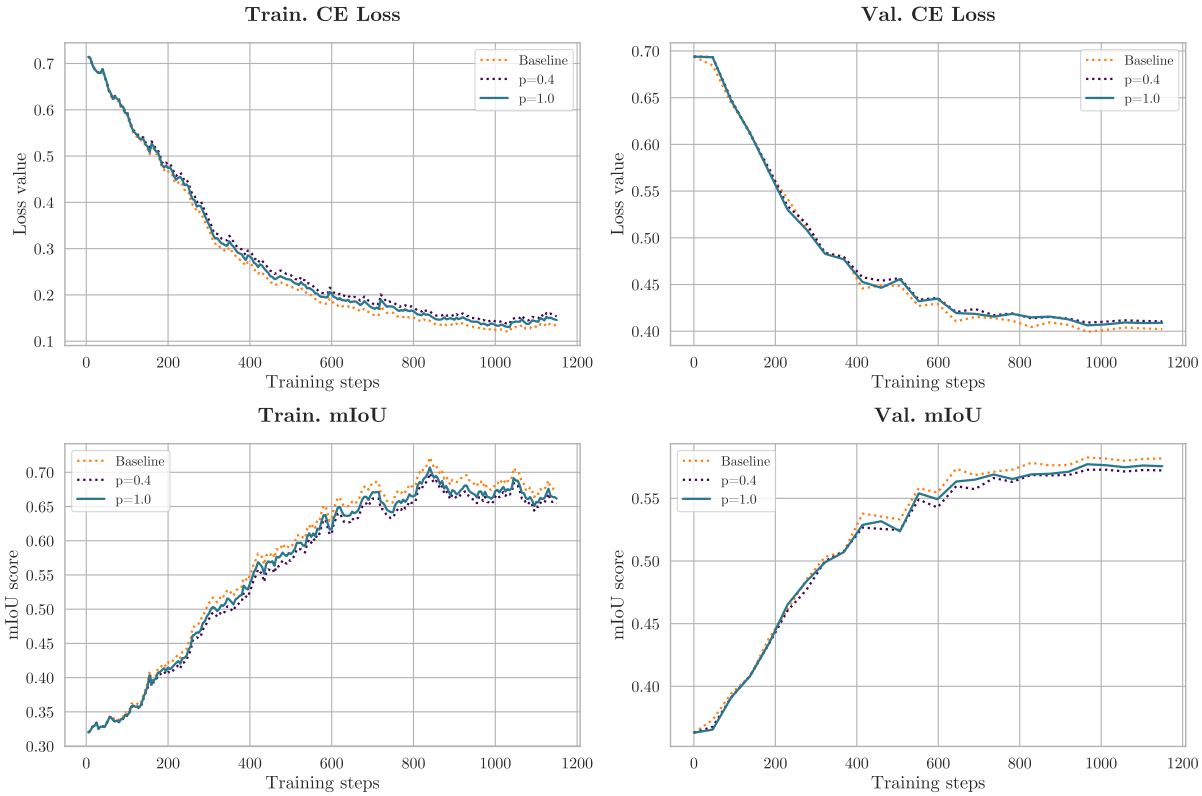



Figure 4.18: The plot illustrates how ablating the subsampling strategy affects the segmentation metrics.

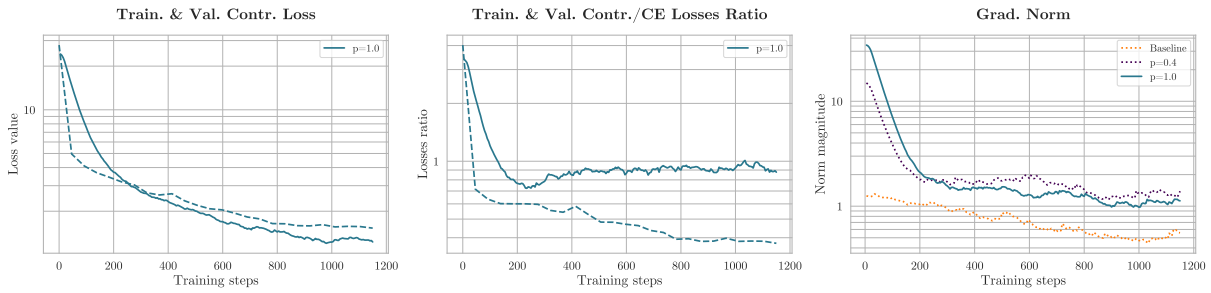



Figure 4.19: The plot shows how ablating the subsampling strategy affects the contrastive metrics and the gradient norm trends.

## Segmentation Encoder Detachment Ablation

This experiment investigates the effect of detaching the segmentation encoder from the contrastive loss. In this setup, the contrastive objective does not update the encoder, which is trained solely by the CE loss. Instead, the contrastive loss is entirely optimized by the bottleneck adapter. Consequently, the segmentation model is trained as in the baseline non-language-guided scenario, with the bottleneck adapter independently optimizing the contrastive loss in parallel. The losses are combined in a multi-task auxiliary setup with $\lambda = 0.1$. The bottleneck adapter is implemented by a `conv+AttnPoolByText` module.

Figure 4.20 shows that detaching the encoder has minimal impact on contrastive loss optimization: the loss increases only slightly while following a similar trend. Gradient norm analysis also reveals comparable behavior, with the detached run exhibiting lower gradients.

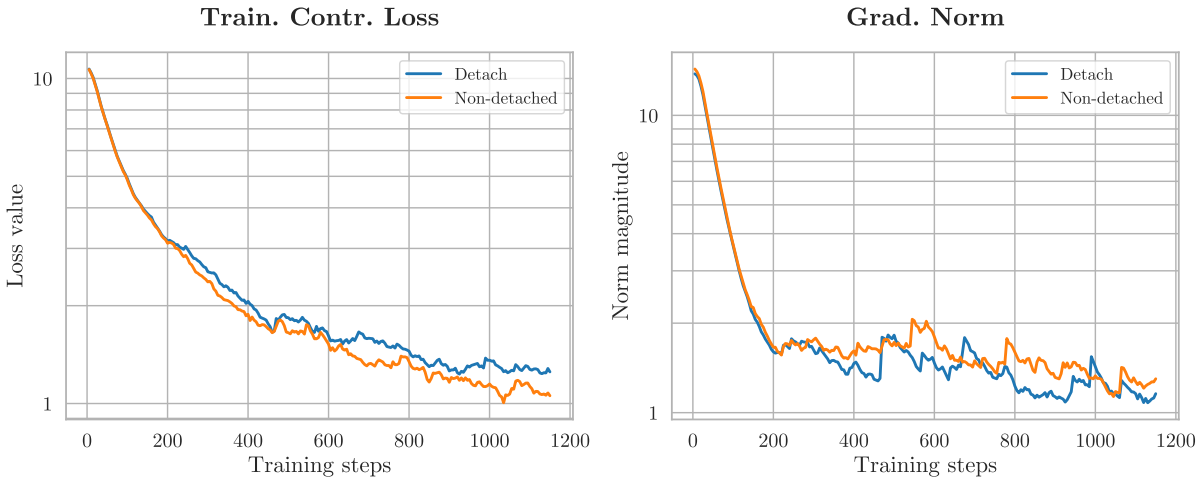



Figure 4.20: The plots show the contrastive loss and gradient norm trends resulting from the encoder detachment from the contrastive loss.

The metrics indicate that the encoder plays only a minor role in contrastive optimization, which proceeds effectively even when optimized solely by a sufficiently capable bottleneck adapter. The gradient norms also suggest that detaching the encoder improves the overall training stability. The observed drop in segmentation performance when employing the encoder language guidance suggests an inverse relationship between the segmentation and contrastive objectives.

# 5 | Conclusions and Future Work

This concluding chapter reviews the primary outcomes of the thesis and discusses their relevance in the broader context of language-supervised SS. Furthermore, it addresses the limitations and open challenges, and suggests potential directions for future research.

## 5.1. Conclusions

This work investigates whether a language-guided pipeline can enhance the robustness of segmentation models by incorporating a contrastive loss grounded in textual supervision. The method extends standard segmentation architectures with modules that produce and encode diagnostic textual descriptions related to model behavior. Alongside the CE loss, a contrastive objective aligns these textual embeddings with adapted intermediate features from the segmentation encoder, encouraging representations to be anchored in meaningful diagnostic information.

Experimental results show that the language-guided pipeline generally fails to outperform baseline segmentation models. The results suggest an inverse relationship between the segmentation and contrastive objectives. Increasing the contrastive weight $\lambda$ consistently degrades the segmentation performance and introduces significant training instability, evidenced by higher gradient norms. This indicates that the two objectives compete rather than cooperate, likely contending for the encoder's representational capacity.

The bottleneck adapter studies provide crucial insights into the interplay between the encoder and the bottleneck adapter. The contrastive loss proves inherently challenging and degrades segmentation performance in proportion to the encoder's participation in its optimization. This explains why stronger bottleneck adapters yield better metrics: their increased complexity allows them to absorb most of the linguistic alignment burden. By shifting contrastive optimization away from the encoder, interference with its primary segmentation task is reduced. This is further supported by the encoder detachment ablation, which shows that the bottleneck adapter can largely handle the alignment task independently.

This task interference dynamic is further illuminated by the FLAIR fine-tuning ablation, which yields a seemingly counterintuitive result: contrary to expectations, fine-tuning the text encoder harms the contrastive loss optimization, yet improves segmentation performance. By adapting the text encoder to SDD, the text embeddings likely become over-specialized, complicating the contrastive loss landscape. As a result, the contrastive objective plateaus in a poorer local minimum, producing smaller, more stable gradients. With its influence reduced, the CE loss no longer competes for control of the encoder's weights, allowing the model to focus more effectively on the primary segmentation task and ultimately achieve better performance.

These findings also clarify the behavior observed in the negative sampling and subsampling strategies. Since the contrastive objective is already prone to interfering with segmentation, design choices that further complicate its optimization exacerbate instability. In particular, synthetic negatives make contrastive alignment harder, thereby amplifying training instability and proving to be less effective than in-batch negatives. Similarly, while moderate data subsampling reduces computational cost without altering the training dynamics, excessively aggressive subsampling disrupts the optimization balance, leading to non-linear performance degradation and unstable behavior.

Overall, although advanced adapters can mitigate some instability, the segmentation encoder appears poorly suited to jointly optimize pixel-level classification and diagnostic text alignment. The latter introduces competition and instability that ultimately undermine segmentation performance. These findings reveal a gap between segmentation and linguistic representations, highlighting the need for innovative, synergistic multi-modal approaches that encourage cooperation rather than competition between pixel-level and textual objectives. Closing this gap will likely require mechanisms that integrate both modalities without overwhelming the encoder's capacity, or the development of specialized representation pathways for the segmentation task across different modalities.

## 5.2. Limitations and Future Work

Despite the methodological rigor of this work, several limitations affect the generalizability of the findings.

First, the methods and experiments primarily leveraged the VOC2012 and COCO2017 datasets and a limited set of segmentation models. More extensive experiments would require the use of additional datasets of diverse types, and models with different architectures and scales.

Second, due to the significant scarcity of publicly available datasets and resources in this research field, diagnostic text generation analyses relied on a limited set of internally sourced annotations, which may reflect the biases of the few annotators involved. More generalizable results would require a larger number of annotations across diverse datasets and from a broader pool of annotators.

Third, the studies primarily evaluated the interaction between the contrastive loss and the CE loss using the mIoU metric. Exploring alternative loss functions and evaluation metrics could provide more generalizable insights. Moreover, complementing these quantitative analyses with explainability techniques (e.g., attention or saliency maps visualization) would allow for a qualitative assessment, capturing aspects that numerical metrics might fail to reflect.

These points already indicate valuable directions for future research. Furthermore, additional work could focus on refining the diagnostic text generation stage by exploring more effective generation methods and testing more advanced models. Significant contributions could arise from exploring alternative approaches for language-guided supervision, ultimately paving the way for a more symbiotic interaction between language and vision modalities.

# Bibliography

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[2] P. Agrawal, S. Antoniak, E. B. Hanna, B. Bout, D. Chaplot, J. Chudnovsky, D. Costa, B. De Monicault, S. Garg, T. Gervet, et al. Pixtral 12b. arXiv preprint arXiv:2410.07073, 2024.

[3] F. Almeida and G. Xexéo. Word embeddings: A survey. arXiv preprint arXiv:1901.09069, 2019.

[4] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. Vqa: Visual question answering. In Proceedings of the IEEE international conference on computer vision, pages 2425–2433, 2015.

[5] R. Azad, M. Heidary, K. Yilmaz, M. Hüttemann, S. Karimijafarbigloo, Y. Wu, A. Schmeink, and D. Merhof. Loss functions in the era of semantic segmentation: A survey and outlook. arXiv preprint arXiv:2312.05391, 2023.

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence, 39(12):2481–2495, 2017.

[7] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, et al. Qwen2. 5-vl technical report. arXiv preprint arXiv:2502.13923, 2025.

[8] L. Bianchi, F. Carrara, N. Messina, and F. Falchi. Is clip the main roadblock for fine-grained open-world perception? In 2024 International Conference on Content-Based Multimedia Indexing (CBMI), pages 1–8. IEEE, 2024.

[9] K. Blagec, G. Dorffner, M. Moradi, and M. Samwald. A critical analysis of metrics used for measuring progress in artificial intelligence. arXiv preprint arXiv:2008.02577, 2020.

[10] R. Bommasani. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258, 2021.

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[12] S. Chang and P. Ghamisi. Changes to captions: An attentive network for remote sensing change captioning. IEEE Transactions on Image Processing, 32:6047–6060, 2023.

[13] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587, 2017.

[14] Z. Chen, Y. Duan, W. Wang, J. He, T. Lu, J. Dai, and Y. Qiao. Vision transformer adapter for dense predictions. arXiv preprint arXiv:2205.08534, 2022.

[15] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259, 2014.

[16] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261, 2025.

[17] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3213–3223, 2016.

[18] G. Csurka, R. Volpi, and B. Chidlovskii. Semantic image segmentation: Two decades of research. arxiv, 2023.

[19] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville. Modulating early visual processing by language. Advances in neural information processing systems, 30, 2017.

[20] Y. Deldjoo. Understanding biases in chatgpt-based recommender systems: Provider fairness, temporal stability, and recency. ACM Transactions on Recommender Systems, 4(2):1–35, 2025.

[21] N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell,

Y. Bai, A. Chen, T. Conerly, et al. A mathematical framework for transformer circuits. Transformer Circuits Thread, 1(1):12, 2021.

[22] O. Elharrouss, Y. Mahmood, Y. Bechqito, M. A. Serhani, E. Badidi, J. Riffi, and H. Tairi. Loss functions in deep learning: A comprehensive review. arXiv preprint arXiv:2504.04242, 2025.

[23] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2):303–338, 2010.

[24] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. International journal of computer vision, 111(1):98–136, 2015.

[25] Z. Guo, T.-J. J. Wang, and J. Laaksonen. Clip4idc: Clip for image difference captioning. arXiv preprint arXiv:2206.00629, 2022.

[26] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. arXiv preprint arXiv:2110.04366, 2021.

[27] R. He, L. Liu, H. Ye, Q. Tan, B. Ding, L. Cheng, J. Low, L. Bing, and L. Si. On the effectiveness of adapter-based tuning for pretrained language model adaptation. In Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers), pages 2208–2222, 2021.

[28] L. Heinemann, A. Jaus, Z. Marinov, M. Kim, M. F. Spadea, J. Kleesiek, and R. Stiefelhagen. Limis: Towards language-based interactive medical image segmentation. In 2025 IEEE 22nd International Symposium on Biomedical Imaging (ISBI), pages 1–5. IEEE, 2025.

[29] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[30] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[31] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF international conference on computer vision, pages 1314–1324, 2019.

[32] R. Hu, Y. Cheng, L. Meng, J. Xia, Y. Zong, X. Shi, and W. Lin. Training an llm-as-a-judge model: Pipeline, insights, and practical lessons. In Companion Proceedings of the ACM on Web Conference 2025, pages 228–237, 2025.

[33] Y. Huang, X. Li, Z. Du, and H. Shen. Spatiotemporal enhancement and interlevel fusion network for remote sensing images change detection. IEEE Transactions on Geoscience and Remote Sensing, 62:1–14, 2024.

[34] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. pmlr, 2015.

[35] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. Perplexity—a measure of the difficulty of speech recognition tasks. The Journal of the Acoustical Society of America, 62(S1):S63–S63, 1977.

[36] T. Jiao, C. Guo, X. Feng, Y. Chen, and J. Song. A comprehensive survey on deep learning multi-modal fusion: Methods, technologies and applications. Computers, Materials & Continua, 80(1), 2024.

[37] D. Jurafsky and J. H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models. 3rd edition, 2025. URL `https://web.stanford.edu/~jurafsky/slp3/`. Online manuscript released August 24, 2025.

[38] M. Kazemi, N. Dikkala, A. Anand, P. Devic, I. Dasgupta, F. Liu, B. Fatemi, P. Awasthi, S. Gollapudi, D. Guo, et al. Remi: A dataset for reasoning with multiple images. Advances in Neural Information Processing Systems, 37:60088–60109, 2024.

[39] A. Kirillov, R. Girshick, K. He, and P. Dollár. Panoptic feature pyramid networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 6399–6408, 2019.

[40] D. M. Kline and V. L. Berardi. Revisiting squared-error and cross-entropy functions for training neural network classifiers. Neural Computing & Applications, 14(4): 310–318, 2005.

[41] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. Advances in neural information processing systems, 35: 22199–22213, 2022.

[42] A. Kumar, A. Raghunathan, R. Jones, T. Ma, and P. Liang. Fine-tuning can

distort pretrained features and underperform out-of-distribution. arXiv preprint arXiv:2202.10054, 2022.

[43] L. H. Li, P. Zhang, H. Zhang, J. Yang, C. Li, Y. Zhong, L. Wang, L. Yuan, L. Zhang, J.-N. Hwang, et al. Grounded language-image pre-training. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10965–10975, 2022.

[44] Y. Li, X. Zhang, X. Cheng, P. Chen, and L. Jiao. Inter-temporal interaction and symmetric difference learning for remote sensing image change captioning. IEEE Transactions on Geoscience and Remote Sensing, 2024.

[45] C.-S. Lin, C.-Y. Wang, Y.-C. F. Wang, and M.-H. Chen. Semantic prompt learning for weakly-supervised semantic segmentation. In 2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pages 8764–8774. IEEE, 2025.

[46] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In Text summarization branches out, pages 74–81, 2004.

[47] M. Lin, Q. Chen, and S. Yan. Network in network. arXiv preprint arXiv:1312.4400, 2013.

[48] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[49] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision, pages 2980–2988, 2017.

[50] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. Advances in neural information processing systems, 36:34892–34916, 2023.

[51] H. Liu, X. Zhang, H. Xu, Y. Shi, C. Jiang, M. Yan, J. Zhang, F. Huang, C. Yuan, B. Li, et al. Mibench: Evaluating multimodal large language models over multiple images. arXiv preprint arXiv:2407.15272, 2024.

[52] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[53] L. Long, R. Wang, R. Xiao, J. Zhao, X. Ding, G. Chen, and H. Wang. On llms-

driven synthetic data generation, curation, and evaluation: A survey. arXiv preprint arXiv:2406.15126, 2024.

[54] F. Meng, J. Wang, C. Li, Q. Lu, H. Tian, J. Liao, X. Zhu, J. Dai, Y. Qiao, P. Luo, et al. Mmiu: Multimodal multi-image understanding for evaluating large vision-language models. arXiv preprint arXiv:2408.02718, 2024.

[55] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[56] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality, 2013.

[57] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In 2016 fourth international conference on 3D vision (3DV), pages 565–571. Ieee, 2016.

[58] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. IEEE transactions on pattern analysis and machine intelligence, 44(7):3523–3542, 2021.

[59] MistralAI. Mistral small 3.1, 2025. URL `https://mistral.ai/news/mistral-small-3-1/`.

[60] Ollama Team. Ollama, 2025. URL `https://ollama.com/`.

[61] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748, 2018.

[62] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics, pages 311–318, 2002.

[63] D. H. Park, T. Darrell, and A. Rohrbach. Robust change captioning. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 4624–4633, 2019.

[64] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

[65] S. Petryk, L. Dunlap, K. Nasseri, J. Gonzalez, T. Darrell, and A. Rohrbach. On guiding visual attention with language specification. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 18092–18102, 2022.

[66] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[67] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In International conference on machine learning, pages 8748–8763. PmLR, 2021.

[68] V. Rajput. Robustness of different loss functions and their impact on networks learning capability. arXiv preprint arXiv:2110.08322, 2021.

[69] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer, 2015.

[70] A. S. Ross, M. C. Hughes, and F. Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. arXiv preprint arXiv:1703.03717, 2017.

[71] C. Rupprecht, I. Laina, N. Navab, G. D. Hager, and F. Tombari. Guide me: Interacting with deep networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 8551–8561, 2018.

[72] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927, 2024.

[73] S. Salman and X. Liu. Overfitting mechanism and avoidance in deep neural networks. arXiv preprint arXiv:1901.06566, 2019.

[74] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4510–4520, 2018.

[75] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision, pages 618–626, 2017.

[76] S. M. Siddiqui, M. A. Sheikh, M. Aleem, and K. R. Singh. Comparative analysis of efficient adapter-based fine-tuning of state-of-the-art transformer models. arXiv preprint arXiv:2501.08271, 2025.

[77] S. Sun, R. Li, P. Torr, X. Gu, and S. Li. Clip as rnn: Segment countless visual concepts without training endeavor. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13171–13182, 2024.

[78] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière, et al. Gemma 3 technical report. arXiv preprint arXiv:2503.19786, 2025.

[79] M. Thoma. A survey of semantic segmentation. arXiv preprint arXiv:1602.06541, 2016.

[80] T. Thrush, R. Jiang, M. Bartolo, A. Singh, A. Williams, D. Kiela, and C. Ross. Winoground: Probing vision and language models for visio-linguistic compositionality. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5238–5248, 2022.

[81] V. Tshitoyan, J. Dagdelen, L. Weston, A. Dunn, Z. Rong, O. Kononova, K. Persson, G. Ceder, and A. Jain. Unsupervised word embeddings capture latent knowledge from materials science literature. Nature, 571:95–98, 07 2019.

[82] M. Uzair Khattak, M. Ferjad Naeem, J. Hassan, M. Naseer, F. Tombari, F. Shahbaz Khan, and S. Khan. How good is my video lmm? complex video reasoning and robustness evaluation suite for video-lmms. arXiv e-prints, pages arXiv–2405, 2024.

[83] M. Varma, J.-B. Delbrouck, Z. Chen, A. Chaudhari, and C. Langlotz. Ravl: Discovering and mitigating spurious correlations in fine-tuned vision-language models. Advances in Neural Information Processing Systems, 37:82235–82264, 2024.

[84] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[85] F. Wang, X. Fu, J. Y. Huang, Z. Li, Q. Liu, X. Liu, M. D. Ma, N. Xu, W. Zhou, K. Zhang, et al. Muirbench: A comprehensive benchmark for robust multi-image understanding. arXiv preprint arXiv:2406.09411, 2024.

[86] Z. Wang, M. Berman, A. Rannen-Triki, P. Torr, D. Tuia, T. Tuytelaars, L. V. Gool, J. Yu, and M. B. Blaschko. Revisiting evaluation metrics for semantic segmentation: Optimization and evaluation of fine-grained intersection over union. In Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2023. URL `https://openreview.net/forum?id=jGyMUum1Lq`.

[87] Z. Wang, H. Zhang, X. Li, K.-H. Huang, C. Han, S. Ji, S. M. Kakade, H. Peng, and H. Ji. Eliminating position bias of language models: A mechanistic approach. arXiv preprint arXiv:2407.01100, 2024.

[88] G. Wardle and T. Sušnjak. Image first or text first? optimising the sequencing of modalities in large language model prompting and reasoning tasks. Big Data and Cognitive Computing, 9(6):149, 2025.

[89] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.

[90] R. Xiao, S. Kim, M.-I. Georgescu, Z. Akata, and S. Alaniz. Flair: Vlm with fine-grained language-informed image representations. In Proceedings of the Computer Vision and Pattern Recognition Conference, pages 24884–24894, 2025.

[91] C. Xie, B. Wang, F. Kong, J. Li, D. Liang, G. Zhang, D. Leng, and Y. Yin. Fg-clip: Fine-grained visual and textual alignment. arXiv preprint arXiv:2505.05071, 2025.

[92] J. Xu, S. De Mello, S. Liu, W. Byeon, T. Breuel, J. Kautz, and X. Wang. Groupvit: Semantic segmentation emerges from text supervision. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 18134–18144, 2022.

[93] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. In The Twelfth International Conference on Learning Representations, 2023.

[94] L. Yao, W. Wang, and Q. Jin. Image difference captioning with pre-training and contrastive learning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 3108–3116, 2022.

[95] M. Yi, Q. Cui, H. Wu, C. Yang, O. Yoshie, and H. Lu. A simple framework for text-supervised semantic segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7071–7080, 2023.

[96] M. Yuksekgonul, F. Bianchi, P. Kalluri, D. Jurafsky, and J. Zou. When and why vision-language models behave like bags-of-words, and what to do about it? arXiv preprint arXiv:2210.01936, 2022.

[97] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language image pre-training. In Proceedings of the IEEE/CVF international conference on computer vision, pages 11975–11986, 2023.

[98] D. Zhang, J. Yang, H. Lyu, Z. Jin, Y. Yao, M. Chen, and J. Luo. Cocot: Contrastive chain-of-thought prompting for large multimodal models with multiple image inputs. arXiv preprint arXiv:2401.02582, 2024.

[99] B. Zhao, Y. Zong, L. Zhang, and T. Hospedales. Benchmarking multi-image understanding in vision and language models: Perception, knowledge, reasoning, and multi-hop reasoning. arXiv preprint arXiv:2406.12742, 2024.

[100] Y. Zhong, J. Yang, P. Zhang, C. Li, N. Codella, L. H. Li, L. Zhou, X. Dai, L. Yuan, Y. Li, et al. Regionclip: Region-based language-image pretraining. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 16793–16803, 2022.

[101] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ade20k dataset. International Journal of Computer Vision, 127(3):302–321, 2019.

[102] K. Zhou, J. Yang, C. C. Loy, and Z. Liu. Learning to prompt for vision-language models. International Journal of Computer Vision, 130(9):2337–2348, 2022.

# A | Visualizations

## A.1. Formatted Evaluation Prompt

The following frame visualizes an example of a fully formatted evaluation prompt for the LLM-as-a-Judge automated evaluation, built from the class-split evaluation prompt template using the settings selected in Subsection 4.1.1. The example query is from VOC2012 with '2007_000323'. The positive class is the PERSON class.

**Formatted LLM-as-a-Judge Evaluation Prompt Example**

You are an intelligent chatbot designed for evaluating the correctness of AI assistant predictions for question-answer pairs. Your task is to compare the predicted answer with the ground-truth answer and determine if the predicted answer is correct or not. Here's how you can accomplish the task:

---

### Instructions

- Focus on the correctness, completeness and accuracy of the predicted answer with the ground-truth.
- Consider predictions with less or more specific details (as long as they show some consistency with the ground truth) as correct evaluation.
- Be strict with your evaluations.
- Expect precision from the predicted answer, it is not enough for it to roughly capture the essence of the ground truth, the prediction has to overlap sufficiently to the ground truth to be considered correct.
- Be critical of significant inconsistencies of error types and spatial locations.

Please evaluate the following answer:

Ground truth correct Answer:

"The boundaries of the prediction mask for the ground truth PERSON region tend to be coarser and over-extended."

Predicted Answer:

"The prediction mask of the ground truth PERSON region is quite rough and irregular. The boundaries are not well-defined, and there are several small holes and over-segmentations within the region."

Provide your evaluation as a correct/incorrect prediction along with the score where the score is an integer value between 0 (fully wrong) and 5 (fully correct). The middle score provides the percentage of correctness. Please generate the response in the form of a Python dictionary string with keys 'pred', 'score' and 'reason', where value of 'pred' is a string of 'correct' or 'incorrect', value of 'score' is in INTEGER, not STRING and value of 'reason' should provide the reason behind the decision.

Only provide the Python dictionary string. Escape properly quotes within the 'reason' string. For example, your response should look like this: {'reason': reason, 'score': 4, 'pred': 'correct'}.

**Important**:

The predicted answer only covers the class PERSON: restrict your focus exclusively on the PERSON class, you must ignore all the statements in the ground truth that consider different classes. If the ground-truth answer does not mention the class PERSON, assume it is saying that "Both masks have no PERSON in them, so there is no deviation".

## A.2. Formatted Inference Prompt

The following frame visualizes an example of a fully formatted prompt, built from the inference prompt template using the settings selected in Subsection 4.1.2. The example query is from VOC2012 with '2007_000061'. The positive class is the BOAT class.

**Formatted Inference Prompt Example**

I am in a binary semantic segmentation context and I want to compare a **prediction** mask with a **ground truth** mask, both segmented over the same **scene**.

In both masks, a color-class mapping is applied: the white color is mapped to the BOAT class, while the black color refers to unlabelled classes.

I will give you two images: the first image is the ground truth mask, the second image is the prediction mask. Both images are overlaid with the scene to support your analysis.

# Instructions

Your task is to find where and how the prediction deviates from the ground truth. Assume the ground truth to be correct. If there are no significant deviations, simply say it.

To help you, I will give you a set of example images, each associated with an ideal answer, which might mention classes whose names are irrelevant to your problem.

EXAMPLE 1.

Input:

Ground Truth.

Prediction.

Output:

"The ground truth AEROPLANE region has been segmented quite well by the prediction, but the boundaries are imprecise and less defined."

---

Now, I ask you to generate the output based on the following input. Remember that the considered class is the BOAT class, reference it explicitly in the answer.

Input:

Ground Truth.

Prediction.

Output:

## A.3. FLAIR Attention Maps

Figure A.1 shows some examples of attention maps comparing the pre-trained and fine-tuned checkpoints of FLAIR (see Section 4.2) on a subset of SDD (see Subsection 3.2.1).

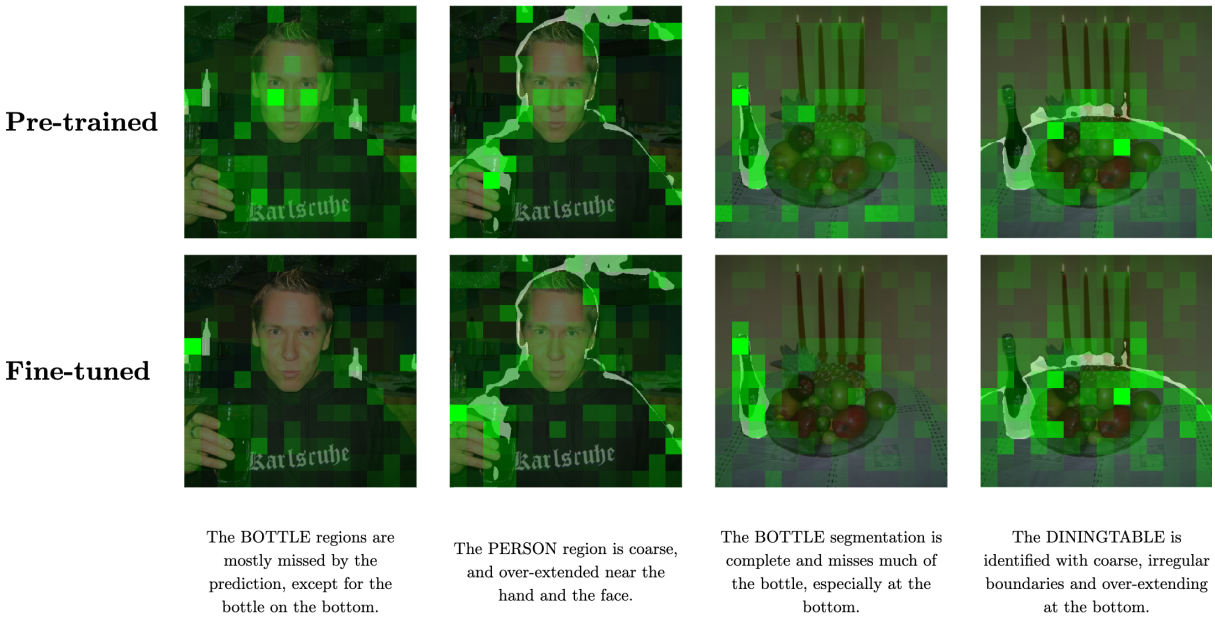



Figure A.1: A subset of attention maps obtained by projecting the texts displayed at the bottom onto the overlaid masks. Samples 1 and 3 demonstrated improved refinement, whereas samples 2 and 4 showed minimal improvement.

## A.4. Synthetic Negatives Word Pools

The following snippet illustrates, as a Python dictionary, the word pools adopted in the first substitution stage of the synthetic negatives generation algorithm (see Subsection 3.3.3) for the experiments presented in Section 4.3. Notably, the `"classes"` pool is dynamically restricted at runtime for each sample, so that only the classes present in either the ground-truth or prediction masks are considered, to encourage the generation of hard negatives.

```
word_pools = {
    "classes":                          ["BACKGROUND", "AEROPLANE", "BICYCLE", "BIRD", "BOAT", "BOTTLE",
    "BUS", "CAR", "CAT", "CHAIR", "COW", "DININGTABLE", "DOG", "HORSE", "MOTORBIKE", "PERSON", "
    POTTEDPLANT", "SHEEP", "SOFA", "TRAIN", "TVMONITOR", "UNLABELLED"],
    "positional":                       ["top", "bottom", "left", "right", "middle", "center"],
    "positional_2":                     ["downwards", "upwards"],
    "colors":                           ["red", "blue", "green", "yellow"],
    "black_white":                      ["black", "white"],
    "size":                             ["big", "large", "small", "tiny", "huge"],
    "significancy":                     ["substantial", "significant", "insignificant", "negligible"],
    "size_relative":                    ["bigger", "larger", "smaller", "tinier"],
    "quality":                          ["good", "bad", "great", "poor", "excellent", "wrong", "flawed",
    "flawless"],
    "quality_detailed":                 ["precise", "imprecise", "regular", "irregular", "complete", "
    incomplete", "noisy", "chaotic", "rough", "defined", "undefined", "clean", "coarse", "detailed", "
    harsh", "sharp"],
```

```
    "quality_adverb":                   ["well", "badly", "greatly", "poorly", "excellently", "flawlessly
    "],
    "quality_relative":                 ["better", "worse"],
    "quality_detailed_adverb":          ["precisely", "imprecisely", "regularly", "irregularly", "
    completely", "incompletely", "noisily", "chaotically", "roughly", "definedly", "undefinedly", "
    cleanly", "coarsly", "Detailedly", "harshly", "sharply"],
    "quality_detailed_adverb_relative": ["rougher", "cleaner", "coarser", "sharper"],
    "quantity_relative":                ["more", "less"],
    "position_relative":                ["above", "below", "beside", "near", "nearby", "around", "along"
    ],
    "area":                             ["boundary", "border", "interior", "center"],
    "areas":                            ["boundaries", "edges", "borders", "interiors", "inners"],
    "lower":                            ["lower", "elevated"],
    "level":                            ["over", "under"],
    "segmented":                        ["oversegmented", "undersegmented", "missegmented", "overextended
    ", "underextended", "hallucinated", "extended", "missed", "hallucinated", "overspilled"],
    "segmentation":                     ["oversegmentation", "undersegmentation", "overspill", "
    hallucination", "overextension", "underextension"]
}
```

# B | Influence of Regularization on Cached Masks Variability

As discussed in Subsection 3.3.4, the subsampling strategy assumes that masks exhibiting the greatest temporal variability are the primary drivers of segmentation model training. However, regularization and augmentation techniques may introduce additional mask variability that is not intrinsic to the optimization process, potentially reducing the effectiveness of the subsampling strategy.

To investigate this, an experiment was conducted to evaluate the impact of common regularization and augmentation techniques on cached mask similarity. For each sample in every batch, the mIoU between the predicted mask and its corresponding cached mask was computed. The mean batch mIoU and its standard deviation were then monitored over time to quantify mask variability. In parallel, the validation mIoU was tracked to assess how each regularization strategy affected segmentation performance. The objective was to identify the technique that introduced the least mask variability while preserving segmentation precision. No language-guided mechanism was applied in this specific experiment.

## Experimental Setup

LR-ASPP-MobileNetV3-Large [31] was trained on the VOC2012 dataset [24] using the same configuration described in Section 4.3, except for the base learning rate, which was set to $10^{-3.5}$. The model underwent an initial training phase of two epochs. No cache prefilling was performed. Therefore, the first epoch does not report cache mIoU values, as no cached masks were available.

All runs used center cropping, except for the configuration with random cropping. Random horizontal flipping was applied with probability $p = 0.5$, and dropout was set to 0.2, as in the original segmentation model protocol [31]. The model was trained in full precision. Cached mask mIoU trends were computed using simple exponential smoothing with a factor of 0.9 applied to the batch-level metrics. To improve plot readability and

prevent overlapping bands, dispersion was visualized as the standard deviation scaled by a factor of 0.3. Validation mIoU was measured at the start of training and after each epoch, then aligned with training steps for plotting.

## Experimental Results

The results, illustrated in Figure B.1, indicate that random horizontal flipping has the smallest impact on mask variability, with the average cache mIoU converging to ∼0.6. Dropout follows closely, converging to a slightly lower value, while random cropping produces the lowest converged average mIoU (∼0.4). As expected, the run without augmentation converges toward near-complete mask similarity. In contrast, combining all tested regularization techniques reduces similarity to ∼ 0.2. Overall, all configurations exhibit similar convergence patterns.

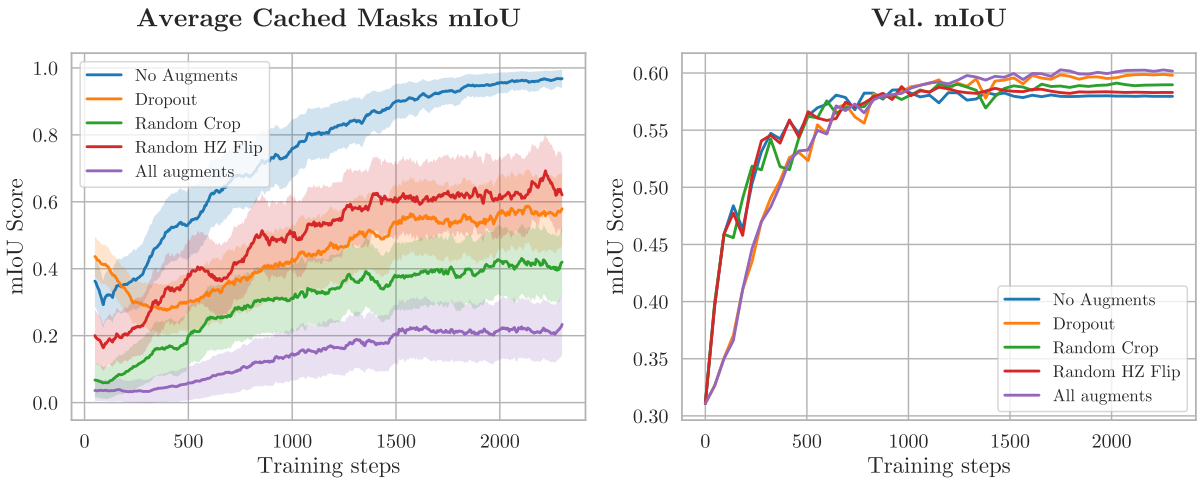



Figure B.1: The plots display the average mIoU trends between the predicted and cached masks, with training results on the left and validation results on the right.

Dropout has the greatest positive impact on validation performance, while the other configurations tend to cause a more pronounced drop in mIoU. Dropout, therefore, provides the best trade-off between preserving mask similarity and maintaining validation performance, and was consequently selected for the pipeline experiments described in Section 4.3.

# C | Contrastive Losses Implementation

## C.1. Grouped SigLIP Loss

What follows is the Python implementation of the Grouped SigLIP loss, built as a PyTorch `nn.Module`. It computes the loss over batches of image and text features, with groupings specified by a vector of image indices. Fully compatible with PyTorch tensors, it leverages tensor operations for fast, parallelized computation.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from typing import Optional

class GroupedSigLipLoss(nn.Module):
    """
    Sigmoid Loss for Language Image Pre-Training (SigLIP) with grouped data and in-batch negatives.

    This loss is designed for datasets where image-text pairs can be grouped.
    Each image is paired with at most one positive text, and each text is paired
    with exactly one image. The key features are:

    1.  **One-to-One Pairing:** Each positive text corresponds to a single image instance.
    2.  **Grouping:** Positive image-text pairs are assigned to groups.
    3.  **Group Normalization:** The total loss (positive + negative) for each
        group is normalized by the number of images in that group. This ensures
        that large groups do not dominate the final batch loss.
    4.  **In-Batch Negatives:** For each image, all texts in the batch that are not
        its designated positive are treated as negatives.
    5.  **Isolated Images:** Images without a positive text are handled correctly,
        contributing only negative loss (against all texts in the batch) and
        forming their own "group of one" for the purpose of the final loss calculation.

    Use cases:
    - Training with data organized into semantic groups (e.g., categories, scenes)
    - Ensuring balanced loss contribution across groups of different sizes
    - Handling datasets where some images lack positive text descriptions

    Attributes:
        ignore_no_positive_samples (bool): Whether to zero out loss for images without positives.
    """
```

```python
    def __init__(
            self,
            ignore_no_positive_samples: bool = True
    ) -> None:
        """
        Initializes the GroupedSigLipLoss module.

        Args:
            ignore_no_positive_samples (bool): If True, images with no positive text pairings
                                               will have their total loss contribution zeroed out,
                                               effectively removing them from the batch's loss
                                               calculation. Defaults to True.
        """
        super().__init__()
        self.ignore_no_positive_samples = ignore_no_positive_samples

    def get_logits(
            self,
            image_features: torch.Tensor,
            text_features: torch.Tensor,
            logit_scale: torch.Tensor,
            logit_bias: Optional[torch.Tensor] = None,
    ) -> torch.Tensor:
        """
        Calculates the pairwise similarity logits between images and texts.

        Computes the scaled dot product similarity matrix between all image-text pairs,
        optionally adding a learnable bias term.

        Args:
            image_features (torch.Tensor): Image embeddings of shape (N, D).
            text_features (torch.Tensor): Text embeddings of shape (M, D).
            logit_scale (torch.Tensor): Learnable temperature scaling parameter.
            logit_bias (Optional[torch.Tensor]): Learnable bias to add to logits.

        Returns:
            torch.Tensor: Similarity logits matrix of shape (N, M).
        """
        logits = logit_scale * image_features @ text_features.T
        if logit_bias is not None:
            logits += logit_bias
        return logits

    def forward(
            self,
            image_features: torch.Tensor,
            positive_text_features: torch.Tensor,
            image_indices_for_pos_texts: torch.Tensor,
            group_indices_for_pos_pairs: torch.Tensor,
            logit_scale: torch.Tensor,
            logit_bias: Optional[torch.Tensor] = None,
            output_dict: bool = False,
    ) -> torch.Tensor | dict[str, torch.Tensor]:
        """
        Computes the SigLIP loss for grouped data with in-batch negatives.

```

```
        The loss computation follows these steps:
        1. Calculate pairwise positive and negative losses between all images and all texts.
        2. Separate the true positive losses and the per-image aggregated negative losses.
        3. Map all images to groups (including creating isolated groups for images without positives).
        4. Aggregate positive and negative losses by group.
        5. Normalize each group's total loss by the number of images in that group.
        6. If 'ignore_no_positive_samples' is True, zero out the loss for groups of isolated images.
        7. Return the mean loss across all groups.

        Args:
            image_features (torch.Tensor): Shape (N, D), where N is the total number of images
                                           and D is the embedding dimension.
            positive_text_features (torch.Tensor): Shape (P, D), where P is the total number of
                                                   positive texts (P <= N).
            image_indices_for_pos_texts (torch.Tensor): Shape (P,), maps each positive text
                to its corresponding image index.
            group_indices_for_pos_pairs (torch.Tensor): Shape (P,), maps each positive pair
                to a group index. Groups are numbered starting from 0.
            logit_scale (torch.Tensor): The learnable temperature scaling parameter.
            logit_bias (Optional[torch.Tensor]): The learnable bias parameter to add to logits.
            output_dict (bool): If True, returns a dictionary with key 'contrastive_loss'.
                                If False, returns the loss tensor directly.

        Returns:
            torch.Tensor | dict[str, torch.Tensor]: The computed loss as a single scalar tensor,
                                                    or a dictionary if output_dict is True.
                                                    Returns 0.0 with requires_grad=True for empty batches
        .
        """
        device = image_features.device
        num_images = image_features.shape[0]
        num_pos_pairs = positive_text_features.shape[0]

        if num_images == 0:
            loss = torch.tensor(0.0, device=device, requires_grad=True)
            return {"contrastive_loss": loss} if output_dict else loss

        # --- Steps 1 & 2: Calculate all pairwise losses (positive and negative) ---
        neg_loss_per_image = torch.zeros(num_images, device=device)
        true_pos_losses = torch.tensor([], device=device)

        if num_pos_pairs > 0:
            # Calculate logits between ALL images and ALL texts in the batch.
            all_logits = self.get_logits(image_features, positive_text_features, logit_scale, logit_bias)

            # Create a label matrix: 1 for true pairs, 0 for negatives.
            labels = torch.zeros_like(all_logits)
            text_indices = torch.arange(num_pos_pairs, device=device)
            labels[image_indices_for_pos_texts, text_indices] = 1

            # Use BCEWithLogitsLoss for a numerically stable way to compute
            # -log(sigmoid(logit)) for positives (where label=1)
            # -log(sigmoid(-logit)) for negatives (where label=0)
            per_entry_loss = F.binary_cross_entropy_with_logits(all_logits, labels, reduction='none')

            # Extract the loss for the true positive pairs
```

```
            true_pos_losses = per_entry_loss[image_indices_for_pos_texts, text_indices]

            # Calculate the total negative loss for each image by summing the losses
            # for all entries where the label is 0.
            neg_loss_matrix = per_entry_loss * (1 - labels)
            neg_loss_per_image = neg_loss_matrix.sum(dim=1)

        # --- Step 3: Map all images to a group ---
        # This logic is identical to GroupedPairedNegativeSigLipLoss
        num_positive_groups = 0
        if num_pos_pairs > 0:
            num_positive_groups = int(group_indices_for_pos_pairs.max().item()) + 1

        group_indices_for_images = torch.full((num_images,), -1, dtype=torch.long, device=device)
        if num_pos_pairs > 0:
            group_indices_for_images.scatter_(0, image_indices_for_pos_texts, group_indices_for_pos_pairs
    )

        isolated_mask = (group_indices_for_images == -1)
        num_isolated = isolated_mask.sum()
        isolated_group_indices = torch.arange(
            num_positive_groups, num_positive_groups + num_isolated, device=device
        )
        group_indices_for_images[isolated_mask] = isolated_group_indices

        num_total_groups = num_positive_groups + num_isolated

        if num_total_groups == 0:
             loss = torch.tensor(0.0, device=device, requires_grad=True)
             return {"contrastive_loss": loss} if output_dict else loss

        # --- Step 4: Aggregate positive and negative losses by group ---
        pos_loss_per_group = torch.zeros(num_total_groups, device=device, dtype=true_pos_losses.dtype)
        if num_pos_pairs > 0:
            pos_loss_per_group.scatter_add_(0, group_indices_for_pos_pairs, true_pos_losses)

        neg_loss_per_group = torch.zeros(num_total_groups, device=device)
        neg_loss_per_group.scatter_add_(0, group_indices_for_images, neg_loss_per_image)

        total_loss_per_group = pos_loss_per_group + neg_loss_per_group

        # --- Step 5: Normalize by group size and compute final loss ---
        group_sizes = torch.bincount(group_indices_for_images, minlength=num_total_groups)
        group_normalizer = torch.clamp(group_sizes, min=1).float()
        normalized_loss_per_group = total_loss_per_group / group_normalizer

        if self.ignore_no_positive_samples and num_isolated > 0:
            isolated_group_start_index = num_positive_groups
            normalized_loss_per_group[isolated_group_start_index:] = 0.0

        loss = normalized_loss_per_group.mean()

        return {"contrastive_loss": loss} if output_dict else loss
```

## C.2. Grouped SigLIP Loss with Synthetic Negatives

What follows is the Python implementation of the Grouped SigLIP loss supporting synthetic negatives, built as a PyTorch `nn.Module`. It computes the loss over batches of image and text features, with groupings specified by a vector of image indices. The negative texts features are provided per-image as a separate input. Fully compatible with PyTorch tensors, it leverages tensor operations for fast, parallelized computation.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from typing import Optional

class GroupedPairedNegativeSigLipLoss(nn.Module):
    """
    Sigmoid Loss for Language Image Pre-Training (SigLIP) with grouped, paired data.

    This loss is designed for datasets where image-text pairs can be grouped.
    Each image is paired with at most one positive text, and each text is paired
    with exactly one image. The key features are:

    1.  **One-to-One Pairing:** Each positive text corresponds to a single image instance.
    2.  **Grouping:** Positive image-text pairs are assigned to groups.
    3.  **Group Normalization:** The total loss (positive + negative) for each
        group is normalized by the number of images in that group. This ensures
        that large groups do not dominate the final batch loss.
    4.  **Paired Negatives:** Each image (regardless of whether it has a positive
        text) is contrasted against its own dedicated non-empty set of negative texts.
    5.  **Isolated Images:** Images without a positive text are handled correctly,
        contributing only negative loss and forming their own "group of one" for
        the purpose of the final loss calculation.

    Use cases:
    - Training with data organized into semantic groups (e.g., categories, scenes)
    - Ensuring balanced loss contribution across groups of different sizes
    - Handling datasets where some images lack positive text descriptions

    Attributes:
        ignore_no_positive_samples (bool): Whether to zero out loss for images without positives.
    """
    def __init__(
            self,
            ignore_no_positive_samples: bool = True
    ) -> None:
        """
        Initializes the GroupedPairedNegativeSigLipLoss module.

        Args:
            ignore_no_positive_samples (bool): If True, images with no positive text pairings
                                               will have their total loss contribution zeroed out,
                                               effectively removing them from the batch's loss
                                               calculation. Defaults to True.
        """
        super().__init__()
```

```
        self.ignore_no_positive_samples = ignore_no_positive_samples

    def get_logits(
            self,
            image_features: torch.Tensor,
            text_features: torch.Tensor,
            logit_scale: torch.Tensor,
            logit_bias: Optional[torch.Tensor] = None,
    ) -> torch.Tensor:
        """
        Calculates the pairwise similarity logits between images and texts.

        Computes the scaled dot product similarity matrix between all image-text pairs,
        optionally adding a learnable bias term.

        Args:
            image_features (torch.Tensor): Image embeddings of shape (N, D).
            text_features (torch.Tensor): Text embeddings of shape (M, D).
            logit_scale (torch.Tensor): Learnable temperature scaling parameter.
            logit_bias (Optional[torch.Tensor]): Learnable bias to add to logits.

        Returns:
            torch.Tensor: Similarity logits matrix of shape (N, M).
        """
        logits = logit_scale * image_features @ text_features.T
        if logit_bias is not None:
            logits += logit_bias
        return logits

    def forward(
            self,
            image_features: torch.Tensor,
            positive_text_features: torch.Tensor,
            image_indices_for_pos_texts: torch.Tensor,
            group_indices_for_pos_pairs: torch.Tensor,
            negative_text_features: torch.Tensor,
            logit_scale: torch.Tensor,
            logit_bias: Optional[torch.Tensor] = None,
            output_dict: bool = False,
    ) -> torch.Tensor | dict[str, torch.Tensor]:
        """
        Computes the SigLIP loss for grouped, paired data.

        The loss computation follows these steps:
        1. Calculate negative loss for all images using their dedicated negatives
        2. Calculate positive loss for all positive pairs
        3. Map all images to groups (including creating isolated groups for images without positives)
        4. Aggregate positive and negative losses by group
        5. Normalize each group's total loss by the number of images in that group
        6. If 'ignore_no_positive_samples' is True, zero out the loss for groups of isolated images.
        7. Return mean loss across all groups

        Args:
            image_features (torch.Tensor): Shape (N, D), where N is the total number of images
                                           and D is the embedding dimension.
            positive_text_features (torch.Tensor): Shape (P, D), where P is the total number of
```

```
                                            positive texts (P <= N). Each positive text is
                                            paired with exactly one image.
        image_indices_for_pos_texts (torch.Tensor): Shape (P,), maps each positive text
            to its corresponding image index. E.g., `image_indices_for_pos_texts[j] = i`
            means `positive_text_features[j]` is the positive for `image_features[i]`.
        group_indices_for_pos_pairs (torch.Tensor): Shape (P,), maps each positive pair
            to a group index. E.g., `group_indices_for_pos_pairs[j] = g` means the
            j-th pair belongs to group g. Groups are numbered starting from 0.
        negative_text_features (torch.Tensor): Shape (N, M, D), where M is the number of
            dedicated negatives for each of the N images. Can have M=0 for no negatives.
        logit_scale (torch.Tensor): The learnable temperature scaling parameter.
        logit_bias (Optional[torch.Tensor]): The learnable bias parameter to add to logits.
        output_dict (bool): If True, returns a dictionary with key 'contrastive_loss'.
                            If False, returns the loss tensor directly.

    Returns:
        torch.Tensor | dict[str, torch.Tensor]: The computed loss as a single scalar tensor,
                                                or a dictionary if output_dict is True.
                                                Returns 0.0 with requires_grad=True for empty batches
    .
    """
    device = image_features.device
    num_images = image_features.shape[0]
    num_pos_pairs = positive_text_features.shape[0]

    if num_images == 0:
        loss = torch.tensor(0.0, device=device, requires_grad=True)
        return {"contrastive_loss": loss} if output_dict else loss

    # --- Step 1: Calculate Negative Loss for ALL images ---
    # This is computed per-image, independent of grouping initially.
    neg_loss_per_image = torch.zeros(num_images, device=device)
    if negative_text_features.numel() > 0 and negative_text_features.shape[1] > 0:
        neg_logits = torch.einsum('nd,nmd->nm', image_features, negative_text_features)
        neg_logits = logit_scale * neg_logits
        if logit_bias is not None:
            neg_logits += logit_bias

        neg_pairwise_loss = -F.logsigmoid(-neg_logits)

        neg_loss_per_image = neg_pairwise_loss.sum(dim=1)

    # --- Step 2: Calculate Positive Loss for all PAIRS ---
    true_pos_losses = torch.tensor([], device=device)
    if num_pos_pairs > 0:
        # Logits between all images and all positive texts
        pos_logits = self.get_logits(image_features, positive_text_features, logit_scale, logit_bias)

        # Loss for positive pairs is -log(sigmoid(logit))
        pos_pairwise_loss = -F.logsigmoid(pos_logits)

        # Extract the loss for the true positive pairs
        text_indices = torch.arange(num_pos_pairs, device=device)
        true_pos_losses = pos_pairwise_loss[image_indices_for_pos_texts, text_indices]

    # --- Step 3: Map all images to a group and aggregate losses ---
```

```
158
159         # Determine the total number of groups. This includes groups defined by
160         # positive pairs and new "isolated" groups for images without positives.
161         num_positive_groups = 0
162         if num_pos_pairs > 0:
163             num_positive_groups = int(group_indices_for_pos_pairs.max().item()) + 1
164
165         # Create a mapping from each image index to its final group index.
166         group_indices_for_images = torch.full((num_images,), -1, dtype=torch.long, device=device)
167         if num_pos_pairs > 0:
168             # Images with positives get their assigned group index
169             group_indices_for_images.scatter_(0, image_indices_for_pos_texts, group_indices_for_pos_pairs
    )
170
171         # Images without a positive pair (still marked as -1) are assigned to
172         # new, unique "isolated" groups.
173         isolated_mask = (group_indices_for_images == -1)
174         num_isolated = isolated_mask.sum()
175         isolated_group_indices = torch.arange(
176             num_positive_groups, num_positive_groups + num_isolated, device=device
177         )
178         group_indices_for_images[isolated_mask] = isolated_group_indices
179
180         num_total_groups = num_positive_groups + num_isolated
181
182         if num_total_groups == 0: # Happens if num_images > 0 but no groups assigned (shouldn't occur
    with this logic)
183             loss = torch.tensor(0.0, device=device, requires_grad=True)
184             return {"contrastive_loss": loss} if output_dict else loss
185
186         # Aggregate positive and negative losses by their final group index
187         pos_loss_per_group: torch.Tensor = torch.zeros(num_total_groups, device=device, dtype=
    true_pos_losses.dtype)
188         if num_pos_pairs > 0:
189             pos_loss_per_group.scatter_add_(0, group_indices_for_pos_pairs, true_pos_losses)
190
191         neg_loss_per_group: torch.Tensor = torch.zeros(num_total_groups, device=device)
192         neg_loss_per_group.scatter_add_(0, group_indices_for_images, neg_loss_per_image)
193
194         total_loss_per_group = pos_loss_per_group + neg_loss_per_group
195
196         # --- Step 4: Normalize by group size and compute final loss ---
197
198         # Group size is the number of IMAGES in each group
199         group_sizes = torch.bincount(group_indices_for_images, minlength=num_total_groups)
200
201         # Normalize the total loss of each group by its size
202         group_normalizer = torch.clamp(group_sizes, min=1).float()
203         normalized_loss_per_group = total_loss_per_group / group_normalizer
204
205         # If specified, zero out the loss contribution from isolated images
206         # (those without any positive text pairs). These form their own groups.
207         if self.ignore_no_positive_samples and num_isolated > 0:
208             # The isolated groups are the ones appended at the end
209             isolated_group_start_index = num_positive_groups
210             normalized_loss_per_group[isolated_group_start_index:] = 0.0
```

```
211
212         # The final loss is the mean of the normalized per-group losses
213         loss = normalized_loss_per_group.mean()
214
215         return {"contrastive_loss": loss} if output_dict else loss
```

# List of Figures

# List of Tables

# Acronyms

**BN** Batch Normalisation. 37, 38, 127

**CE** Cross-Entropy. 11–13, 24, 40, 52–54, 78–81, 85, 90, 92–95, 131

**COCO2017** Microsoft Common Objects in Context 2017. 15, 18, 49, 50, 125

**CoT** Chain-of-Thought. 32, 33, 35, 36, 126

**CV** Computer Vision. 1, 7, 15, 23

**DL** Deep Learning. 1, 4, 8, 10

**GAP** Global Average Pooling. 62, 63

**GRU** Gated Recurrent Unit. 38, 40

**HITL** Human-In-The-Loop. 1–3

**ICL** In-Context Learning. 31

**IDC** Image Difference Captioning. 30, 31

**LLM** Large Language Model. 19–21, 26, 27, 31–37, 48, 49, 65, 69, 125, 126

**LM** Language Model. 16–19

**LSTM** Long Short-Term Memory. 37

**MHA** Multi-Head Attention. 22, 23, 25, 126

**mIoU** mean Intersection over Union. 14, 78–81, 83, 85, 95, 113, 114, 129, 131

**MIU** Multi-Image Understanding. 27, 29, 30, 126

**MLLM** Multi-modal Large Language Model. 26–31, 33–36, 43–47, 49, 52–54, 60, 65, 66, 68, 69, 71, 73, 74, 78, 126, 127, 131

**MLP** Multi-Layer Perceptron. 21, 37

**NLP** Natural Language Processing. 2, 16, 23

**NN** Neural Network. 1, 8, 9

**PA** Pixel-wise Accuracy. 13, 14

**SDD** Synthetic Diagnostic Dataset. 49–52, 75–77, 88, 110, 127, 131

**SDPA** Scaled Dot-Product Attention. 21, 23, 126

**SS** Semantic Segmentation. 1–4, 7–15, 93, 131

**VLE** Vision-Language Encoder. 24, 27, 49, 52, 54, 64, 79

**VLM** Vision-Language Model. 23, 31

**VOC2012** Pascal Visual Object Classes 2012. 15–18, 48–50, 66, 78, 107, 108, 113, 125

**VQA** Visual Question Answering. 26, 37

# Acknowledgements

In concluding this work, I would like to express my sincere gratitude to my advisor, Professor Matteo Matteucci, and my co-advisors, Nico, Sara, and Chiara, for their guidance throughout this project. Their ideas, advice, and constant support helped me face the challenges of this thesis with greater confidence. I am especially grateful for their patience and encouragement during the most demanding moments.

Lastly, I would like to thank my family and my close friends, in Mantua and in Milan, who were by my side during these long months of study. Their support and presence meant more than they know and made this achievement truly meaningful.