UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITY OF MODENA AND REGGIO EMILIA

"Enzo Ferrari" Department of Engineering

Master's Degree in Artificial Intelligence Engineering (LM-32)

Less Vision Is All You Need

Reducing Visual Context in MLLMs

Supervisor:
Prof. Lorenzo Baraldi
Prof. Marcella Cornia
Co-supervisor:
Davide Caffagni

Candidate:
Nicola Ricciardi
Student ID 202000

ACADEMIC YEAR 2025/2026

Abstract

Less Vision Is All You Need: Reducing Visual Context in MLLMs

The recent proliferation of Multimodal Large Language Models (MLLMs) - particularly vision-language frameworks such as the LLaVA (Large Language-and-Vision Assistant) family - has heralded a new era of artificial intelligence characterized by sophisticated reasoning and instruction-following capabilities. However, this profound expressive power is inextricably linked to a significant computational bottleneck. In standard architectural formulations, MLLMs ingest the entirety of the visual tokens extracted by a pre-trained vision encoder. Because the self-attention mechanism at the core of the underlying Large Language Models (LLMs) scales quadratically with sequence length, processing an uncompressed, high-resolution grid of visual tokens becomes profoundly resource-intensive. To address this computational inefficiency, a dedicated module is proposed to systematically aggregate the vast set of visual features extracted by a vision encoder into a highly compact, yet semantically rich, set of dense vectors. By acting as an information bottleneck, this module aims to preserve the foundational semantic integrity required for complex multimodal reasoning while drastically reducing the quadratic computational overhead of LLMs. To determine the most effective compression strategy, three distinct aggregation paradigms are investigated. First, text-agnostic vision-only aggregation operates exclusively on the visual modality, compressing features through self-supervised reconstruction without relying on textual context. Second, text-influenced vision-centric aggregation incorporates textual influence directly into the training objective, conditioning the model to select task-relevant visual features. Finally, text-guided vision aggregation introduces a dynamic, context-aware mechanism where textual features actively govern the visual aggregation process. Through a rigorous evaluation of these paradigms across a comprehensive suite of downstream tasks, this research seeks to establish a scalable methodology for visual information synthesis, proving that the majority of visual tokens are not necessary, but rather a small set of dense ones are enough to achieve good results in downstream tasks, overcoming in thereby the computational bottlenecks of current MLLMs.

Keywords: Multimodal LLMs, Vision Token Compression, Computational Efficiency, Text-Guided Aggregation, Multimodal Reasoning

Contents

List of Figures

1. Introduction

Over the past decade, the landscape of Artificial Intelligence has been fundamentally redefined by the advent and rapid scaling of Large Language Models (LLMs) [1, 2]. Characterized by architectures encompassing billions of parameters and trained on massively diverse textual corpora, these models have demonstrated unprecedented capabilities across a wide array of cognitive tasks, notably in zero-shot generalization, complex multi-step reasoning, and deep natural language understanding.

Recently, this purely textual trajectory has naturally and necessarily evolved toward multimodal perception, heralding the proliferation of Multimodal Large Language Models (MLLMs) as the next frontier in generative AI. Pioneering frameworks such as the LLaVA (Large Language-and-Vision Assistant) family [3], Flamingo [4] and BLIP/BLIP-2 [5, 6] have successfully bridged the longstanding gap between continuous perceptual features and discrete textual tokens.

By explicitly treating visual inputs as a structural equivalent to a "foreign language" - which is then systematically mapped into the native linguistic embedding space of the model - these systems equip generative architectures with sophisticated visual reasoning and nuanced instruction-following capabilities. Consequently, MLLMs can seamlessly engage in rich, multi-turn conversations that are heavily grounded in real-world imagery, excelling in demanding analytical tasks ranging from detailed Visual Question Answering (VQA) to highly complex document understanding.

Nevertheless, this profound expressive power is inextricably linked to a severe computational bottleneck that fundamentally limits practical scalability. In standard architectural formulations, MLLMs rely heavily on pre-trained, high-capacity vision encoders - such as CLIP [7] or DINO [8] - to extract high-dimensional spatial feature maps directly from the raw input images. To preserve absolute visual fidelity, especially as input image resolutions scale up to accurately capture fine-grained semantic details, these underlying encoders inherently generate an immense quantity of visual tokens.

Because the self-attention mechanism, which resides at the core of the underlying Transformer-based LLMs [9], scales quadratically with respect to the input sequence length, ingesting an uncompressed, high-resolution grid of visual tokens becomes profoundly resource-intensive. As a result, a single high-resolution image can easily saturate a significant portion of the language

model's available context window. This saturation directly leads to unacceptable inference latency, massive memory footprints during processing, and a severely degraded signal-to-noise ratio that often overwhelms and obfuscates the model's core reasoning capabilities.

Such severe computational constraints are particularly detrimental when considering deployment in dynamic domains requiring real-time responsiveness and strict power efficiency. In critical fields such as autonomous robotics, edge computing networks and mobile consumer applications, the available computational budget is strictly limited. In these resource-constrained scenarios, deploying standard, uncompressed MLLMs remains largely unfeasible. The heavy computational burden thus necessitates a fundamental optimization and restructuring of the entire visual processing pipeline to make these advanced models viable for everyday, mobile applications.

To mitigate this quadratic computational overhead, the current literature has largely explored token pruning strategies - techniques primarily aimed at selectively discarding redundant background patches based on various attention-based metrics. Yet, traditional pruning methodologies inherently carry a substantial risk; by completely eliminating tokens, the model risks losing subtle, fine-grained visual details that may ultimately prove critical for successfully executing complex downstream reasoning tasks.

In direct response to this critical inefficiency and the limitations of hard pruning, this work proposes a fundamental paradigm shift from token pruning to visual token aggregation. Rather than indiscriminately discarding potentially valuable structural information, this research introduces the meticulous design and implementation of dedicated aggregator modules. Acting as an information bottleneck, an aggregator systematically distills the vast, unwieldy set of raw visual features into a highly compact, yet semantically rich, set of dense representational vectors. The fundamental objective of this approach is to preserve the foundational semantic integrity required for advanced multimodal reasoning while simultaneously achieving a drastic reduction in the overall input sequence length that must be processed by the underlying LLM.

To achieve a comprehensive understanding of optimal visual compression strategies, this work systematically investigates three distinct aggregation paradigms. The first paradigm, characterized as text-agnostic vision-only aggregation, operates exclusively on the visual modality without any external textual influence. Leveraging Object-Centric Learning principles - most notably Slot Attention frameworks [10, 11] - alongside simple, robust pooling mechanisms, this approach compresses visual features entirely independently of the textual context, relying solely

on the inherent semantic and spatial structure of the source image. The second paradigm, defined as text-influenced vision-centric aggregation, seamlessly integrates the proposed aggregator directly into the end-to-end MLLM training pipeline. In this specialized scenario, the language modeling implicitly conditions the aggregator to prioritize, select, and merge the visual features that possess the highest relevance to general instruction-following tasks. Finally, the third paradigm, known as text-guided vision aggregation, introduces a highly dynamic, context-aware mechanism into the compression pipeline. By utilizing advanced cross-modal architectures like the Perceiver Resampler and implementing sophisticated turn-by-turn routing mechanisms, the user's specific textual prompt actively and explicitly governs the visual aggregation process. This ensures that the visual bottleneck dynamically adapts in real-time to extract only the precise visual information strictly necessary to formulate an accurate answer to the user's specific query.

Through a rigorous empirical evaluation leveraging robust baselines, this work deeply analyzes the architectural trade-offs inherent to these distinct aggregation modules. The experimental scope is exceptionally broad, spanning from highly compact, edge-deployable models like Qwen3 0.6B [1] and LLaMA 3.2 1B [2], extending to mid-range architectures like Qwen3 1.7B [1], and scaling up to expansive models such as Vicuna 7B [12]. Extensive, multi-faceted testing conducted systematically across the Cambrian dataset collection definitively demonstrates that implementing aggressive visual compression is not only technologically viable but highly effective in practice. By intelligently compressing hundreds of raw visual tokens down to an incredibly sparse sequence of as few as eight or sixteen dense representations, the proposed architectural modifications achieve drastic, measurable reductions in both required FLOPs and overall inference time.

Remarkably, the empirical results show that in many complex reasoning scenarios, these heavily compressed representations effectively filter out extraneous visual noise. This noise filtering actively allows the efficiently compressed models to match - and occasionally even surpass - the raw reasoning performance of their computationally heavy, entirely uncompressed baseline counterparts.

Ultimately, this research seeks to establish a scalable, efficient, and dynamic methodological foundation for advanced visual information synthesis within modern generative architectures. By conclusively proving that the vast majority of raw visual tokens generated by standard encoders are semantically redundant when properly and intelligently aggregated, this work paves a definitive way forward. It lays the crucial groundwork for a new, highly optimized generation

of efficient Multimodal Large Language Models that remain fully capable of sophisticated visual reasoning, fundamentally operating without the prohibitive computational costs that currently hinder their widespread deployment in real-world, resource-constrained environments.

2. Background

In this chapter a comprehensive overview of the related work will be provided, covering the fundamental concepts and techniques that underpin the research presented in this document.

2.1 Machine Learning and Deep Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) centered on the development of mathematical models and statistical techniques that enable systems to improve their performance on a specific task through experience, without being explicitly programmed.

Formally, a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [13].

Deep Learning (DL) is a subset of Machine Learning characterized by the wide use of *artificial neural networks* (ANNs) with multiple hidden layers - hence the term "deep" [14] - composed of a set of learnable parameters (also called *weights*) that are optimized during the *back-propagation* phase, in which weights themselves are adjusted to minimize a loss function by computing gradients using *stochastic gradient descent* mechanism.

Unlike traditional ML algorithms, which often rely on handcrafted features - i.e. manually designed by human experts - DL models excel in unstructured data processing, discovering the "optimal" features directly from the data.

2.1.1 Learning Paradigms

Models can be trained using different learning paradigms, depending on the nature of the data and the task. A widely accepted taxonomy of learning paradigms categorizes them into three main types: supervised learning, unsupervised learning, and reinforcement learning [15].

In supervised learning, the model is trained on a labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where each input $x \in \mathcal{X}$ is paired with a corresponding expected output $y \in \mathcal{Y}$, called *label*. The objective is to approximate the mapping function $f : \mathcal{X} \to \mathcal{Y}$. Typical supervised learning tasks include classification, regression and sequence-to-sequence generation.

In unsupervised learning, the model is trained on an unlabeled dataset $\mathcal{D} = \{x_i\}_{i=1}^{N}$, where

the goal is to discover underlying patterns, structures, or representations in the data without any explicit guidance. For example, clustering algorithms (e.g., K-Means [16]) group similar data points together, while dimensionality reduction techniques like Principal Component Analysis (PCA) identify the most informative features in the data [17].

Moreover, Reinforcement learning (RL) is a learning paradigm in which an agent learns to make decisions by interacting with an environment $\mathcal{E}$, receiving feedback in form of rewards or penalties based on its actions. This approach is particularly effective for tasks that involve sequential decision-making - such as game playing, robotics and autonomous systems - where the state space is too large to be covered by traditional supervised learning methods, requiring a sort of *"trial and error"* strategy.

Additionally, nowadays, a new learning paradigm is emerging to bridge the gap between supervised and unsupervised learning: *self-supervised learning* (SSL), in which "pseudo-labels" are generated from the data themselves. This approach has become the de facto standard for training large-scale models, where large amount of training data are needed and human annotation is expensive.

2.1.2 Objective Functions

Beyond the learning signal, DL models are taxonomized by the mathematical manifold of their output space and the specific nature of the inference task.

- **Classification**: the model learns to assign input data to one of several predefined categories. The output is typically a probability distribution over the classes obtained using the softmax function and the objective function is often the cross-entropy loss.
- **Regression**: the model learns to predict a continuous value based on input data. The output is a single scalar or a vector of continuous values. The objective function is often the mean squared error (MSE) or mean absolute error (MAE).
- **Alignment and Multi-modal Mapping**: the model learns a common representation for different data modalities (e.g., text and vision).
- **Sequence-to-Sequence Generation**: the model learns to generate sequences of data, such as text, audio, or time series. The output is a sequence of values and the objective function can be a combination of cross-entropy loss for token prediction and additional regularization terms.

2.1.3 Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) serves as the foundational paradigm of *feedforward* artificial neural networks [18]. It is characterized by a hierarchical structure of interconnected nodes, called *neurons*, organized into layers: an input layer, one or more hidden layers, and an output layer which provides the final result. A first rudimentary learnable implementation of a neuron can be found in *Rosenblatt's Perceptron* [19], which is a single-layer binary classifier that can only learn linearly separable patterns, inspired by the biological neuron formalization of McCulloch and Pitts [20].

Formally, an MLP is a learning algorithm that learns a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ where n is the number of input features and m is the number of dimensions for output targets.

Neurons are connected by weighted edges, and each neuron applies a *non-linear activation function* σ (e.g., sigmoid, hyperbolic tangent or ReLU) to the weighted sum of its inputs, enabling the network to capture complex patterns and relationships in the data. In particular, for a given layer l, the output vector $h^{(l)}$ can be computed as follows:

$$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)})$$

where $W^{(l)}$ is the weight matrix connecting layer $l-1$ to layer l, $b^{(l)}$ is the bias vector for layer l, and σ is the activation function applied element-wise.

The presence of σ is critical: without non-linearity a multi-layer network would mathematically collapse into a single-layer linear model, as the product of multiple linear transformations is itself a linear transformation.

2.1.4 Encoder-Decoder Architecture

The Encoder-Decoder architecture - formally introduced within the context of Neural Machine Translation (NMT) [21, 22] - is a fundamental neural network design pattern widely used for tasks that involve transforming an input sequence, signal, or representation into a different output representation.

This architectural paradigm is fundamentally bifurcated into two distinct functional components: a compression mechanism (*Encoder*) and a generative mechanism (*Decoder*), mediated by a latent representation (often called *embedding* or *context vector* z) which lies on a lower-dimensional space called *embedding space*.

The encoder $f_{enc} : \mathcal{X} \rightarrow \mathcal{Z}$ is responsible for extracting meaningful features from the input data $x \in \mathcal{X}$ and transforming them into a fixed-length structured latent representation $z = f_{enc}(x)$, also called *code*. In early implementations for NMT tasks [21, 22] the encoder consisted of stacked recurrent neural networks, typically long short-term memory (LSTM) or gated recurrent unit (GRU) architectures (these models process the input sequence sequentially and maintain a hidden state that captures contextual information from previous elements).

The decoder $f_{dec} : \mathcal{Z} \rightarrow \mathcal{Y}$ generates the output sequence $y = f_{dec}(z)$ generally by conditioning on the encoded representation produced by the encoder. Similar to the encoder, early decoder architectures were implemented using recurrent neural networks.

2.2 Transformer

The Transformer architecture, introduced in the seminal paper *"Attention is All You Need"* [9], represents a paradigm shift in sequence modeling and natural language processing. Prior to its inception, the state-of-the-art was dominated by Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which processed data sequentially. The Transformer fundamentally abandoned recurrence in favor of a purely *attention-based mechanism*, enabling unprecedented levels of parallelization and the ability to capture long-range dependencies within data.

2.2.1 Attention Mechanism

Unlike recurrent mechanisms that compress a sequence into a single hidden state, the attention mechanism allows the model to dynamically assign weights to different parts of the input sequence based on their relevance to a specific query.

Formally, given a query $q \in \mathbb{R}^{d_k}$, a set of N key-value pairs $(k, v) \in \mathbb{R}^{d_k} \times \mathbb{R}^{d_v}$ and a similarity function $f : \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \rightarrow \mathbb{R}$, the attention function computes a weighted sum of the values, where the weight $a_i \in \mathbb{R}$ assigned to each value is determined by the similarity between the query and keys [9]:

$$Attention(\, q,\, \{k_i\}_N,\, \{v_i\}_N \,) = \sum_{i=0}^{N-1} a_i \cdot v_i \in \mathbb{R}^{d_v}$$

$$a = softmax(\alpha) = [\, \sigma(\alpha_0), \sigma(\alpha_1), ..., \sigma(\alpha_{N-1}) \,] \in \mathbb{R}^N \qquad \sum_{j=0}^{N-1} a_i = 1$$

$$\alpha = [\, f(q, k_0), f(q, k_2), ..., f(q, k_{N-1}) \,] \in \mathbb{R}^N$$

where $\alpha_i = f(q, k_i) \in \mathbb{R}$ and f nowadays is typically implemented as a scaled dot-product.

In practice matrix formulation is used, in which the attention operation is performed simultaneously on a set of queries packed together into a matrix $Q \in \mathbb{R}^{M \times d_k}$, keys packed together into a matrix $K \in \mathbb{R}^{N \times d_k}$, and values packed together into a matrix $V \in \mathbb{R}^{N \times d_v}$, where N_q, N_k, and N_v are the number of queries, keys, and values respectively. This allows for efficient computation using matrix multiplication and parallelization on modern hardware through the following formulation:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

2.2.2 Multi-Head Attention

A *head* is a computational unit block in which inputs are pre-elaborated by a fully-connected block (also called *projector*) and then attention operation is performed. Fully-connected layers are learnable linear transformations that project the input data into a different space, allowing the model to capture different aspects of the data and learn more complex representations.

Multi-Head Attention (MHA) is a mechanism that allows the model to attend information from different representation subspaces at different positions. It consists of multiple attention heads, each with its own set of learnable parameters, that operate in parallel. As illustrated in Figure 2.1, the outputs of these heads are then concatenated and linearly transformed to produce the final output.

Based on which input is used as query, key and value, different types of attention can be defined:

- **Self-Attention**: the query, key, and value all come from the same source X. This allows the model to capture dependencies within a single sequence, making it particularly effective for tasks like language modeling and machine translation.
- **Cross-Attention**: the query comes from one source X, while the key and value come from another source Y. This is useful for tasks that involve multiple modalities or different

sequences, such as in the encoder-decoder architecture where the decoder attends to the encoder's output.

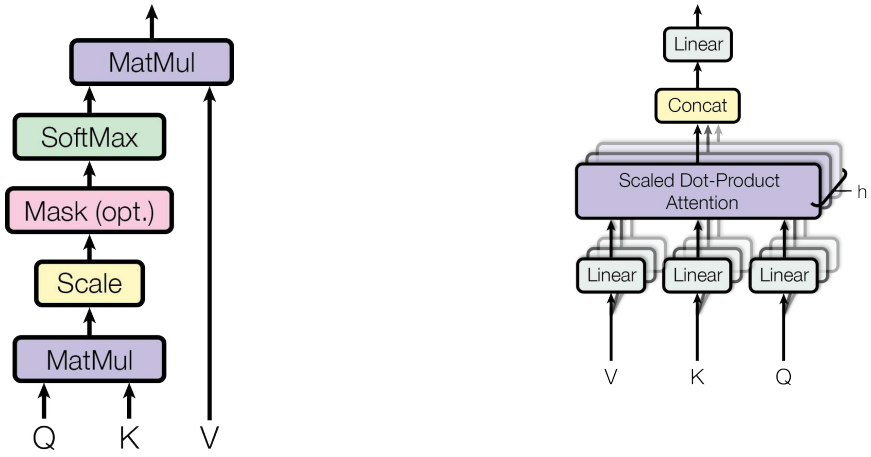


Figure 2.1: On left the Scaled Dot-Product Attention. On right the Multi-Head Attention [9]

2.2.3 Transformer Architecture

Illustrated in Figure 2.2, the original Transformer architecture was designed for sequence-to-sequence machine translation tasks and consists of an encoder and a decoder [9] which have characteristic explained in 2.1.4.

Encoder is composed of a stack of N identical layers, each containing two sub-layers:

- A multi-head self-attention mechanism, which allows the model to attend to different positions within the input sequence to capture contextual relationships.
- A position-wise fully connected feed-forward network, which applies a non-linear transformation to the output of the attention mechanism, enabling the model to learn complex patterns and representations.

Similarly to encoder, the decoder is also composed of a stack of N identical layers, which contain three sub-layers:

- A *causal masked* multi-head self-attention mechanism, which allows the decoder to attend only previous tokens within the encoder's output sequence, avoiding information leakage from future ones.

- A multi-head cross-attention mechanism, which allows the decoder to attend to the output of the encoder, enabling it to incorporate information from the input sequence when generating the output.

- A position-wise fully connected feed-forward network, similar to the one in the encoder, which applies a non-linear transformation to the output of the attention mechanisms.

In each encoder and decoder sub-layer, a residual connection is employed around the MHA, followed by layer normalization. This design choice helps to mitigate the *vanishing gradient problem* - when gradients become too small during the back-propagation due to the depth of the network - allowing to train effectively deep networks.

In addition, since the Transformer architecture does not inherently capture the sequential order of the input data, a *positional encoding* is added to the input embeddings at the bottom of the encoder and decoder stacks - which injects information about the position of each token in the sequence, enabling the model to capture the order of the input data. In the original paper, the authors proposed a sinusoidal positional encoding, which allows the model to learn the relative positions of tokens in the sequence. However, nowadays, learnable positional encodings are more commonly used in modern models [23, 24], which are trained along with the rest of the model parameters.

Embeddings are vector representations of discrete tokens (e.g., words, subwords, or characters) in a continuous vector space. They are obtained firstly parsing the input sequence using a *tokenizer*, which is a pre-processing step that converts raw text into a sequence of tokens belonging to a predefined *vocabulary*. Tokens are then mapped to dense vectors through an embedding layer.

2.2.4 Training

During training, the ground-truth target sequence is provided to the decoder all at once, exploiting parallelization effectively (Figure 2.3). Thanks to the causal mask applied to multi-head self-attention in decoder, the model can attend only to the previous tokens in the output sequence, ensuring that the prediction for a given token is based solely on the preceding tokens and the encoded input representation. In practice mask sets the attention scores of future positions to $-\infty$ before the softmax layer, effectively nullifying them.

To facilitate this process, two special tokens are introduced:

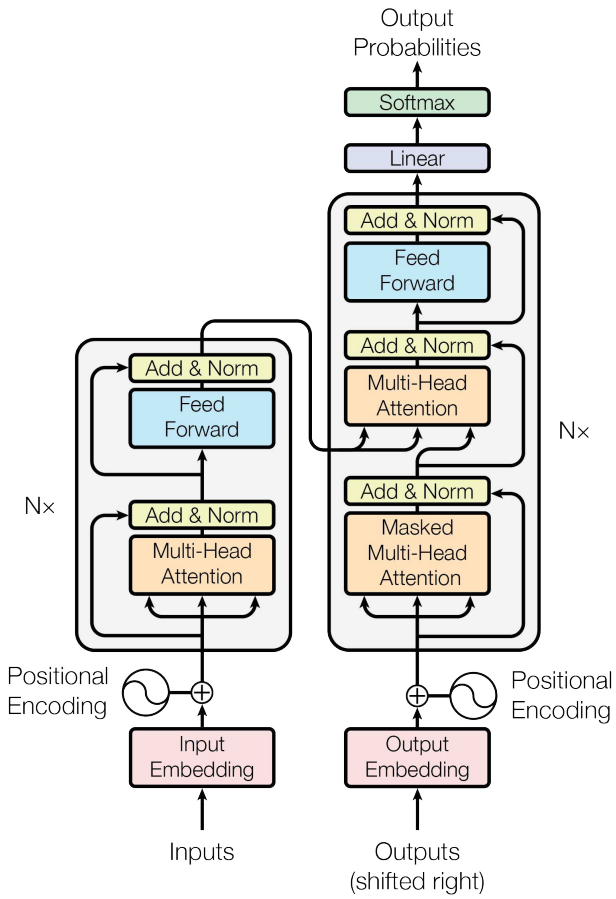


Figure 2.2: The original Transformer architecture introduced in *"Attention is All You Need"* [9].

- **Begin-of-Sequence** (<BOS>): Acts as a trigger for the decoder to start generating text.
- **End-of-Sequence** (<EOS>): Signals that the sequence has reached its conclusion.

Generally, the training lifecycle of a Transformer-based model typically follows a two-stage paradigm: Large-scale Pre-training on unlabeled data, followed by downstream adaptation (*Fine-tuning*) for specific tasks. This approach leverages transfer learning, allowing models to capture general linguistic patterns before specializing in a particular domain [23].

Generally, the pre-training phase involves training the model on a large corpus of text using a self-supervised learning objective - such as masked language modeling or autoregressive tasks - where the specific objective depends on the model architecture [9, 23, 24].

Once pre-trained, the model can be fine-tuned on a smaller, task-specific dataset by adjusting the weights of the model to optimize performance on the new task. Theoretically, fine-tuning involves the entire model, but in practice this has a high computational cost (especially for large models) and could lead to *catastrophic forgetting* [25] - a phenomenon where the model forgets previously learned information when trained on new data.

To mitigate these issues, parameter-efficient fine-tuning (PEFT) techniques have been developed, which involve updating only a small subset of the model's parameters during fine-tuning, such as adapter layers, low-rank factorization, or prompt tuning [26, 27, 28].

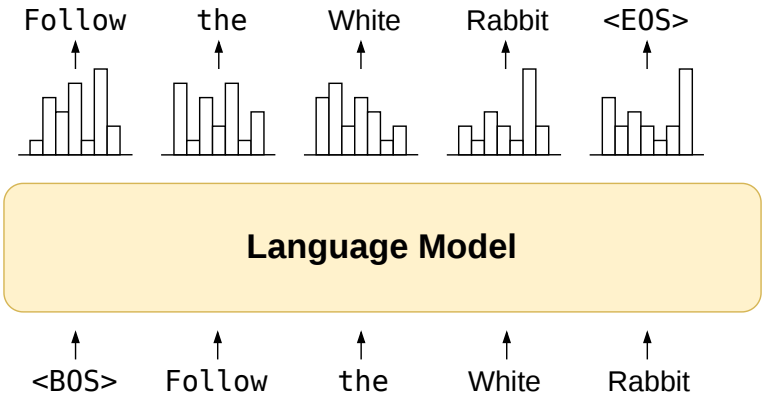


Figure 2.3: Training process of the Transformer architecture.

2.2.5 Inference

A defining characteristic of the Transformer is the fundamental difference between its behavior during the training phase and its operation during inference. In contrast to the parallel nature of training, inference (generation) is *autoregressive*. Since the "ground truth" is not available, the model must generate *one token at a time*, starting from the <BOS> token, using its own previous outputs as context for the next step.

The model produces a probability distribution over the vocabulary for the next token at each step, and the token with the highest probability (or different sampling strategies such as *top-k*) is selected as the output for that step. As already mentioned and illustrated in Figure 2.4, this loop continues until the model generates the <EOS> token, signaling the end of the sequence.

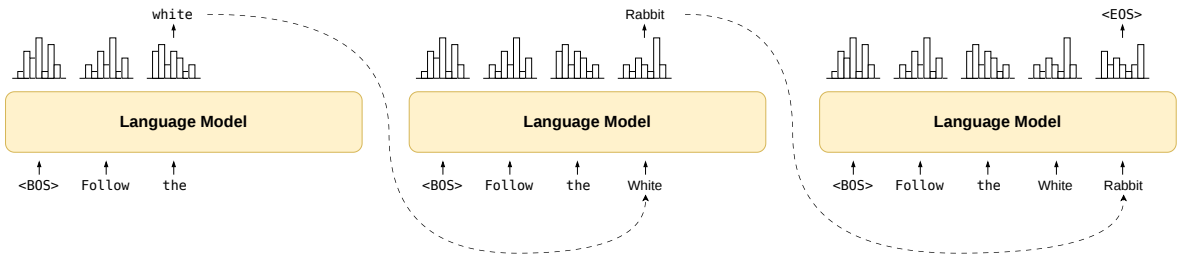


Figure 2.4: Autoregressive generation which starts from the <BOS> token and generates one token at a time until the <EOS> token.

2.3 Vision Encoder

Vision encoders serve as the foundational component for extracting meaningful information from visual data, such as images or videos. In practice, primary objective is to map an image $I \in \mathbb{R}^{C \times H \times W}$ into a compact and informative representation $z \in \mathbb{R}^{d}$ that captures the essential features and semantics of the visual content. Over the last decade, this field has evolved from local feature extractors to global attention-based architectures and, more recently, to multimodal alignment models.

2.3.1 Convolutional Neural Network

Prior to the rise of Transformers, Convolutional Neural Networks (CNNs) were the *de facto* standard for visual representation learning.

Through the application of learnable filters (*kernels*), CNNs perform local operations that capture spatial hierarchies, ranging from low-level edges in early layers to complex object parts in deeper layers. The architecture typically alternates between convolutional layers, non-linear activation functions (e.g., ReLU), and pooling layers (e.g., Max-Pooling) to reduce spatial resolution while increasing feature depth [29].

Despite their efficiency in processing local patterns, CNNs - such as VGG16 [30] - often struggle to capture long-range global dependencies due to their limited receptive field, which only grows linearly with the number of layers.

2.3.2 Vision Transformer

The Vision Transformer (ViT) [31] represents a significant departure from the convolutional paradigm by applying the standard Transformer encoder architecture directly to images with minimal modifications.

In practice, to handle 2D images, the ViT reshapes the input image $I \in \mathbb{R}^{C \times H \times W}$ into a flattened sequence of patches $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^{N \times (P^2 \cdot C)}$ at which positional encoding is added to retain spatial information (Figure 2.5).

In addition, sometimes, a special `[CLS]` vector is prepended to the sequence of patches, which serves as a global representation of the image and, for example, it could be used for classification tasks.

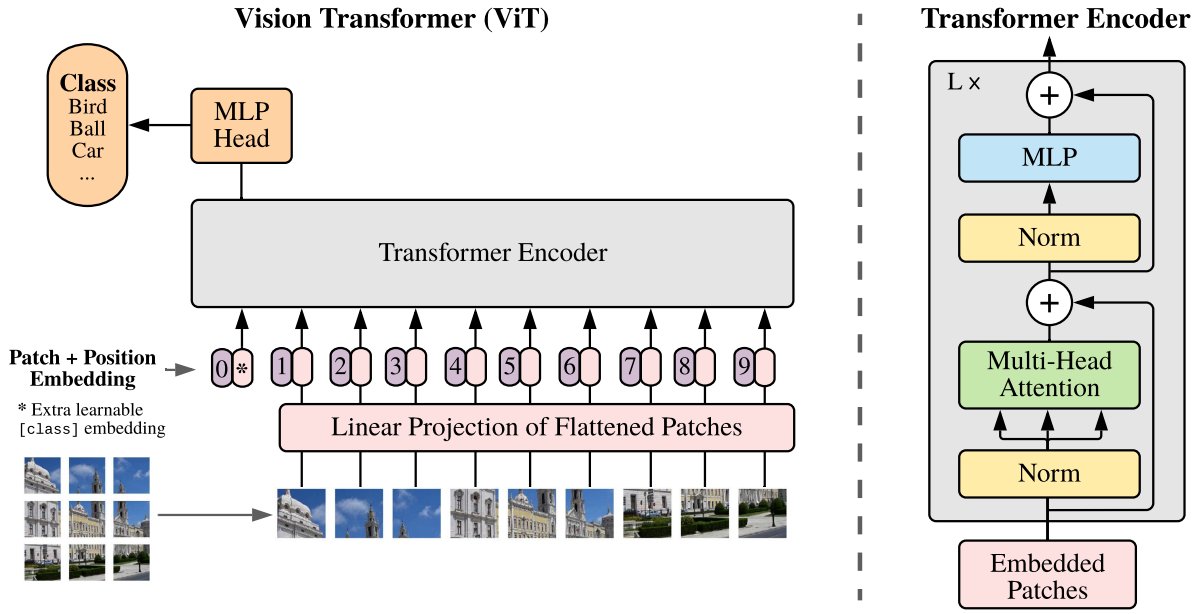


Figure 2.5: Vision Transformer architecture for image classification [32].

2.3.3 Contrastive Language-Image Pre-training (CLIP)

Contrastive Language-Image Pre-training (CLIP) shifted the focus from purely visual representation to *multimodal alignment*. Rather than training a vision encoder on a fixed set of discrete labels (i.e., classes such as *"cat"*, *"dog"*), CLIP is trained on a massive dataset of 400 million image-text pairs collected from the Internet, where paired text is the caption of paired image [33].

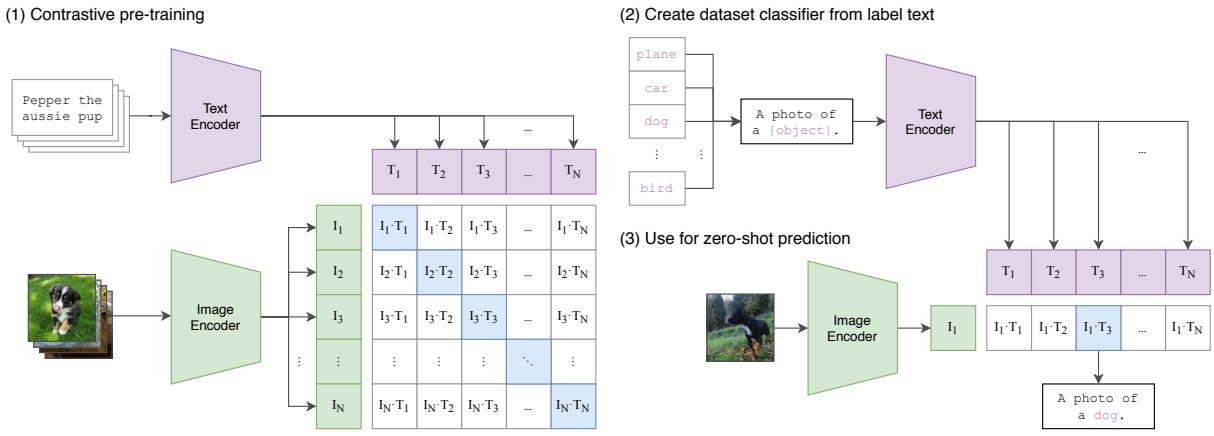


Figure 2.6: Contrastive Language-Image Pre-training (CLIP) alignment methodology and how it enables zero-shot open-vocabulary capabilities in classification task [33].

This approach overcomes the limitations of traditional classification (supervised) learning, in which 1) images must be focused on a single subject in order to simplify the learning signal;

2) single noun labels are poorly descriptive of the image content (which could contain multiple objects, actions, and relationships); 3) supervised datasets are expensive to create due to costly human annotation; and 4) the model is limited to a fixed set of classes (*closed vocabulary*).

CLIP learns to align visual and textual representations in a shared embedding space, maximizing the cosine similarity between right pairs, while minimizing the similarity with other captions in the batch. This approach is also known as *contrastive learning*, because the model learns to distinguish between correct (*positive*) and incorrect (*negative*) pairs.

In practice, given a batch size of N image-text pairs, the model computes the cosine similarity between all possible image-text embeddings combinations exploiting matrix multiplication, resulting in a similarity matrix of size $N \times N$.

Embeddings are computed using a vision encoder (e.g., ViT with `CLS` token or ResNet with attention pooling) for images and a text encoder (e.g., BERT) for captions [33]. Both encoders must produce an input global representation of the same dimension d to allow for the alignment in the shared embedding space.

Zero-shot Capabilities

CLIP demonstrates remarkable zero-shot capabilities, allowing it to perform various downstream tasks without any task-specific fine-tuning. Common zero-shot evaluation task is image classification, where the model is evaluated on its ability to classify images into categories it has never seen during training. This particular task is often called *open vocabulary* classification [33].

In practice, as illustrated in Figure 2.6, text prompts are used to define the classes of interest, such as *"a photo of a cat"* or *"a photo of a dog"*, and the model computes the cosine similarity between the image embedding and the embeddings of these prompts to determine the most likely class for the image. More advanced prompting techniques, such as prompt engineering and prompt tuning, have been developed to further enhance the zero-shot performance [7].

Limitations

Despite the robust zero-shot capabilities of CLIP, the architecture exhibits specific structural and semantic limitations that influence how it is integrated into downstream multimodal systems.

Firstly, given that CLIP is trained following a contrastive learning paradigm, the model needs to be trained on a large batch size to ensure a sufficient number of negative samples for effective learning. This requirement can lead to significant computational costs and memory constraints,

due to communication overhead and the need to store large batches of data in memory, especially when scaling up the model or using high-resolution images [33].

Secondly, CLIP is primarily optimized for global semantic alignment rather than fine-grained spatial understanding. For this reason, CLIP often struggles with tasks requiring precise localization or understanding of spatial relationships.

Thirdly, the final (linear) projection layer used to map visual features into the shared embedding space is essential for maximizing cosine similarity with text embeddings, but it acts as a semantic bottleneck. The projection process compresses the visual signal to discard information that is not strictly necessary for text-image matching. Consequently, fine-grained structural details - such as texture, precise edges, and local geometry - are often filtered out in favor of high-level concepts.

To mitigate the loss of spatial information, many modern models utilize the features from the penultimate layer, which preserves a higher degree of visual "grounding", because these features have not yet been condensed into the joint text-vision space [3].

Instead, to mitigate large batch size requirements, CLIP-inspired alternatives such as SigCLIP have been developed, - which leverage a sigmoid-based contrastive loss on individual image-text pairs rather than a softmax-based loss on batches of pairs - enabling effective training with smaller batch sizes [34].

2.3.4 Self-Distillation with No-Label (DINO)

DINO (Self-Distillation with No-Labels) represents a paradigm shift in self-supervised learning for Vision Transformers. Unlike CLIP, which relies on large-scale multimodal supervision, DINO operates exclusively on purely unlabelled images using a knowledge *distillation* setup.

Distillation refers to the process of transferring knowledge from a generally larger and more complex model (the *teacher*) to a generally smaller, simpler model (the *student*).

In DINO teacher and student networks consist of two structurally identical networks. The framework processes two different random crops of the same input image. The student network is trained to match the output of the teacher network [8].

One of the most remarkable findings of the DINO framework is that, when applied to Vision Transformers, the self-attention maps inherently learn explicit semantic segmentation of the scene. In other words, the model effectively "discovers" object boundaries without any human-provided labels or masks (Figure 2.7).

In particular the attention heads in the last layer of a DINO-trained ViT naturally focus on coherent semantic regions (e.g., the foreground object). This makes DINO-encoded features particularly valuable for downstream tasks that require localized spatial awareness, serving as a powerful alternative to the more "text-centric" representations of CLIP [8].

Figure 2.7: DINO self-attention maps in a trained Vision Transformer [8].

2.4 Large Language Model

The landscape of natural language processing has been fundamentally redefined by the emergence of Large Language Models (LLMs). These models, characterized by billions of parameters and trained on trillion-token corpora, have transitioned from simple text predictors to versatile reasoners capable of zero-shot generalization. Modern LLMs primarily adopt a decoder-only Transformer architecture, optimized for efficient autoregressive generation [24].

By default, LLMs lack conversational capabilities, but they can be adapted to dialogue-based interactions through fine-tuning on conversational datasets. Moreover, the introduction of instruction tuning and *Reinforcement Learning with Human Feedback* (RLHF) has further enhanced their ability to follow user instructions and engage in more natural conversations [35].

RLHF is a training technique that leverages human feedback to optimize the model's behavior, particularly in terms of safety and alignment with human values. This process involves collecting human feedback on model outputs, training a reward model to predict this feedback, and then fine-tuning the language model using reinforcement learning algorithms to maximize the predicted reward, resulting in a model that is more aligned with human preferences and less likely to produce harmful or undesirable outputs [35].

2.4.1 LLaMA

Introduced by Meta AI in early 2023, LLaMA is a family of LLMs which represented a pivotal moment for the democratization of AI research. Unlike contemporary proprietary models (e.g., GPT-4), LLaMA was designed to be efficient, enabling high-performance inference on consumer-grade hardware while maintaining state-of-the-art capabilities [36].

The first LLaMA architecture introduces three critical modifications to the original Transformer design to improve training stability and performance:

- **Pre-normalization**: To improve training stability, LLaMA normalizes the input of each transformer sub-layer instead of the output. It utilizes the Root Mean Square Layer Normalization (RMSNorm), which provides similar benefits to standard LayerNorm but with lower computational overhead.
- **SwiGLU Activation Function**: Standard ReLU is replaced with the SwiGLU activation function. This hybrid approach combines the properties of *Swish and Gated Linear Units*, allowing the model to learn more complex representations.
- **Rotary Positional Embeddings (RoPE)**: This technique encodes positional information by rotating the Query and Key representations in a complex plane, naturally capturing relative distances between tokens and allowing for better context window extrapolation.

LLaMA 2

Later in 2023, LLaMA 2 was released, which builds upon the original architecture with improvements in training data quality and model scaling, designed to bridge the gap between research-oriented models and production-ready systems [37]. In particular, major contributions of LLaMA 2 include:

- **Training Scale and Context**: While the original LLaMA was trained only on public data, LLaMA 2 was trained on a dataset of 2 trillion tokens (40 % more than LLaMA 1) composed of a mixture of publicly available and Meta's internal data, primarily focused on data quality and diversity. Furthermore, the context window was doubled from 2,048 to 4,096 tokens, facilitating better performance on long-form generation and document analysis.

- **Grouped-Query Attention** (GQA): To optimize inference latency, the 70B parameter variant incorporated Grouped-Query Attention. By sharing key and value projections across multiple heads, GQA significantly reduced the memory bandwidth requirements during decoding, allowing for more efficient execution on consumer-grade hardware compared to standard MHA blocks.

- **Safety and RLHF**: A cornerstone of LLaMA 2 was the introduction of RLHF to enhance the model's safety and alignment with human values.

LLaMA 3

Finally, LLaMA 3 series (and its subsequent 3.1 and 3.2 updates) was released in late 2024 [2] and it represents the latest iteration in the LLaMA family. This generation focuses on the principle that model performance is predominantly driven by data volume and quality, even at relatively modest parameter counts.

This philosophy can be observed directly in the major contributions of LLaMA 3, which include:

- **15 Trillion Token Pre-training**: The most significant advancement in LLaMA 3 is the scale of its pre-training regime. Trained on over 15 trillion tokens—predominantly high-quality, non-English, and code-centric data—the model demonstrates superior reasoning, coding, and multilingual capabilities.

- **Tokenizer Optimization**: LLaMA 3 transitioned to a 128k-token vocabulary and this expansion allows for more efficient text encoding, resulting in better compression and improved performance on technical and multi-lingual datasets compared to the 32k-token vocabulary used in previous versions.

- **Universal GQA and Context Expansion**: Unlike its predecessor, LLaMA 3 implemented Grouped-Query Attention across all model sizes (8B and 70B). With the release of LLaMA 3.1, the context window was expanded to 128k tokens, enabling the processing of entire technical manuals or complex codebases in a single prompt.

2.4.2 Vicuna

While the base LLaMA model is a powerful text completer, it lacks the ability to follow complex instructions or engage in natural dialogue. Vicuna was developed to bridge this gap, becoming one of the first open-source models to achieve performance parity with proprietary assistants like OpenAI's ChatGPT [12].

The primary innovation of Vicuna is its training data. It was fine-tuned starting from a LLaMA-13B base using approximately 70,000 user-shared conversations from ShareGPT. This dataset consists of high-quality interactions where humans interact with ChatGPT, providing the model with a rich signal for complex instruction-following and multi-turn dialogue [12].

The training process utilized Supervised Fine-Tuning (SFT), where the model is trained to predict the assistant's responses given the user's prompt. To handle long-form conversations, the researchers extended the context length and optimized the training cost through the use of FSDP (Fully Sharded Data Parallel) and gradient checkpointing [12].

2.4.3 Qwen

Qwen is a family of foundation models developed to advance performance, inference efficiency, and multilingual capabilities. The latest iteration - Qwen3 released in 2025 - represents a significant evolution in the open-weight landscape by bridging the gap between open models and proprietary flagships. Qwen3 encompasses a diverse series of dense and *Mixture-of-Experts* (MoE) architectures - a technique that allows for dynamic activation of different specialized subsets of the model's parameters based on the input [38] - ranging in scale from 0.6 to 235 billion parameters and expanding multilingual support to 119 languages and dialects [1].

The fundamental Qwen3 architecture builds upon its predecessors by utilizing Grouped Query Attention (GQA), SwiGLU activation functions, Rotary Positional Embeddings (RoPE) and pre-normalization via RMSNorm. To enhance functionality and usability, Qwen3 introduces several critical architectural and training innovations:

- **Unified Operating Framework**: A major breakthrough in Qwen3 is the integration of a *thinking mode* (for complex, multi-step reasoning) and a *non-thinking mode* (for rapid, context-driven responses) into a single unified framework. This eliminates the need to alternate between separate chat-optimized and dedicated reasoning models, enabling dynamic mode switching via chat templates.

- **Thinking Budget Mechanism**: To give users fine-grained control over inference-time scaling, Qwen3 incorporates a *thinking budget*. This adaptive mechanism allows users to allocate computational resources by setting a token threshold during reasoning tasks, efficiently balancing latency and performance based on task complexity.

- **Attention Mechanism Stability (QK-Norm)**: While removing the QKV-bias present in older iterations, Qwen3 introduces Query-Key Normalization (QK-Norm) to the attention mechanism. This architectural refinement ensures stable training dynamics across large parameter scales.

- **Exclusion of Shared Experts and Global Balancing**: Unlike previous MoE iterations in the family, the flagship Qwen3 MoE models exclude shared experts entirely, utilizing 128 total experts with 8 activated per token. To encourage fine-grained expert specialization and structural efficiency, a global-batch load balancing loss is implemented during training.

- **Strong-to-Weak Distillation**: To optimize lightweight edge models, Qwen3 utilizes an advanced off-policy and on-policy distillation pipeline from flagship teacher models. By minimizing Kullback-Leibler (KL) divergence against teacher logits, smaller models inherit robust reasoning and mode-switching capabilities at a fraction of the computational training cost.

2.5 Object-Centric Learning

Traditional Deep Learning architectures have demonstrated remarkable success in visual recognition tasks. However, these models learn to distinguish different objects based on statistical regularities in the data, without explicitly representing objects as discrete entities. In contrast, human perception is inherently organized around discrete entities or "objects", which serve as the fundamental building blocks for reasoning, planning, and systematic generalization [39].

Object-Centric Learning (OCL) aims to bridge this gap by introducing inductive biases that encourage neural networks to decompose complex scenes into a set of distinct, reusable representations, often referred to as "slots" or "entities" [39, 10].

Formally, OCL seeks to learn a mapping from a high-dimensional input space $\mathcal{X}$ to a set of K latent variables $\{ s_1, ..., s_K \}$, where each slot s_i ideally corresponds to a single object in the scene. A defining characteristic of OCL is *permutation invariance*: since the order of objects

in a scene is arbitrary, the internal representations should be exchangeable.

Nowadays, the most widely adopted architecture for OCL is *Slot Attention*, which utilizes an iterative attention mechanism to bind input features to a fixed number of slots [10]. While Slot Attention has shown impressive results on synthetic datasets, it struggles to scale to real-world images due to its reliance on pixel-level reconstruction, which is often dominated by high-frequency textures and background details rather than object-level semantics.

2.5.1 DINOSAUR

DINO-based Slot Attention U-Reconstruction (DINOSAUR) represents a pivotal shift in object-centric learning, specifically designed to bridge the gap between synthetic toy datasets and complex real-world imagery. While traditional Slot Attention methods rely on pixel-level reconstruction [10] - which often fails in natural scenes due to high-frequency textures and intricate backgrounds - DINOSAUR instead reconstructs the semantically rich feature space of a pre-trained self-supervised model [40].

In particular, the framework utilizes a frozen pre-trained DINO encoder to extract input features $X \in \mathbb{R}^{N \times D_{feat}}$. These features are passed through a standard Slot Attention bottleneck to be aggregated into a fixed set of slots $S = \{s_1, \ldots, s_K\} \in \mathbb{R}^{K \times D_{slots}}$. DINO vision encoder was chosen because of its unique ability to learn semantic segmentation without supervision, as discussed in Section 2.3.4, which provides a powerful inductive bias for object discovery in complex scenes.

Crucially, the decoding process aims to predict the original DINO features rather than RGB values. The authors propose and contrast two distinct feature decoder architectures engineered specifically to handle the reconstruction of self-supervised target vectors rather than raw pixel data. The first variant is the *Spatial Broadcast Decoder* (SBD) - a computationally efficient MLP that projects individual slot vectors across patch dimensions, augmented with learned positional encodings, and merging them token-wise through an alpha maps. The second variant introduces a more expressive autoregressive Transformer decoder that decodes patch features jointly across all slots.

By supervising the slots to reconstruct semantic embeddings — which naturally ignore low-level noise and pixel-perfect details — the model is forced to focus on the high-level compositional structure and semantic entities within the scene. The reconstruction loss is defined as the mean squared error in the feature space: $L_{recon} = \|\hat{F} - F\|^2$, where F are

the ground-truth DINO features and $\hat{F}$ is the reconstructed feature set. This approach has demonstrated state-of-the-art results on challenging unsupervised object discovery benchmarks like MS COCO [41] and PASCAL VOC [42].

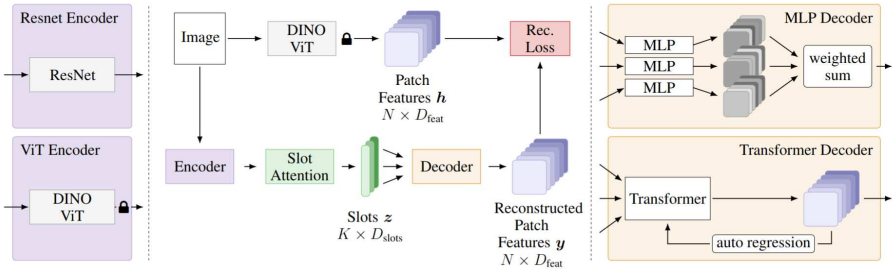


Figure 2.8: Overview of the proposed DINOSAUR architecture [40].

However, DINOSAUR retains some of the fundamental limitations of the Slot Attention mechanism. Exactly as its predecessors, it requires a pre-defined fixed number of slots K as a dataset-dependent hyperparameter, leading to potential over-segmentation or under-segmentation when the actual object count varies across instances. This is well illustrated in Figure 2.9, where the same image is processed using models trained with different slot counts, resulting in varying degrees of fragmentation or merging of objects.

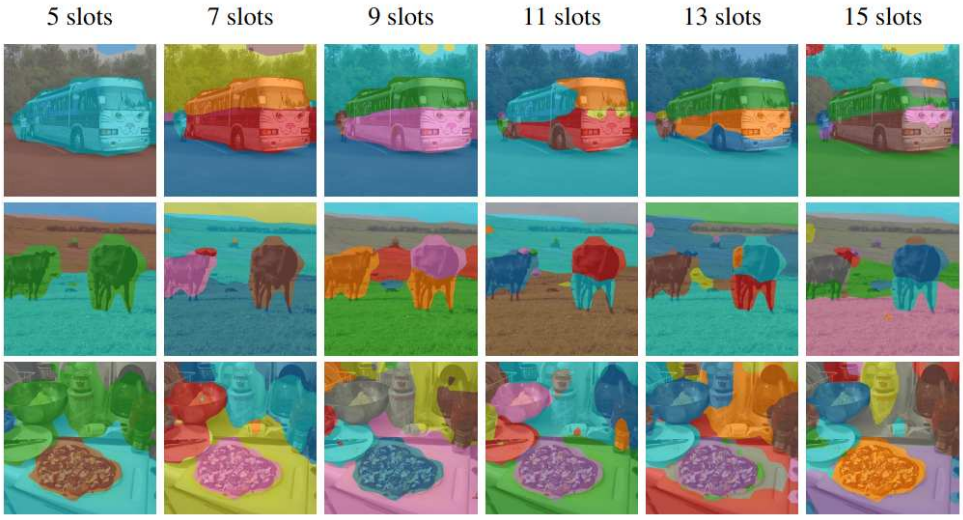


Figure 2.9: Three scenes of low/medium/high complexity on COCO 2017, varying the number of slots during training [40].

Furthermore, because it relies on pre-trained DINO features, its discovery capabilities are inherently biased toward the semantic categories learned during the self-supervised pre-training phase, which may result in the neglect of smaller objects or regions that lack semantic distinctness

in the DINO feature space. Finally, the use of a frozen encoder limits the model's ability to refine its feature representations specifically for the task of object discovery, potentially preserving biases or noise patterns from the original DINO backbone [40].

2.5.2 AdaSlot

Adaptive Slot Attention (AdaSlot) addresses the primary architectural limitation of standard Slot Attention, introducing a complexity-aware object auto-encoder framework designed to dynamically determine the optimal number of slots based on the specific content of an image, avoiding the necessity of predefining a fixed number of slots K, which fails to account for the inherent variability of objects in natural scenes [43].

In particular, as illustrated in Figure 2.10, the framework extracts image features $F \in \mathbb{R}^{H' \times W' \times D}$ using a vision encoder - a frozen self-supervised backbone like DINO was proposed - and initializes a set of K_{max} potential slots. Rather than utilizing all slots for reconstruction, AdaSlot employs a discrete slot sampling module that selects a subset of slots $S_{sub} \subseteq S$. For each slot S_i, a lightweight neural network predicts a keeping probability π_i, which is then used to generate a hard binary decision mask $Z \in \{0, 1\}^{K_{max}}$ via a Gumbel-Softmax operation to maintain end-to-end differentiability [44].

Crucially, the reconstruction is performed by a masked slot decoder $\hat{x} = f_{dec}(S, Z)$ that suppresses the information of unselected slots - typically by forcing their corresponding decoder's alpha masks to zero. The training objective incorporates a complexity-aware regularization term which penalizes the model for utilizing an excessive number of slots. This forces the mechanism to balance reconstruction quality with slot sparsity, enabling the model to effectively merge redundant slots and adapt the representation to the complexity of the instance [43].

However, AdaSlot exhibits certain limitations when applied to complex, real-world environments. The model primarily excels in scenarios with clearly defined and well-segmented objects such as CLEVR [45], but it encounters challenges in densely packed scenes such as MS COCO [41], where cluttered compositions and incomplete annotations may lead to discrepancies between learned objects and manual labels. Additionally, the complex part-whole hierarchies inherent in natural scenes remain difficult to resolve, often resulting in segmentations that do not perfectly align with human-annotated structural granularities.

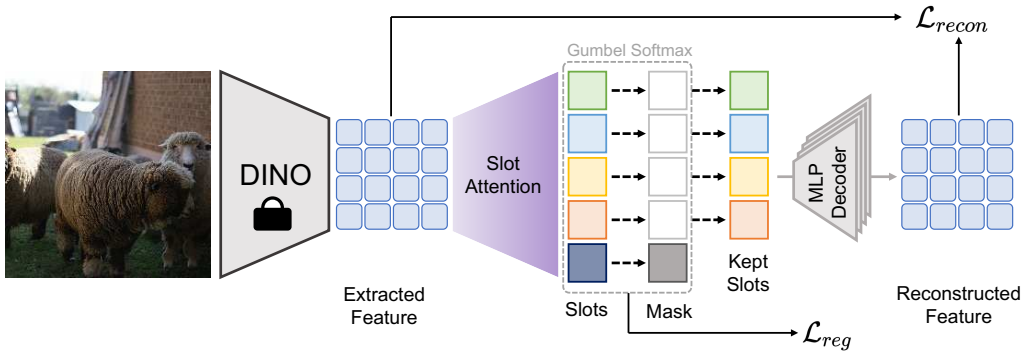


Figure 2.10: Adaptive Slot Attention proposed architecture [43].

2.5.3 MetaSlot

MetaSlot represents an alternative architecture with respect to AdaSlot to overcome the limitations of a fixed slot count and random initialization in regular Slot Attention. Unlike standard architectures that rely on static hyper-parameters, MetaSlot utilizes a global vector-quantized (VQ) codebook that stores idealized object prototypes, providing a unified prior to guide slot initialization [46].

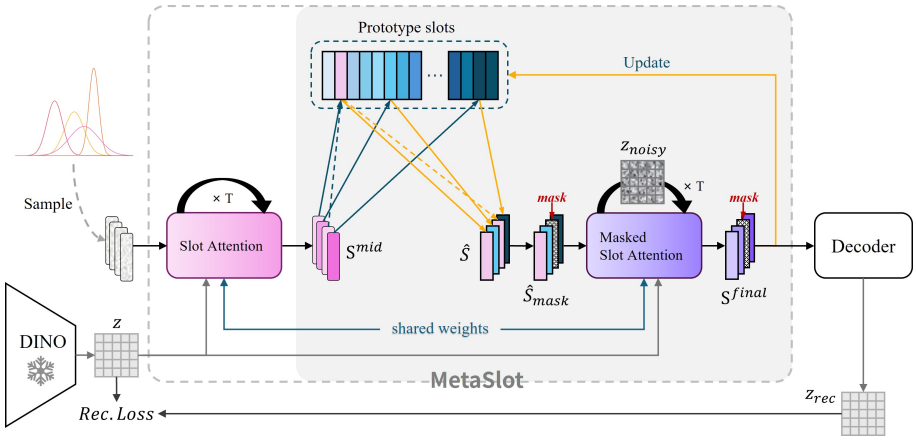


Figure 2.11: MetaSlot proposed architecture [46].

As illustrated in Figure 2.11, the architecture operates through a two-stage aggregation framework. In the first stage, input features extracted by an encoder - authors proposed DINOv2 similarly to DINOSAUR and AdaSlot models - are aggregated via standard Slot Attention to produce a set of intermediate slots S^{mid}. These vectors are matched against the global codebook using a nearest-neighbor search, resulting in a set of semantically consistent, discrete slots $\hat{S}$. To handle scene variability, a de-duplication mask is applied to elements in $\hat{S}$ picking the same

prototype, retaining only distinct "object-aware" slots while pruning redundant representations to yield the masked subset $\hat{S}_{\text{mask}}$.

The second stage refines this masked subset $\hat{S}_{\text{mask}}$ through *Masked Slot Attention* (MSA) to generate the refined slots S^{final}. Crucially, the model injects progressively attenuated isotropic Gaussian noise into the features during this stage. This mechanism, interpreted as implicit simulated annealing, encourages exploration of the latent space in early iterations and facilitates precise alignment in the final steps. To ensure stability and identifiability, the codebook is updated via a mini-batch K-means strategy using *Exponential Moving Averages* (EMA).

MetaSlot consistently improves performance when integrated into diverse decoding paradigms, including Transformer-based (SLATE [47]), MLP-based (DINOSAUR [40]), and diffusion-based frameworks. Comparative evaluations on real-world datasets like MS COCO and PASCAL VOC demonstrate that MetaSlot effectively mitigates over-segmentation issues and produces more interpretable, semantically coherent slot representations than standard Slot Attention [10] or advanced variants like AdaSlot [46].

However, the framework faces several structural limitations: 1) the reliance on absolute positional encoding inherited from the original Slot Attention makes the model non-equivariant to image translations, which can cause prototypes to capture position-dependent noise; 2) the current prototypes primarily encode global shape or semantic information, often overlooking finer-grained attribute compositions. MetaSlot's two-stage iterative design also introduces computational complexity - especially due to the quantized codebook; 3) while it shows strong adaptability, its robustness under extreme out-of-distribution (OOD) conditions has not yet been extensively studied [46]; 4) slot pruning approach relies on the flawed assumption that an image contains only a single instance of each object category. Under this logic, when multiple slots associate with the same object type, only one is retained while the others are discarded. However, real-world scenes frequently feature multiple distinct objects of the identical type - such as a street containing several different cars - causing this pruning method to mistakenly eliminate valid object representations.

2.6 Multimodal Large Language Model

Multimodal Large Language Models (MLLMs) represent an important shift in generative intelligence, extending the advanced reasoning and linguistic capabilities of Large Language Models

(described in Section 2.4) to encompass multiple modalities such as vision, audio, and video. By bridging the gap between perceptual features and textual tokens, MLLMs facilitate a unified dialogue-based interface that supports complex content understanding and instruction-following across diverse data types.

A standard MLLM architecture typically integrates three primary components: a pre-trained LLM, a modality-specific encoder and a cross-modal connector. In vision-centric configurations, a Vision Transformer — frequently pre-trained CLIP or DINO — is employed to extract high-level visual features from an input image. These features are then mapped into the language model's embedding space by a cross-modal connector, such as a Multi-Layer Perceptron or a cross-attention module, which transforms the visual representations into compact, LLM-understandable tokens. Finally, the pre-trained LLM is fine-tuned to align these new visual tokens with its existing linguistic knowledge, enabling the system to perform complex multimodal reasoning and instruction-following tasks.

Visual tasks are historically the most explored in the literature:

- **Visual Question Answering** (VQA): The model answers complex questions regarding the contents of an image or video, requiring a synthesis of perceptual identification and logical reasoning.
- **Image Captioning**: Models identify objects, spatial relationships, and temporal trends to generate semantically rich descriptions of a scene.
- **Visual Dialogue**: MLLMs engage in multi-turn conversational interactions centered on visual data, maintaining contextual coherence across the dialogue.

2.6.1 Flamingo

Flamingo represents a pioneering MLLM architecture, specifically designed to excel at few-shot learning on interleaved sequences of images, videos, and text. By intelligently leveraging pre-trained, heavily optimized, and subsequently frozen vision and language components, Flamingo seamlessly adapts to a wide variety of downstream multimodal tasks using only a handful of examples in its input prompt. This architecture effectively bridges the gap between large-scale generative text models and specialized visual-language systems [4].

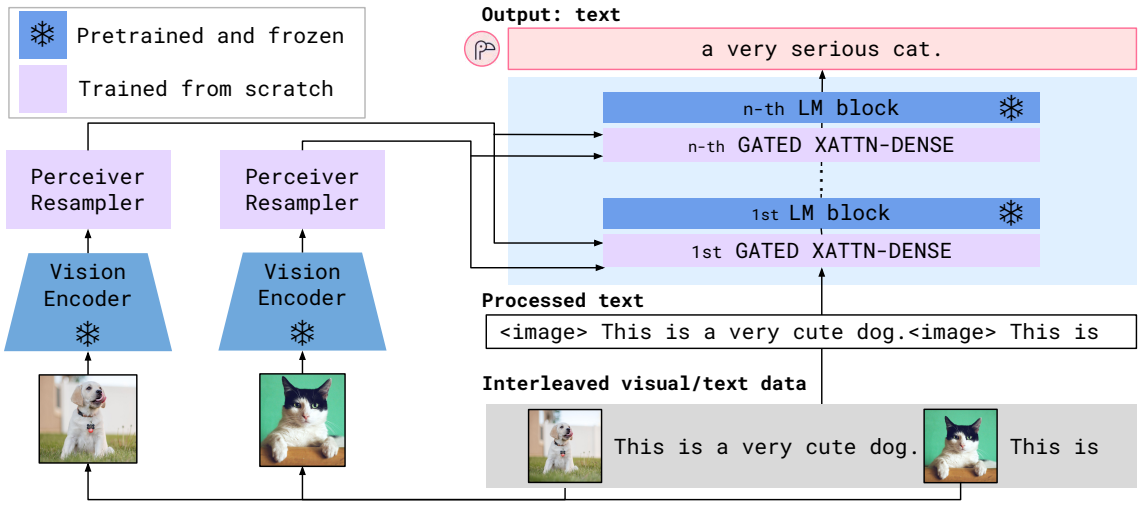


Figure 2.12: Flamingo architecture [4].

The overarching architecture, illustrated in Figure 2.12, is defined by its hybrid approach to training and inference. It integrates a frozen vision encoder, a novel trainable component called *Perceiver Resampler*, and a frozen LLM equipped with newly initialized, trainable cross-attention layers. By keeping the foundational models frozen, Flamingo retains the robust, zero-shot generalization capabilities of both the vision and language models while focusing computational resources on learning the cross-modal alignment.

The process begins with the vision encoder, which is typically a pre-trained Normalizer-Free ResNet (NFNet) or a Vision Transformer. When Flamingo receives an interleaved multi-modal prompt, it extracts the visual elements (either static images or video frames) and passes them through this frozen encoder. The encoder processes these inputs to extract dense, high-dimensional spatio-temporal feature maps.

However, these raw visual feature maps present a significant architectural challenge: their size varies depending on the resolution of the image or the duration of the video, and the sheer number of spatial tokens would overwhelm the context window of the language model, leading to quadratic increases in computational complexity during attention operations.

This bottleneck is elegantly resolved by the Perceiver Resampler - detailed in Figure 2.13 - that is a transformer-based module designed to ingest a variable number of spatio-temporal visual features and compress them into a strict, fixed-length sequence of visual tokens (typically 64 tokens per image or frame).

Specifically, the raw visual features - denoted as X_f with shape (T, S, d), where T is time, S is spatial dimension, and d is depth - are augmented with learned time-positional embeddings

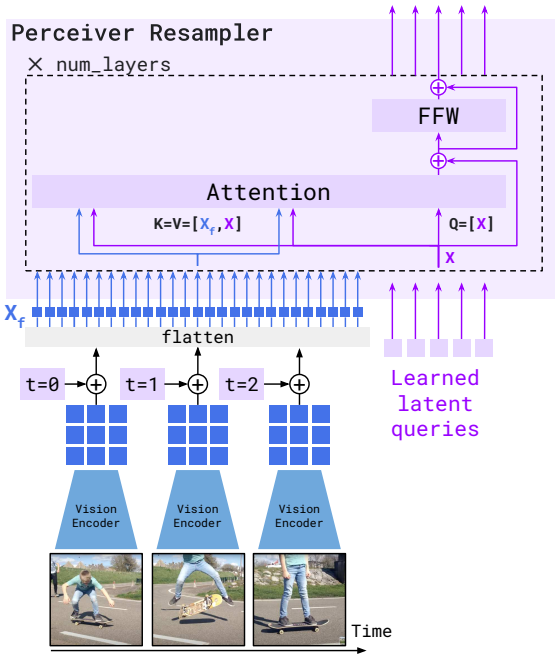
to retain temporal awareness, which is critical for video understanding. These features are then flattened into a 2D tensor of shape $(T \times S, d)$. Simultaneously, the model initializes a fixed set of learned latent queries - denoted as x with shape $[R, d]$, where R is the fixed number of desired output tokens (e.g., 64). During the forward pass of the Perceiver Resampler, an asymmetric attention mechanism is applied over several layers. The queries Q are formed exclusively from the learned latents x, while the keys K and values V are formed by concatenating the flattened visual features with the latents: $K = V = [X_f, x]$. This cross-attention allows the small set of latent queries to actively aggregate the massive set of visual features, extracting only the most semantically relevant information. Following the attention step, the features pass through a standard Feed-Forward Network (FFW). After enough iterations, the output is a highly condensed, fixed-size representation of the visual scene [4].

Once the visual data is compressed into this manageable set of tokens, it must be fused into the language generation process. Flamingo achieves this without altering the pre-trained weights of the LLM by utilizing *Gated Cross-Attention* layers, strategically inserted between the existing, frozen layers of the language model, as depicted in Figure 2.14.

In these newly added layers, the text tokens Y act as queries, while the compressed visual tokens X_{pr} from the Perceiver Resampler act as keys and values. This mechanism enables the language model to selectively attend to specific visual details only when necessary to predict the next word. To ensure training stability, Flamingo employs a hyperbolic tangent (tanh) gating mechanism. The outputs of both the cross-attention layer and its subsequent dense feed-forward layer are multiplied by a learned gating parameter α - which is explicitly initialized to 0. Mathematically, the update rule is $Y' = Y + \tanh(\alpha) \cdot Attention(Q = Y, K = X_{pr}, V = X_{pr})$.

Thanks to $\tanh(0) = 0$, the newly initialized cross-attention layers essentially output zero at the start of training. This acts as an identity function, allowing the model to perfectly mimic the original frozen language model at step zero, preventing catastrophic forgetting and ensuring a smooth training curve as the gating parameters slowly learn to incorporate visual context [4].

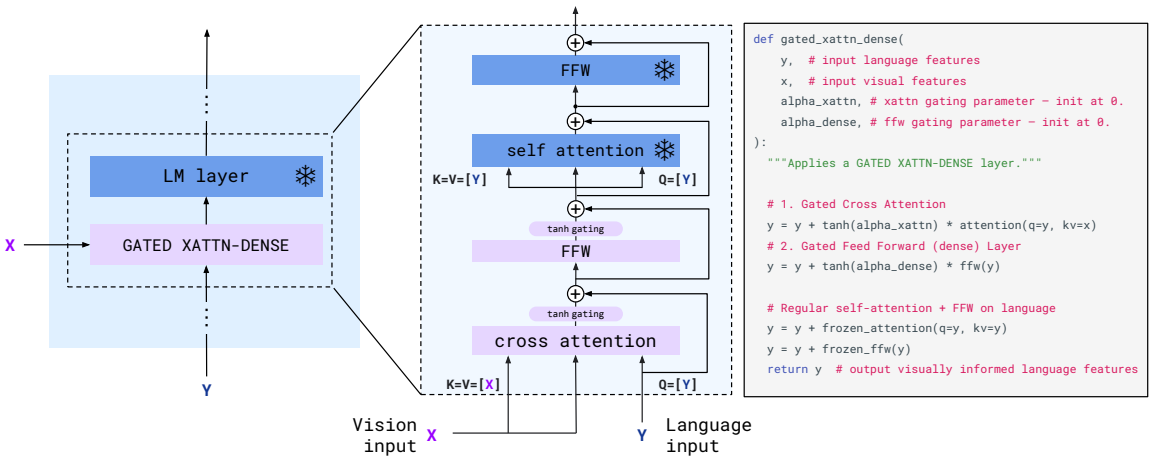
The combination of the Perceiver Resampler and Gated Cross-Attention allows Flamingo to seamlessly process interleaved text and images, enabling the rich multi-turn conversational capabilities and Visual Question Answering (VQA). The model naturally learns to associate the `<image>` tag in its text input with the corresponding 64 tokens output by the Perceiver Resampler, establishing a fluid multimodal context.




```python
def perceiver_resampler(
    x_f,  # The [T, S, d] visual features (T=time, S=space)
    time_embeddings,  # The [T, 1, d] time pos embeddings.
    x,  # R learned latents of shape [R, d]
    num_layers,  # Number of layers
):
  """The Perceiver Resampler model."""

  # Add the time position embeddings and flatten.
  x_f = x_f + time_embeddings
  x_f = flatten(x_f)  # [T, S, d] -> [T * S, d]
  # Apply the Perceiver Resampler layers.
  for i in range(num_layers):
    # Attention.
    x = x + attention_i(q=x, kv=concat([x_f, x]))
    # Feed forward.
    x = x + ffw_i(x)
  return x
```

Figure 2.13: Flamingo perceiver resampler [4].




```python
def gated_xattn_dense(
    y,  # input language features
    x,  # input visual features
    alpha_xattn, # xattn gating parameter – init at 0.
    alpha_dense, # ffw gating parameter – init at 0.
):
  """Applies a GATED XATTN-DENSE layer."""

  # 1. Gated Cross Attention
  y = y + tanh(alpha_xattn) * attention(q=y, kv=x)
  # 2. Gated Feed Forward (dense) Layer
  y = y + tanh(alpha_dense) * ffw(y)

  # Regular self-attention + FFW on language
  y = y + frozen_attention(q=y, kv=y)
  y = y + frozen_ffw(y)
  return y  # output visually informed language features
```

Figure 2.14: Flamingo gated cross-attention mechanism [4].

Despite its architectural brilliance, Flamingo faces several inherent limitations typical of large-scale multimodal systems. The reliance on a frozen, massive LLM introduces significant computational and memory overhead during inference, making it difficult to deploy in resource-constrained environments. Additionally, the model remains susceptible to *hallucinations* - when the LLM confidently generates text that is not grounded in the provided visual input. It also frequently struggles with tasks requiring fine-grained spatial reasoning, reading small text within images, or detecting tiny objects. This granularity issue is a direct, unavoidable trade-off of the Perceiver Resampler: while compressing high-resolution spatial data into a mere 64 tokens is

computationally efficient, it acts as an aggressive information bottleneck, inherently filtering out microscopic details. Finally, Flamingo's few-shot performance remains highly sensitive to the quality, quantity, and sequence of the provided examples, necessitating careful and deliberate prompt engineering to achieve optimal, reliable results [4].

2.6.2 BLIP/BLIP-2

BLIP (Bootstrapping Language-Image Pre-training) and its successor BLIP-2 constitute a highly influential framework in the field of multimodal learning, designed to efficiently bootstrap vision-language pre-training from off-the-shelf frozen models. While the original BLIP focused on a unified mixture of encoder-decoder architectures and data bootstrapping to handle noisy web data, BLIP-2 introduces a more modular and computationally efficient strategy. By acting as a bridge between a frozen image encoder and a frozen LLM, BLIP-2 reached state-of-the-art performance on various multimodal tasks with significantly fewer trainable parameters than its predecessors [5, 6].

The core of the BLIP-2 architecture is the *Querying Transformer* (also known as Q-Former), a lightweight transformer that serves as the central mechanism for bridging the modality gap. The Q-Former is responsible for extracting a fixed number of relevant visual features from the frozen vision encoder. It employs a set of learnable query tokens that interact with the visual features through cross-attention and with the text through self-attention (Figure 2.15). This design ensures that the model can compress high-dimensional visual information into a compact set of "visual tokens" that the language model can effectively interpret [5, 6].

The training process is uniquely structured into two distinct stages: 1) vision-language representation learning and 2) vision-to-language generative learning. In the first stage, the Q-Former is trained to align visual queries with text descriptions using three objectives: image-text matching, image-text contrastive learning, and image-grounded text generation. In the second stage, the output of the Q-Former is connected to a frozen LLM via a linear projection. This stage forces the Q-Former to extract visual information that is most useful for the LLM to generate the target text (Figure 2.16) - effectively teaching the language model to "see" without updating its own weights [5, 6].

Despite its efficiency and performance, BLIP-2 shares several inherent limitations with other large-scale multimodal models. While the use of frozen components reduces training costs, the Q-Former acts as a significant information bottleneck: by condensing complex images into a

limited number of tokens, the model may overlook fine-grained visual details or small objects essential for precise spatial reasoning. Additionally, similarly to Flamingo, BLIP-2 is prone to hallucinations and can be highly sensitive to the phrasing of the input prompt, often requiring specific instruction tuning to follow complex user commands accurately [5, 6].

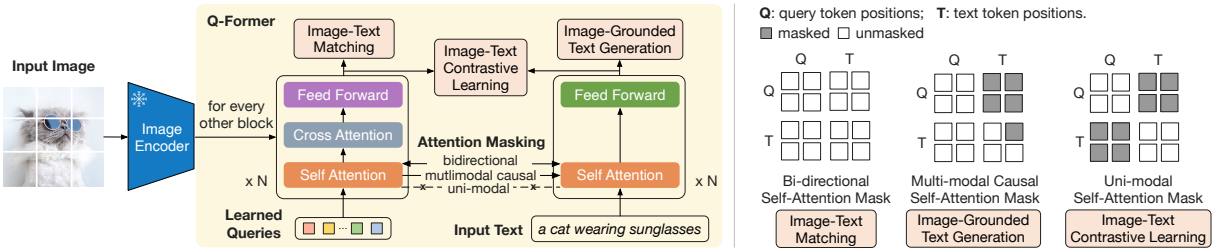


Figure 2.15: BLIP-2 architecture and masking behavior for different loss functions [5].

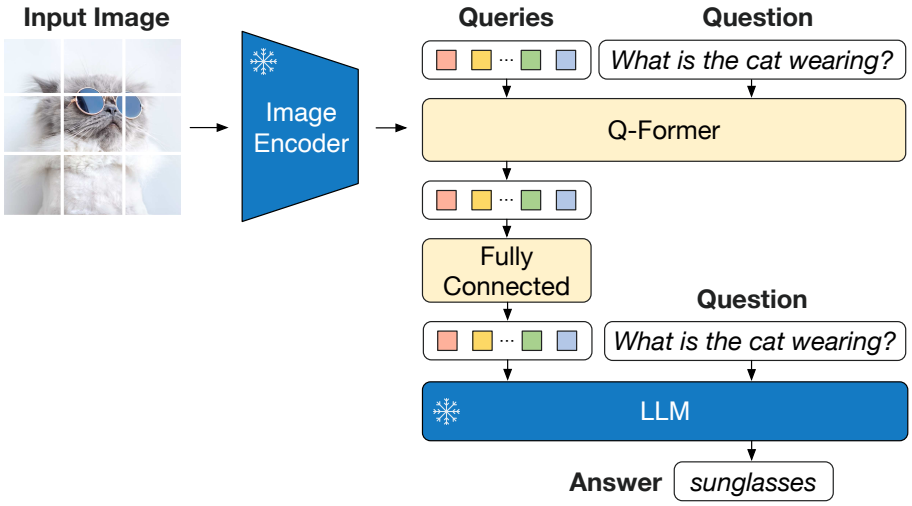
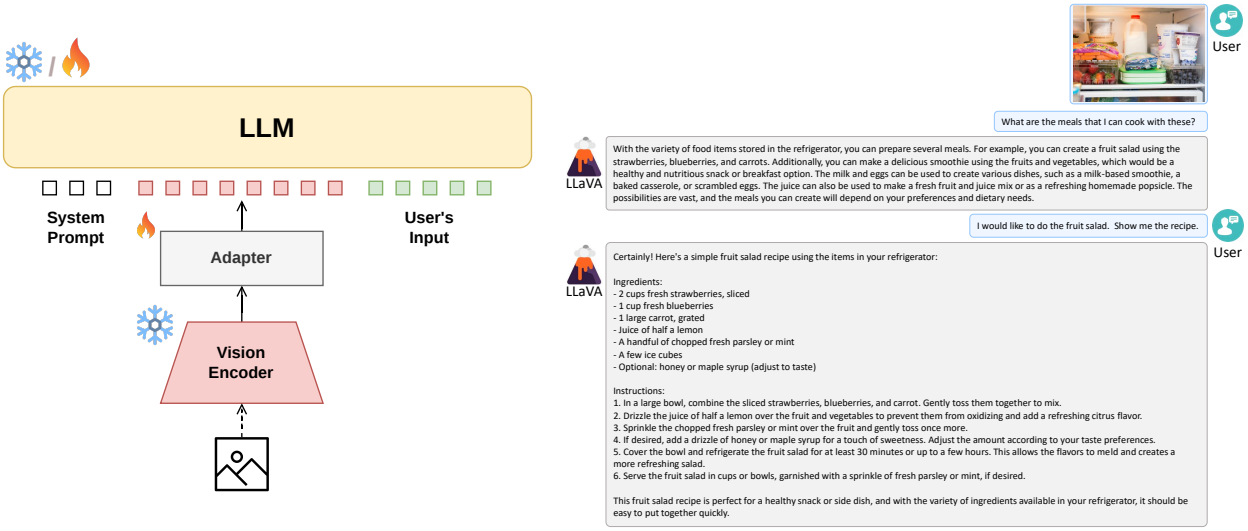


Figure 2.16: BLIP-2 pipeline during visual question answering [5].

2.6.3 LLaVA

LLaVA (Large Language and Vision Assistant) represents a seminal approach in the development of general-purpose visual assistants, pivoting toward the use of visual instruction tuning to align language models with visual inputs. Unlike previous models that relied on complex bridging architectures, LLaVA adopts a minimalist design philosophy (Figure 2.17a), effectively connecting a pre-trained vision encoder with a LLM through a simple *adapter* module - one linear layer in the original paper or two linear layers with a non-linear activation between in newer proposals [48]. This framework treats images as "foreign languages" by converting visual features into tokens that reside in the same word embedding space as the text, allowing the LLM

to process multimodal inputs in a unified manner. This is possible because the vision encoder - typically a CLIP model - is already trained to produce semantically rich features that are closely aligned with natural language concepts, making a simple linear projection sufficient to bridge the modality gap without the need for complex cross-attention mechanisms [3].



(a) LLaVA architecture [3].

(b) LLaVA conversation example [3].

The original LLaVA architecture integrates the vision tower from CLIP and Vicuna as the LLM. The training follows a two-stage paradigm: 1) a feature alignment stage uses image-text pairs to train only the adapter while keeping both the vision and language components frozen; 2) an end-to-end fine-tuning stage updates both the adapter and the LLM on a diverse set of multimodal instruction-following data, while vision encoder still remains frozen [3]. VQA examples are synthesized using GPT-4 and enable the model to engage in complex conversations and reasoning tasks grounded in visual context (Figure 2.17b).

LLaVA 1.5

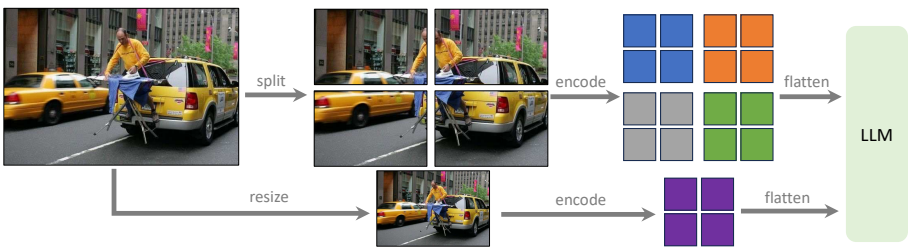


Figure 2.18: Spatial tiling strategy for LLaVA-1.5 architecture [48].

Building on the success of the initial framework, LLaVA-v1.5 introduces a series of targeted architectural and data-driven enhancements that significantly improved the original multimodal performance.

The primary architectural evolution lies in the transition from a single linear projection layer to a MLP. Specifically, LLaVA-v1.5 utilizes a two-layer MLP with GELU activation, a modification that provides a more expressive non-linear mapping between the visual features and the language model's embedding space [48].

Secondly, input image resolution was increased from 224×224 to 336×336 pixels, allowing the model to capture finer visual details and improving performance on tasks that require spatial reasoning [48].

Additionally, to address the limitations of fixed-resolution encoders, experimental configurations such as LLaVA-1.5-HD began employing a *spatial tiling strategy* (Figure 2.18) - splitting images into a 2×2 grid - to preserve high-frequency details (e.g., small text) that would otherwise be lost during downsampling.

Finally, the instruction-tuning set was expanded and diversified, including academic-oriented data [48].

LLaVA 1.6

LLaVA-1.6, also known as LLaVA-NeXT, formalizes the tiling approach through the *AnyRes* (Any Resolution) strategy. Unlike previous versions limited to a single scale, AnyRes dynamically adapts to various aspect ratios by sub-dividing a high-resolution input into multiple patches (e.g., up to 672×672 or higher). Each patch is encoded independently at the native 336×336 resolution, while a downsampled "global" image is provided to maintain overall spatial context [49].

Despite the performance gains achieved through higher resolutions, the strategy of concatenating visual tokens from multiple image patches introduces significant computational and architectural bottlenecks. The primary issue is visual token inflation: as the image is decomposed into finer grids (e.g., 2×2 or 3×3), the number of tokens grows linearly with the number of tiles. In a typical LLaVA-NeXT configuration using a 2×2 AnyRes grid plus a global context image, a single input can generate approximately 2880 tokens (576×5).

This massive sequence length triggers a quadratic increase in self-attention complexity, leading to substantially higher inference latency and VRAM consumption. Furthermore, this

"high-density" visual input consumes a large portion of the LLM's context window, effectively limiting the model's capacity for long-form text generation or multi-turn dialogues involving multiple images. Beyond resource constraints, there is also a risk of signal-to-noise ratio degradation; providing such an expansive set of tokens can sometimes overwhelm the language backbone, making it harder for the model to attend to the most salient visual features amidst redundant background information. In particular, performance analysis across a wide range of LLMs revealed that a strong degradation in performance occurs when the input length (i.e., the number of tokens) reaches approximately half of the context window size [50, 51, 52].

2.6.4 Gemma-4

Developed by Google DeepMind as a downstream open-weight extension of the proprietary Gemini technology, the Gemma family introduced native vision integration across its entire model lineup with its fourth generation, Gemma-4. Designed to optimize the density of intelligence for consumer-grade and edge execution environments, Gemma-4 models process native visual inputs without forcing rigid aspect ratio conversions. It supports highly configurable image processing parameters that natively reconcile context window efficiency with visual fidelity [53].

Given the structural integration of vision inside Gemma-4, three central components built around patch extraction, spatial embedding and down-sampling are proposed:

- **Aspect Ratio Preservation and Variable Resolution**: Unlike classic vision-language models that distort images into fixed uniform dimensions, Gemma-4 resizes input images dynamically. It enforces two baseline conditions: the resolution must fit a user-defined overall token capacity, and both width and height must be strictly divisible by 48 pixels to guarantee clean segmentation without edge distortion (Figure 2.19).

- **Dual Positional Encodings**: Spatial topography is embedded explicitly at the base pixel-patch level. Gemma-4 tracks spatial relationships by pairing learned 2D positional embeddings with a multi-dimensional RoPE system to ensure visual structures map cleanly to the sequential context space.

- **Configurable Image Token Budgets**: Visual granularity can be scaled dynamically on a per-image basis using pre-defined budgets (ranging across 70, 140, 280, 560, or 1120 tokens). This variable budgeting equips deployment pipelines with a strict computational

knob to switch between coarse spatial sketches for basic classification and high-density token profiles for Optical Character Recognition.

Vision Soft Token Pooling

A fundamental innovation within Gemma-4's visual pipeline is the transition away from static token parsing toward a variable-length *Soft Token Pooling* paradigm. Traditional setups project distinct non-overlapping 16×16 pixel matrices directly into separate discrete visual words, producing massive context lengths that quickly exhaust the attention window of edge networks.

Gemma-4 natively structures images around a larger 48×48 soft-token framework to dramatically compress active context footprints [53] exploiting the following operations:

1. **Patch Extraction**: An arbitrary input canvas is divided into tightly arrayed base image patches.

2. **Structural Grouping**: Continuous matrices of individual patch embeddings are logically mapped into spatial coordinate groups. For instance, a regular cluster might bundle a 9×9 layout containing 81 raw image tokens.

3. **Average Pooling Transformation**: Rather than streaming all raw tokens into the Transformer layers, the model executes a spatial average pooling transformation across the vector embeddings inside each individual coordinate group. Given a coordinate group G containing N raw spatial patch embeddings $\mathbf{x}_i \in \mathbb{R}^d$, the generalized soft token representation $\mathbf{s}$ is calculated as:

$$\mathbf{s} = \frac{1}{N} \sum_{i \in G} \mathbf{x}_i$$

The localized pooling step averages group boundaries into highly informative localized vectors, collapsing an extensive token string into a concentrated array of *soft tokens* (Figure 2.20). By filtering micro-redundancies out of the pixel data before it arrives at the main language decoder, this detailed pooling protocol ensures the model retains essential semantic and layout hierarchies while minimizing downstream compute constraints [53].

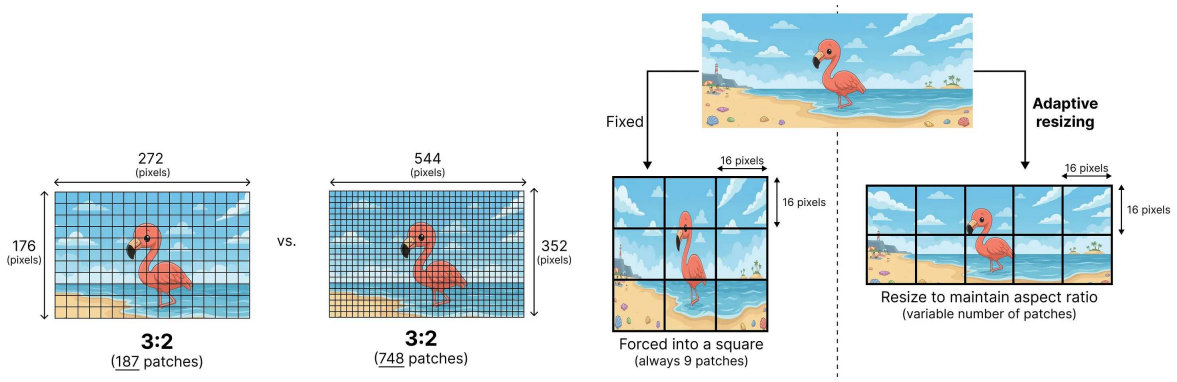


Figure 2.19: Gemma-4's vision preprocessing strategies [53].

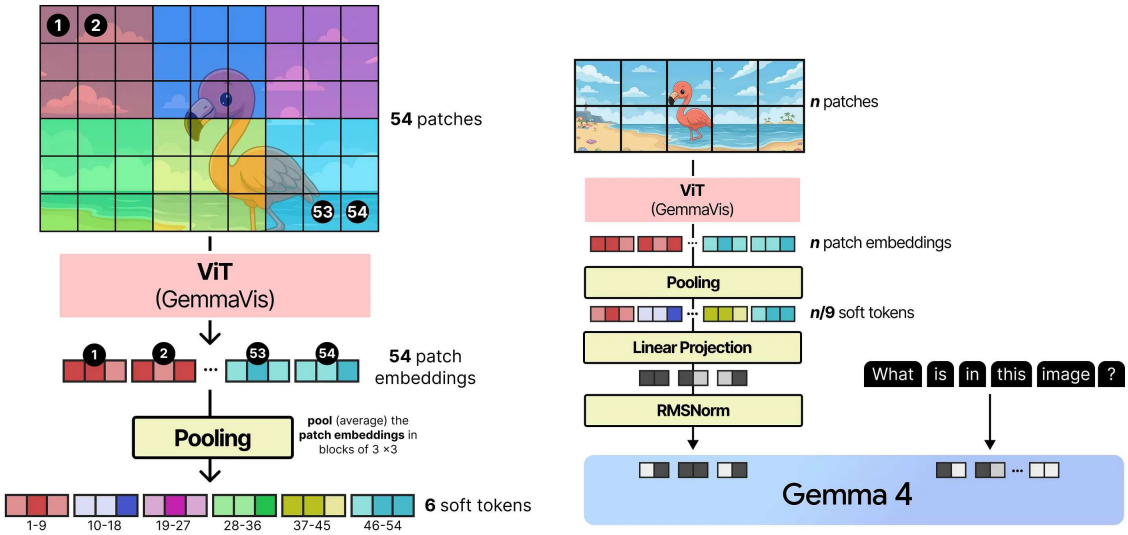


Figure 2.20: Gemma-4's Soft Token Pooling [53].

2.7 Token Pruning

As already mentioned in previous sections, in the context of Vision-Language Models (VLMs), the inclusion of visual tokens - while essential for multimodal understanding - introduces significant computational and memory overhead, particularly as image resolutions increase or when processing long video sequences. For instance, as described in section 2.6.3, a single high-resolution image in models like LLaVA can generate thousands of visual tokens, often occupying more than half of the total context length. However, visual information is inherently more sparse than natural language, containing a high degree of redundancy that does not always contribute to the model's linguistic reasoning.

Beyond technical implementation, visual token pruning strategies can be fundamentally categorized based on their requirement for optimization: training-free and training-based approaches.

Training-free approaches generally operate by identifying and *removing* redundant tokens during the inference phase without altering the underlying model parameters. By leveraging the pre-trained alignment and attention mechanisms already present in the VLM, these methods offer a more computationally efficient and modular solution, allowing for immediate deployment on off-the-shelf models without the overhead of specialized training cycles.

Nevertheless, considering a perfect model which is *not disturbed by duplicated or redundant information*, actively removing input tokens intrinsically reduces the amount of information available to the model and this implies a mathematical limit in the upper bound of the model's performance:

$$\text{Performance using fewer tokens} \leq \text{Performance using all tokens}$$

In contrast, training-based methods typically introduce learnable modules or require a full end-to-end fine-tuning stage to teach the model how to interpret compressed or pruned visual inputs. While these methods can achieve high levels of compression by tailoring the model's perception to a reduced sequence length, they entail computational costs and necessitate access to extensive multimodal instruction datasets.

2.7.1 SparseVLM

SparseVLM is a training-free and text-guided approach to visual token sparsification that eliminates the need for additional parameters or fine-tuning costs. Unlike previous text-agnostic methods that prune tokens based solely on visual importance, SparseVLM argues that the significance of a visual token is fundamentally dependent on the specific question or prompt provided. By leveraging the existing self-attention mechanism within the LLM decoder, the framework adaptively selects relevant visual patches that align with the textual context [54]. Visually illustrated in Figure 2.21, this means that the model can dynamically focus on different regions of an image depending on the user's query, allowing for a more efficient and contextually relevant processing of visual information.

The methodology of SparseVLM is structured around three primary technical components:

- **Relevant Text Token Selection**: The framework identifies specific "rater" tokens within the input prompt that are strongly correlated with visual signals through cross-attention. This step ensures that insignificant words, such as prepositions or pronouns, do not inaccurately influence the rating of visual tokens.

- **Visual Significance Estimation and Adaptive Pruning**: SparseVLM reuses the self-attention logits from the transformer layers to quantify how relevant each visual token is to the selected text raters. To handle varying levels of information density across different inputs, it employs a rank-based strategy where the rank of the attention matrix P indicates the redundancy level, allowing the model to adaptively determine the sparsification ratio for each decoder layer.

- **Visual Token Recycling**: Instead of directly discarding pruned tokens, SparseVLM introduces a recycling mechanism to minimize information loss. Pruned tokens with relatively high significance are grouped using a k-nearest neighbor density peak aggregation algorithm and reconstructed into a more compact representation via an element-wise sum operation.

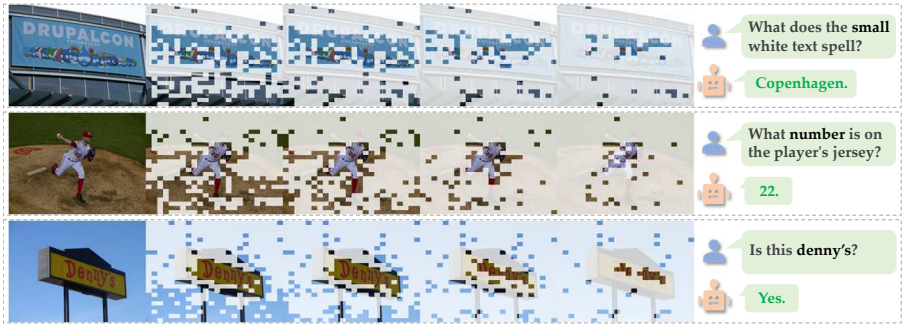


Figure 2.21: SparseVLM conversation example [54].

Experimental results are obtained considering the Vanilla LLaVA model with 576 visual tokens (of CLIP penultimate layer) as baseline, the SparseVLM approach was tested in three different configurations: 64, 128 and 192 visual tokens. When retaining 192 tokens, SparseLLaVA exhibits only a marginal 0.9% decrease in average accuracy while outperforming alternative methods such as ToMe by 10.2%. Instead, in more aggressive sparsification scenarios, such as the 64-token configuration, thanks to the text-aware selection strategy, SparseVLM surpassed the state-of-the-art FastV method by a substantial 17.3% margin [54]. However, despite its performance advantages, SparseVLM obtains only 89.3% with respect to the baseline model

when 64 (approximately 11%) tokens are retained. Ideally, we would obtain a much higher performance or reach the same performance of the baseline model with a much smaller number of tokens, but unfortunately, this is not shown in this work.

2.7.2 PyramidDrop

PyramidDrop is a progressive visual redundancy reduction strategy designed to optimize both the training and inference efficiency of large VLMs. The method is grounded in an empirical observation regarding the layer-wise nature of visual understanding in transformers: while shallow layers require a global, dense representation of the image to establish initial context, deeper layers exhibit a "concentrated distribution" where the model progressively focuses on localized regions relevant to the instruction. Unlike previous methods that drop tokens prematurely in the very first layers or compress them before they enter the LLM, PyramidDrop adopts a "pyramid" structure that retains all visual information initially and gradually filters out redundant tokens as the sequence depth increases [55].

The architecture of PyramidDrop is defined by three core technical features:

- **Multi-Stage Progressive Dropping**: The transformer layers of the LLM are partitioned into S stages (typically $S = 4$ for a 32-layer model). Rather than a single pruning event, tokens are removed at the end of each stage based on a predefined ratio λ. This ensures that the sequence length decreases exponentially, significantly reducing the computational burden in the deeper, more redundant layers of the network.

- **Lightweight Attention-Based Ranking**: To maintain high efficiency, token importance is determined by calculating the similarity between all remaining image tokens and the hidden state of the last instruction token (t_j^I). By reusing the existing query and key matrices from the transformer's self-attention block, the strategy incurs a linear computational overhead while ensuring that the selected tokens are semantically aligned with the user's prompt.

- **Training and Inference Consistency**: PyramidDrop is unique in its ability to serve as both a plug-and-play inference accelerator and an end-to-end training strategy. When applied during training, it encourages the model to condense essential visual information into a smaller set of highly informative representations, which actually improves performance

in high-resolution scenarios by reducing the "confusion" caused by excessive redundant data.

Experimental evaluations demonstrate that PyramidDrop achieves a superior balance between speed and accuracy compared to its predecessors. Considering an average amount of 270 tokens, when applied to LLaVA-NeXT with Vicuna 7B, due to its massive input resolution, it facilitates a 40.4% reduction in training time and a 55% acceleration in inference FLOPs while maintaining or even slightly exceeding vanilla performance across 16 multimodal benchmarks. Instead, when applied to the standard LLaVA-1.5 with Vicuna 7B model, PyramidDrop training time reduction drops to 24.0% with respect to the baseline. Moreover, based on a subset of 5 benchmarks with respect to full set used by SparseVLM, PyramidDrop outperforms it in 64, 128 and 192 token configurations by 1.7%, 2.6% and 1.3% respectively [55]. Even in this case, the performance gain is not so significant and the method does not reach the same performance of the baseline model when 64 tokens are retained, which is the most interesting configuration in terms of computational efficiency.

2.7.3 VisionZip

VisionZip is a text-agnostic visual token reduction framework designed to address the significant computational and memory overhead introduced by excessively long visual sequences. Even in this case, the base hypothesis is that visual encoders such as CLIP and SigLIP used in high-resolution models like LLaVA-NeXT (Section 2.6.3) exhibit extreme information sparsity and a large portion of the generated visual tokens contribute minimal value to the model's understanding. Specifically, attention patterns in these encoders typically concentrate on a limited subset of tokens, while the majority contribute minimal informational value. Unlike previous strategies that rely on the LLM to identify text-relevant patches, VisionZip performs redundancy reduction within the vision encoder itself, ensuring compatibility with all downstream LLM optimization algorithms and real-world scenarios such as multi-turn dialogues [56].

The framework evaluates token importance by examining internal attention scores from the vision encoder's penultimate layer. For encoders with a `[CLS]` token, such as CLIP, VisionZip identifies "dominant tokens" that receive the highest attention from the `[CLS]` head. For non-`[CLS]` encoders like SigLIP, significance is determined by calculating the average attention that each token receives from all others in the sequence [56].

To avoid the loss of fine-grained or peripheral details that may be omitted during selection, the remaining non-dominant tokens are merged rather than discarded. By identifying key-state (K) similarity between remaining patches, VisionZip aggregates redundant information into a compact set of "contextual tokens" through an average merging operation, thereby preserving background semantics at a significantly reduced token count [56].

VisionZip was also evaluated against the standard LLaVA-1.5 with Vicuna 7B model (576 visual tokens) in three configurations (64, 128 and 192 visual tokens) considering a larger set of benchmarks with respect to SparseVLM. Results show that VisionZip outperforms FastV and SparseVLM overall in all configurations [56] and that in some particular cases when the number of retained tokens is 192, it is able to reach performance comparable to the baseline model.

2.7.4 SEPatch3D

SEPatch3D is a dynamic token compression framework designed to accelerate Vision Transformer architectures in the context of 3D object detection and spatial reasoning - capabilities that are increasingly integral to VLM agents navigating the physical world. The method addresses the inherent inefficiency of processing dense, uniform visual tokens in multi-view environments where high-resolution details are only necessary for localized subjects. By revisiting traditional compression techniques such as pruning and merging (Figure 2.22a and Figure 2.22b), SEPatch3D introduces a hierarchical approach that preserves fine-grained semantics while significantly reducing the token count through adaptive patch-size management [57] (Figure 2.22c).

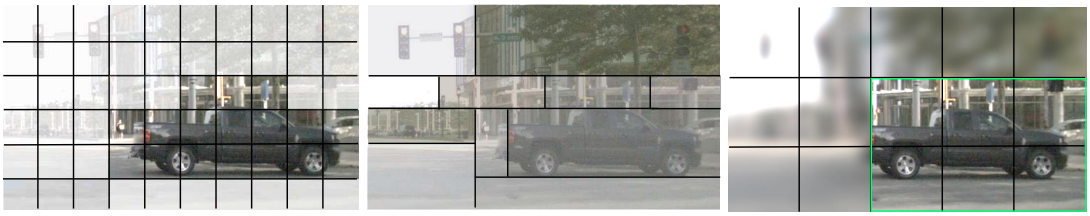
SEPatch3D introduces three core technical innovations:

- **Spatiotemporal-aware Patch Size Selection (SPSS)**: This module dynamically assigns varying patch sizes based on the content of the scene. Smaller patches are assigned to regions containing nearby or complex objects to preserve critical geometric details, while larger patches are utilized for background-dominated areas to minimize computational overhead.

- **Informative Patch Selection (IPS)**: To prevent the loss of semantic cues in coarser regions, IPS identifies the most informative patches across the sequence for dedicated feature refinement. This ensures that the model maintains a high level of perceptual accuracy even when the overall sequence length is significantly reduced.

- **Cross-Granularity Feature Enhancement (CGFE)**: This component bridges the gap between different patch sizes by injecting fine-grained details back into the selected coarse tokens. By leveraging multi-granularity representations, CGFE enriches the semantic features of the pruned sequence, mitigating the common performance decline associated with heavy token processing.

In other words, this approach tries to reduce the number of considered patches similarly to classical pruning methods, preserving full information in relevant image regions in order to mitigate the performance drop.



(a) Token pruning [57]. (b) Token merging [57]. (c) SEPatch3D [57].

2.8 Object-Centric Vision Token Pruning

Object-Centric Vision Token Pruning (OC-VTP) is a training-once approach that - unlike previous methods that rely on handcrafted attention metrics or text-guided heuristics - is designed to select the most representative tokens by minimizing the information loss relative to the original unpruned set.

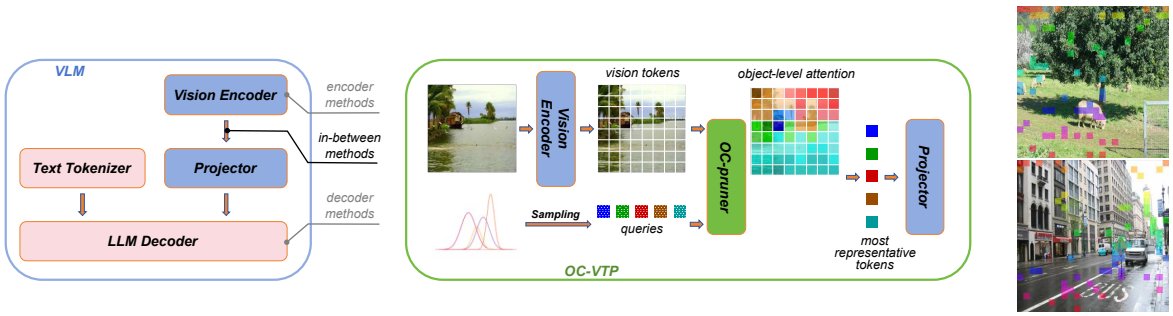


Figure 2.23: OC-VTP proposed architecture [11].

By leveraging Object-Centric Learning principles discussed in Section section 2.5, the framework ensures that the selected tokens represent distinct objects or object parts, maintaining a high degree of perceptual completeness in the compressed visual representation [11].

The architecture of OC-VTP follows the idea of inserting a lightweight OC-pruner module between the vision encoder and the projector. In this case, the pruner utilizes a Slot Attention module to aggregate vision tokens into a predefined number of slots, clustering redundant visual information into a compact set of informative object-level features [11].

Rather than utilizing the aggregated slots themselves as input for the LLM, OC-VTP uses the resulting attention maps to *identify* the most representative vision token for each slot - authors proposed to consider an inner layer given that empirical results show that they are more effective than last ones. By selecting the token at the `argmax` index of the attention map - empirically they proposed the 9th layer of the vision encoder - the method preserves the original feature distribution of the vision encoder, ensuring seamless integration with the frozen LLM decoder without further fine-tuning (Figure 2.23). In order to achieve this, the OC-pruner is pre-trained on a random subset of 40,000 images from the COCO dataset [11].

To ensure that small but critical objects are not discarded, the OC-pruner is trained to minimize a novel *Area-Weighted Mean-Squared Error*. This loss function reciprocally weights reconstruction errors based on the area of each object's mask, forcing the model to retain fine-grained visual details that would otherwise be overshadowed by larger background regions [11].

Following past works, experiments was conducted on the standard LLaVA-1.5 with Vicuna 7B model (576 visual tokens) considering 64, 128 and 192 visual tokens configurations. Despite novel OCL approach, results show that OC-VTP is capable of reaching comparable performance to the canonical state-of-the-art token pruning methods (such as SparseVLM and PyramidDrop) and to outperform them specifically when a smaller number of visual tokens is considered (e.g., 64). Instead when compared in a LLaVA-NeXT with Vicuna 7B setup, results show that OC-VTP is able to outperform the baseline model using only 22.2% of the original visual tokens (i.e., 640 instead of 2880). This counterintuitive result is attributed to the fact that OC-VTP's object-centric selection strategy effectively filters out redundant background information, thereby enhancing the signal-to-noise ratio and allowing the model to focus on the most semantically relevant features, which is particularly beneficial in high-resolution scenarios where token inflation can overwhelm the LLM's attention mechanism [11]. However, it is important to take this result with caution since no extensive experiments were conducted and the performance gain could be attributed to a specific configuration of the model, a lucky initialization or to a specific subset of benchmarks.

2.9 Compact Object-centric Representations

CORE (Compact Object-centric REpresentations) is an object-centric visual token compression paradigm designed to mitigate the limitations of token-centric methods that often lack high-level semantic understanding. CORE adopts a *"one object, one token"* approach, consolidating each distinct semantic entity into a single, compact representation. This paradigm shift ensures that the model preserves complete scene context while eliminating intra-object redundancy, providing a more robust foundation for complex multimodal tasks [58].

The architecture of CORE utilizes an end-to-end framework built upon a shared hierarchical visual encoder (*ConvNeXt-L*) to maintain efficiency in multi-scale and an internal segmentation head which employs learnable queries to generate object masks that are subsequently filtered via a pixel-wise competitive strategy, identifying the unique query with the highest probability for each pixel location. Guided by these filtered soft or hard masks, visual features from the encoder's semantically-rich final layer are merged on an object-by-object basis [58].

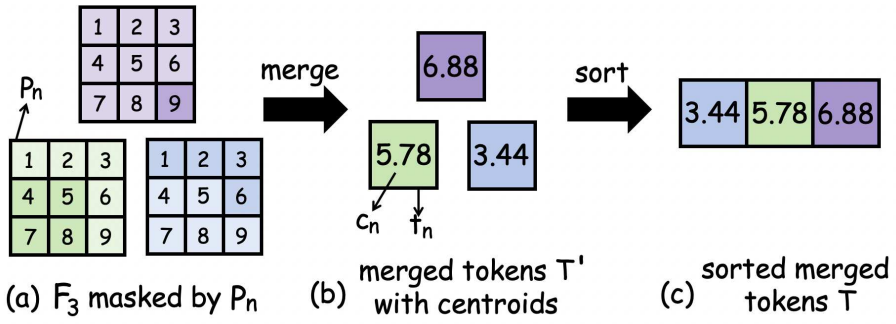


Figure 2.24: Centroid-guided sorting mechanism proposed in CORE [58].

Moreover, to restore the spatial coherence required for accurate linguistic reasoning, CORE introduces a novel *centroid-guided sorting mechanism* (Figure 2.24). The framework calculates the centroid of each segmentation mask through a weighted average of pixel coordinates and sorts the merged tokens in ascending order accordingly. This mechanism is critical for maintaining positional information, as demonstrated by the model's superior performance in detecting objects' positional relationships, such as correctly identifying a character riding a bird rather than the bird being on the character's shoulder. Training follows a two-stage strategy involving feature alignment pre-training and visual instruction tuning, utilizing LoRA adapters for parameter-efficient fine-tuning of the language decoder [58].

2.10 Slot-MLLM

Slot-MLLM represents a shift toward a deep use of object-centric learning in multimodal large language models. Rather than discarding visual tokens like in the majority of previous methods, instead it introduces the *Slot Q-Former*, a visual tokenizer designed to decompose images into a fixed sequence of discrete, semantically distinct tokens. By integrating Slot Attention into a unified next-token prediction framework, the model achieves a "bilingual" capability, effectively processing and generating visual content as a structured sequence of object-oriented primitives [59].

Slot Q-Former replaces the standard cross-attention layers of the Q-Former with a competitive Slot Attention mechanism. Moreover, to bridge the gap between continuous visual embeddings and the discrete nature of LLM vocabularies, Slot-MLLM employs a Vector Quantizer based on Residual Vector Quantization (RVQ). This module discretizes the continuous slot embeddings into a sequence of tokens from a shared codebook of size 8,192. This discretization allows the LLM to treat visual information identically to text, facilitating seamless autoregressive modeling across multimodal tasks [59].

The training procedure is bifurcated into two primary stages to optimize for both reconstruction fidelity and cross-modal semantic alignment:

- **Continuous Slot Embedding Training** (Stage 1): The Slot Q-Former is trained alongside a diffusion-based visual decoder. The model optimizes a composite loss function $\mathcal{L}_1$ comprising image reconstruction (diffusion and CLIP-MSE losses) and image-text alignment. Specifically, an image-text contrastive loss and an image-grounded text generation loss are utilized to ensure that the slot embeddings are semantically compatible with textual tokens.

- **Discrete Slot Token Training** (Stage 2): In this stage, the Slot Q-Former and Visual Decoder are frozen. The RVQ and an additional Transformer block are trained to reconstruct the original continuous embeddings, guided by commitment and reconstruction losses.

Despite the innovative architecture, Slot-MLLM faces significant limitations. Firstly, the multi-stage training pipeline is more computationally demanding than "plug-and-play" pruning methods or simpler projection-based MLLMs. Secondly, the discretization process, while enabling seamless integration with the LLM, introduces quantization errors that can degrade

performance, particularly in tasks requiring fine-grained visual understanding. In fact, results show that Slot-MLLM does not reach standard approach performance, but only validate a novel architectural direction, serving as a promising foundation for future optimization.

2.11 LLaVA-Mini

LLaVA-Mini is part of the efficient MLLMs family, proposing a novel approach to reduce the number of vision tokens.

The conceptual foundation of LLaVA-Mini stems from a layer-wise analysis of how LMMs process visual information. Empirical observations indicate that vision tokens are most critical in the early layers of the LLM, where text tokens actively seek and fuse visual data through attention mechanisms. In deeper layers, the importance of these tokens diminishes sharply as the model shifts focus toward instruction tokens that have already internalized the relevant visual context. Consequently, LLaVA-Mini introduces a *modality pre-fusion* module to execute this integration process before the tokens enter the LLM [60].

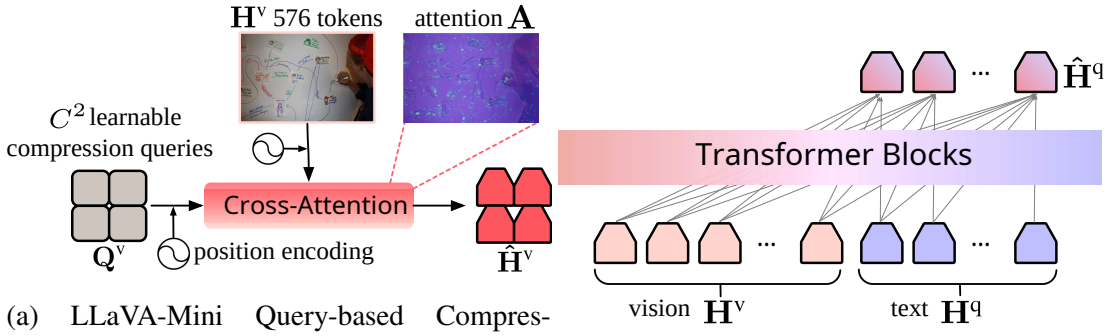
The architecture of LLaVA-Mini introduces two key components to achieve efficient visual token compression while preserving performance:

- **Query-based Compression** (Figure 2.25a): To reduce the input sequence length, the framework employs learnable queries that interact with the original vision tokens via cross-attention. This mechanism adaptively extracts salient visual features into a compact set of tokens, preserving spatial relationships through (2D) sinusoidal positional encodings.

- **Modality Pre-fusion** (Figure 2.25b): To compensate for potential information loss during compression, LLaVA-Mini utilizes N Transformer blocks (sharing hyperparameters with the LLM) to allow text tokens to fuse information from the full set of original vision tokens in advance. The resulting fusion tokens are then extracted and passed to the LLM alongside the compressed vision tokens.

Similarly to standard LLaVA, training phase is divided into two stages; however the novel components are not trained in both stages, but only in the second one. In particular, the first stage is exactly the same as the standard LLaVA's training procedure, where *all* the visual features are projected and fed to the LLM - in other words, this stage is used to pre-train only the little MLP adapters, aligning the visual features with the LLM's language space. In the second stage,

instead, the novel components are trained (from scratch) together with the LLM - while the vision encoder remains frozen. Unlike the standard LLaVA's second stage training procedure, in this case the batch size and the number of epochs are doubled - 256 global batch size and 2 epochs - in order to allow the model to learn how to fuse the visual features and to reach a good performance even with a smaller number of vision tokens. This means that LLaVA-Mini is more computationally expensive to train with respect to the standard LLaVA, needing computational resources to handle a double batch size for twice the time.



(a) LLaVA-Mini Query-based Compression [60].

(b) LLaVA-Mini Modality Pre-fusion [60].

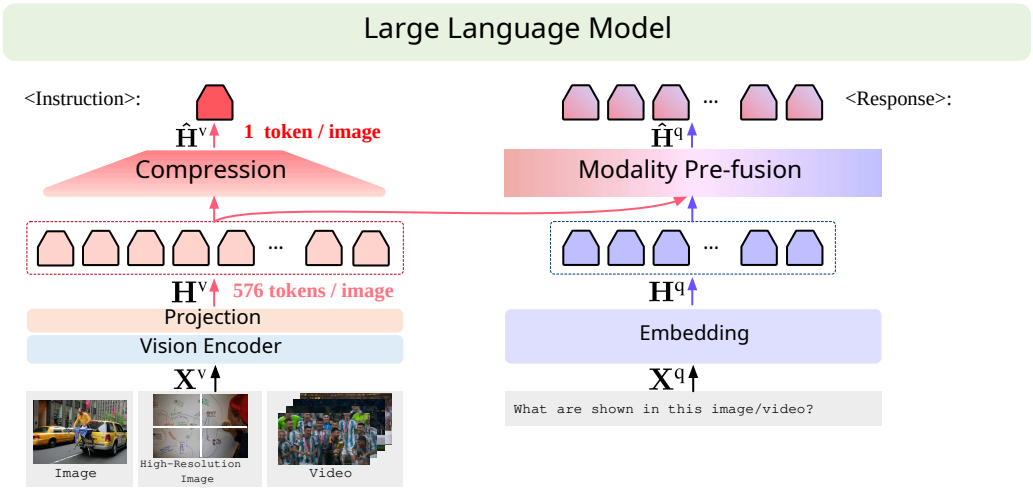


Figure 2.26: Architecture of LLaVA-Mini [60].

During inference, LLaVA-Mini reduces Floating Point Operations (FLOPs) by approximately 77% compared to the LLaVA-v1.5 baseline [60] and experimental results demonstrate that LLaVA-Mini achieves remarkable performance, outperforming the LLaVA-v1.5 baseline in the limited set of proposed benchmarks by 1.6%.

However, as already discussed for other methods, it is important to note that the method was evaluated on a limited set of benchmarks and that the performance gain could be attributed to

a specific configuration of the model. Moreover, even if LLaVA-Mini represents an interesting architectural direction for future research, it is important to note that the method relies on a pre-fusion mechanism that only "move" the problem of having too many vision tokens in the additional layers, instead of solving the problem directly. Additionally, due to its training procedure, it is less flexible and more computationally expensive to train compared to other methods illustrated in this chapter.

3. Methodology

This chapter delineates the technical framework and the experimental design formulated to address the computational problems inherent in Multimodal Large Language Models during the elaboration of multimodal inputs. Specifically, it details the architectural modifications, the aggregation paradigms and the custom training pipelines developed to optimize the visual processing workflow of the standard MLLM architectures.

While the conventional LLaVA architecture (Section 2.6.3) ingests visual tokens extracted by a pre-trained vision encoder in their entirety - leading to a computational bottleneck due to the quadratic scaling of the self-attention mechanism - this research tries to mitigate MLLMs computational issues associated to visual information elaboration, exploiting a dedicated *aggregator module* designed to systematically distill the vast set of visual features provided by vision encoder (such as CLIP, described in Section 2.3.3) into a highly compact, yet semantically rich, set of dense vectors.

This aggregation philosophy shares conceptual lineage with other foundational frameworks that address this bottleneck through distinct cross-attention designs, such as Flamingo's Perceiver Resampler (Section 2.6.1) or the Q-Former (Section 2.6.2) module's learnable queries to extract a fixed number of visual representations. By condensing the visual input rather than discarding information - as occurs in traditional token pruning approaches - this work aims to preserve the foundational semantic integrity required for complex multimodal reasoning, while drastically reducing the quadratic computational cost of the underlying language model's self-attention mechanism.

To achieve a holistic understanding of how best to accomplish this compression, three fundamentally distinct paradigms are explored: 1) *text-agnostic vision-only* aggregation, 2) *text-influenced vision-centric* aggregation and 3) *text-guided vision* aggregation. The first approach operates exclusively on the visual modality, regardless of the accompanying text. Other approaches, instead, leverage the textual modality to help the visual compression process, either by allowing the text to directly influence the visual feature selection or by directly guiding the visual aggregation based on the text features.

Every paradigm is explored through the lens of various aggregator modules, ranging from simple pooling operations to more complex attention-based mechanisms. The final goal is to

plug the aggregator module and its dependencies into the standard LLaVA architecture, replacing the original vision encoder, and to evaluate the performance of the resulting MLLM on a set of common downstream tasks.

3.1 Aggregator

The aggregator is the fundamental component which is responsible for the synthesis of information, bringing them into a compact set of dense concepts. In other words, this module could be considered an information bottleneck which is responsible for the compression, while trying to preserve as much as possible the information of the original input.

Formally, the aggregator can be defined as a differentiable function g that takes as input a set of visual features $X = \{x_1, x_2, ..., x_N\}$ extracted by a *encoder* and produces a set of aggregated features $g(X) = Y = \{y_1, y_2, ..., y_K\}$ where $K << N$. Generally, the aggregator is followed by a *decoder* that takes the aggregated features Y and produces the final output, which can be used for various downstream tasks (Figure 3.1).

In this work, several different aggregator modules are explored, mainly to compress the visual information, even if some of them can also be exploited in other modalities - such as text, video or audio.

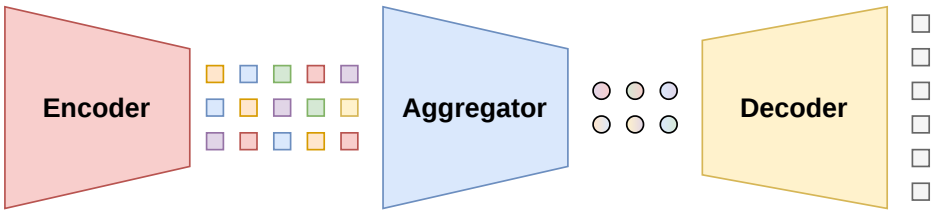
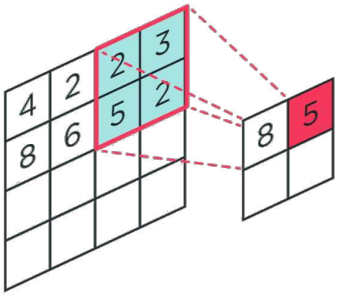


Figure 3.1: General aggregation mechanism in the proposed architecture.

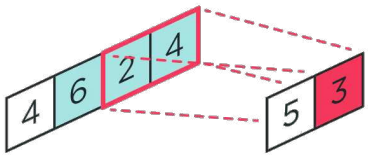
3.1.1 Pooling

Pooling is the most straightforward approach to aggregate information. It simply applies a fixed function - such as mean, max or sum - to the input features, without any learnable parameters, producing a smaller set of features. While this approach is computationally efficient and easy to implement, it has the major drawback of being too simplistic: it might not be able to capture the complex relationships between the input features or naturally divide the same object into different regions.

In particular, when applied to the visual modality, 2D pooling (Figure 3.2a) is used to reduce the spatial dimensions of the feature maps extracted by the vision encoder, producing a compact representation of the visual information, while when applied to other modalities, such as text, 1D pooling (Figure 3.2b) is considered, given that the input features are represented as a monodimensional sequence, instead of a 2D grid.



(a) Example of 2D Max Pooling [61].



(b) Example of 1D Average Pooling [61].

3.1.2 Convolution

In contrast to pooling, convolution introduces learnable parameters to aggregate information, offering a more flexible and powerful alternative. This approach slides a set of learnable filters (also called *kernels*) across the input features, performing element-wise multiplications and summing the results to produce a feature map.

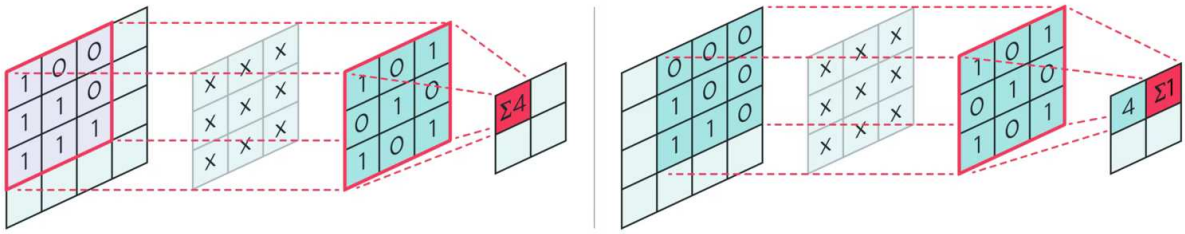


Figure 3.3: Example of 2D Convolution with 1 filter 3×3, stride 1 and no padding [61].

The primary advantage of convolution lies in its ability to automatically learn and extract hierarchical patterns and complex relationships within the data, effectively mitigating the information loss typical of simple pooling operations. Furthermore, thanks to the property of weight sharing - where the same filter is applied across the entire input - convolution remains highly efficient, drastically reducing the number of parameters compared to a fully connected layer.

However, this approach comes with a higher computational cost and requires more data and training time to properly optimize the learnable weights.

Similarly to pooling (Section 3.1.1), when applied to the visual modality, 2D convolution is employed to capture spatial hierarchies, such as edges, textures, and complex shapes from the 2D grid of images. Conversely, when dealing with sequential modalities like text, 1D convolution is utilized - in this case, the filter slides along a single dimension, allowing the model to capture local temporal patterns or n-gram combinations within the one-dimensional sequence of word embeddings.

3.1.3 Cross Attention

Described in Section 2.2.1, Cross Attention is an attention-based mechanism - introduced in the Transformer architecture [9] - that allows a set of query vectors to attend to a different set of key-value pairs.

Proposed in Set Transformer [62] and exploited successfully in various architectures (such as CoCa [63]), Cross Attention (also known in this scenarios as *Attention Pooling*) is a powerful mechanism for aggregating information from a set of input features, reducing the total amount of vectors, allowing at the same time the model to learn complex relationships and dependencies between the inputs.

When employed as an information aggregation mechanism without any other mechanisms, this methodology exhibits a natural permutation invariance property. Unlike sequential architectures, the operation treats input features as a set rather than an ordered sequence, making it exceptionally well-suited for unstructured data domains such as point clouds or set-based learning frameworks. However, the natural lack of order may pose challenges in scenarios where spatial (or temporal) relationships are crucial, as the model must learn to infer these relationships solely from the content of the features, without any inherent positional cues. For this reason, in some cases, positional encodings or other forms of spatial bias (such as a sort of spatial sorting proposed described in Section 2.9) may be necessary to guide the attention mechanism in capturing relevant spatial relationships effectively.

Moreover, when downstream task does not intrinsically lead to a set of different information, model might tend to collapse queries, extracting redundant information with each query. To mitigate this issue, careful design of the query initialization and the attention mechanism is crucial, potentially incorporating techniques such as learnable query embeddings, regularization

strategies to encourage diversity among the queries or starting from orthogonal vectors.

Additionally, while it reduces the token count for downstream operations, the initial cross-attention layer still scales quadratically with respect to the input dimensions, specifically exhibiting a complexity of $O(N \times M)$, where N represents the number of queries and M denotes the size of the input feature set. For this reason, the total amount of queries must be carefully balanced to ensure that the computational benefits of reducing the token count for downstream task are not offset by the overhead introduced by the cross-attention mechanism itself.

Self and Cross Attention

A possible extension of the cross attention aggregation mechanism introduced in Section 3.1.3 is the introduction of a self-attention mechanism among the queries themselves - similarly to the original proposed Transformer architecture [9] - allowing them to interact and share information before attending to the input features. This approach aims to enhance the representational capacity of the queries by enabling them to capture inter-query relationships and dependencies, potentially leading to a more coherent and structured aggregation of the input features.

In other words, instead of treating the queries as independent entities that attend to the input features in isolation, Self-Cross Attention allows the queries to first interact with each other through a self-attention mechanism, capturing any relevant relationships or dependencies among them. This can be particularly beneficial in scenarios where the queries are expected to capture different aspects of the input features, as it encourages a more coordinated and complementary representation among the queries.

However, this interaction introduces additional architectural complexity and increases the computational overhead, as the model must compute cross-query relationships alongside the standard input-to-query attention weights. Additionally, the design of the self-attention mechanism must be carefully balanced to avoid over-smoothing, where excessive communication could lead to homogenized representations of the queries.

3.1.4 Slot Attention

Slot Attention (SA) represents a significant advancement in the unsupervised discovery of objects, becoming the most common starting point architecture for OCL purpose (Section 2.5). Unlike standard Attention mechanisms used in Transformers (Section 2.2) - where attention is typically computed over the spatial dimensions or sequence length - Slot Attention employs a

competitive procedure to bind specific features to a fixed number of learnable slots [10].

In particular, Slot Attention maps a set of input features $X \in \mathbb{R}^{N \times D_{feat}}$, extracted from an image thanks to a vision encoder (e.g., CNN) to a set of K slots $S = \{ s_1, ..., s_K \} \in \mathbb{R}^{K \times D_{slots}}$ [10].

As explained in Algorithm 1, in each step, initial slots are sampled from a learnable Gaussian distribution (mean μ and standard deviation σ are trainable parameters), and at each iteration, the slots compete to explain the input features through a softmax-based attention mechanism [10].

The mechanism is characterized by an iterative refinement process, typically repeated for 3 iterations [10]. In each iteration, the slots act as queries $Q = Linear(S)$, while the input features act as keys $K = Linear(X)$ and values $V = Linear(X)$. The attention coefficients are computed using a Softmax operation. Crucially, in Slot Attention, the softmax operation is applied *over the slots* (the query dimension) rather than the keys. This forces the inputs to compete for assignment to a particular slot. The updates for the slots are computed as a weighted sum of the values V, followed by a Gated Recurrent Unit (GRU) update to maintain stability across iterations [10]. In Figure 3.4 is illustrated an example of Slot Attention iterative refinement process and its applications to unsupervised object discovery and supervised set prediction.

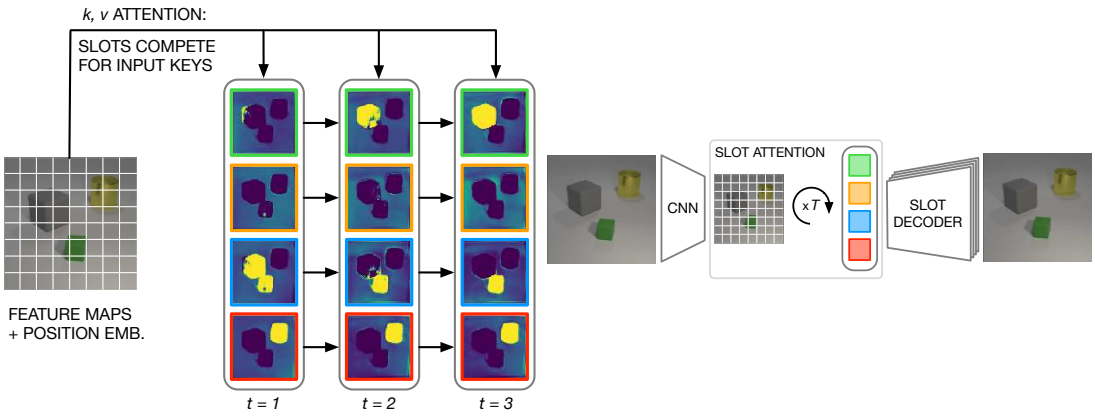


Figure 3.4: Slot Attention module and original proposed pipeline [10].

However, despite its effectiveness in unsupervised object discovery, Slot Attention faces several architectural and practical limitations. First, the number of slots K must be predefined and remains fixed during inference, which restricts the model's flexibility in scenes where the number of objects varies significantly [10]. Furthermore, the competitive softmax mechanism and the reliance on simple reconstruction objectives often struggle to scale to complex, natural images with textured backgrounds; this frequently leads to the fragmentation of single objects across multiple slots or the conflation of background elements with foreground entities [40, 47].

The iterative nature of the refinement process also introduces a computational bottleneck, as the sequential GRU updates and repeated attention calculations increase the overhead compared to standard single-pass Transformer architectures (Section 2.2.3). Finally, Slot Attention is highly sensitive to the initialization of the slot distributions, where poorly sampled initializations can lead to redundant representations or *dead slots* - where only one or a few slots aggregate the majority of the visual information while others collapse to small or background regions, leading to a performance drop since most of the aggregation capacity remains unused - that fail to capture the underlying compositional structure of the scene [64].

Algorithm 1 Slot Attention module. The input is a set of N vectors of dimension D_{inputs} which is mapped to a set of K slots of dimension D_{slots}. Slots are initialized by sampling their initial values as independent samples from a Gaussian distribution with shared, learnable parameters $\mu \in \mathbb{R}^{D_{\text{slots}}}$ and $\sigma \in \mathbb{R}^{D_{\text{slots}}}$. Proposed number of iterations was $T = 3$.

Input: $\texttt{inputs} \in \mathbb{R}^{N \times D_{\text{inputs}}}$, $\texttt{slots} \sim \mathcal{N}(\mu, \text{diag}(\sigma)) \in \mathbb{R}^{K \times D_{\text{slots}}}$
Layer params: k, q, v: linear projections for attention; GRU; MLP; LayerNorm (×3)
$\texttt{inputs} = \texttt{LayerNorm}(\texttt{inputs})$
for $t = 0 \ldots T$ **do**
    $\texttt{slots\_prev} = \texttt{slots}$
    $\texttt{slots} = \texttt{LayerNorm}(\texttt{slots})$
    $\texttt{attn} = \texttt{Softmax}\left(\frac{1}{\sqrt{D}} k(\texttt{inputs}) \cdot q(\texttt{slots})^T, \texttt{axis} = \text{`slots'}\right)$ # norm. over slots
    $\texttt{updates} = \texttt{WeightedMean}(\texttt{weights} = \texttt{attn} + \epsilon, \texttt{values} = v(\texttt{inputs}))$ # aggregate
    $\texttt{slots} = \texttt{GRU}(\texttt{state} = \texttt{slots\_prev}, \texttt{inputs} = \texttt{updates})$ # GRU update (per slot)
    $\texttt{slots} \mathrel{+}= \texttt{MLP}(\texttt{LayerNorm}(\texttt{slots}))$ # optional residual MLP (per slot)
end for
return $\texttt{slots}$

Self-Slot Attention

To address the inherent limitations of the standard Slot Attention mechanism, similarly to approach described in Section 3.1.3, a possible extension is the introduction of a self-attention mechanism among the slots themselves, allowing them to interact and share information before attending to the input features. This approach, which can be referred to as *Self-Slot Attention*, aims to enhance the representational capacity of the slots by enabling them to capture inter-slot relationships and dependencies, potentially leading to a more coherent and structured

aggregation of the input features. However, exactly as in the case of Self-Cross Attention, this interaction introduces additional architectural complexity, increasing the computational overhead, and may lead to an over-smoothing effect on the slot representations.

3.1.5 Perceiver Resampler

Introduced in Flamingo (Section 2.6.1), Perceiver Resampler is a specific type of cross attention mechanism designed to efficiently aggregate information from a large set of input features into a smaller set of output representations, while maintaining the ability to capture complex relationships and dependencies within the data. In this case as well, it operates by using a fixed number of learnable query vectors that attend to the input features through a cross-attention mechanism, similar to the standard cross attention described in Section 3.1.3. However, unlike the standard approach, Perceiver Resampler incorporates a crucial architectural modification: queries attend to both the input features and the queries themselves. This design allows the model to capture not only the relationships between the input features but also the interactions among the queries, enabling a more flexible and powerful aggregation process, generally reducing the risk of over-smoothing caused by Self-Attention.

Nevertheless, because it relies on a set of learnable query vectors and without any explicit mechanism to encourage diversity among them, Perceiver Resampler may still face challenges in scenarios where the queries collapse to similar representations, leading to redundant outputs that fail to capture the full range of information present in the input features - similarly to previous approaches, this issues can be mitigated through careful design of the query initialization or regularization strategies to encourage diversity among the queries. Additionally, due to the nature of the attention mechanism (Section 2.2.1), the output could be a weighted average of the input features, which may lead to a loss of fine-grained details and a potential blurring of important information, especially in scenarios where the input features are highly complex or contain multiple distinct entities.

3.2 Text-agnostic Vision-only Aggregation

The simplest approach to aggregate visual information is to operate exclusively on the visual modality, without any influence from the accompanying text. This approach, which can be referred to as *text-agnostic vision-only* aggregation, focuses solely on distilling the visual features

extracted by the vision encoder into a compact representation, without considering the textual context or any potential interactions between the modalities.

Following the encoder-bottleneck-decoder paradigm described previously, the aggregator module is designed to merely *compress* the visual information - obtained by the selected vision encoder - into a compact set of dense vectors, which are then passed to a decoder which is responsible for producing the final output (Figure 3.5a).

In particular, as suggested by DINOSAUR (Section 2.5.1) results, the aggregator is trained in a self-supervised manner, using the decoder to reconstruct the original visual features from the aggregated representation, optimizing a *reconstruction loss* - such as mean squared error (MSE) - between the reconstructed features and the original features extracted by the vision encoder (instead of the raw images, which would require more computational resources and a more complex decoder).

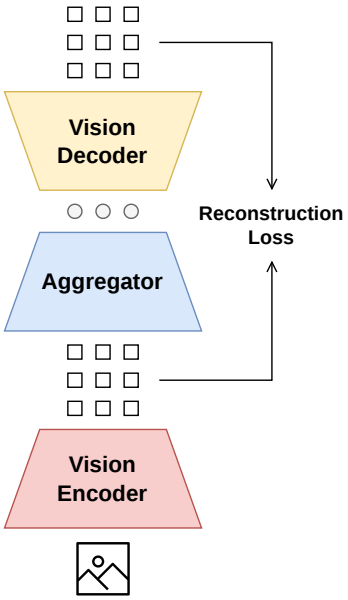
This approach requires a separated training phase to permit the aggregator to learn how to effectively compress the visual information, without any influence from the text, which is not yet available during this phase. Once the aggregator is trained, it can be plugged into the standard LLaVA architecture (Figure 3.5b) - remaining frozen in both training stages and replacing the original vision encoder - and the resulting MLLM can be fine-tuned on downstream tasks using the standard training pipeline. It is important to note that decoder is used only during the training phase of the aggregator, whereas during the fine-tuning phase of the MLLM only the encoder and the bottleneck are kept, while the decoder is discarded. For this reason, the majority of capabilities should be concentrated in the encoder and the bottleneck, while the decoder should be as simple as possible to reduce the computational cost during the training phase of the aggregator and the risk of discarding useful layers.

Due to the need to train the aggregation from scratch in isolation, pre-training phase requires a large number of images that are in-distribution with the downstream tasks are required to allow the aggregator to learn how to effectively compress the visual information - without any influence from the text that is not yet available during this phase. Following the DINOSAUR proposal and given that 1) the final goal is to plug the aggregator into the standard LLaVA architecture - replacing the original vision encoder - and 2) visual instruction tuning dataset was created starting from a superset of MS COCO [3], then it is reasonable to use MS COCO itself as the training dataset for the aggregator, given that it is a large-scale dataset of natural images that contains a wide variety of visual concepts and scenes, which can provide a rich and diverse

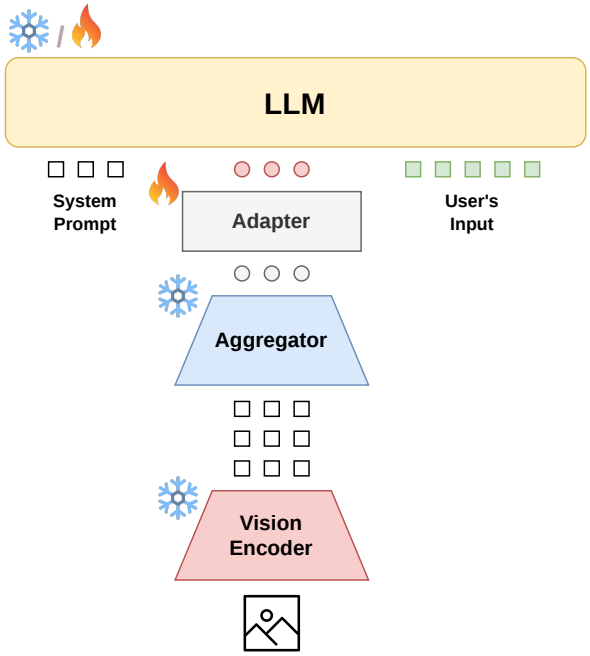
set of visual features for the aggregator to learn from. Moreover, using the same dataset for both the pre-training of the aggregator and the fine-tuning of the MLLM can help to ensure that the visual features learned by the aggregator are well-aligned with the downstream tasks, potentially leading to better performance.

A text-agnostic vision-only aggregator which focuses solely on the visual information - without any influence from the text - allows the aggregator module to be trained *only once* and plug it into the standard LLaVA training pipeline many times, without any need to retrain it when LLM changes (e.g., when a new LLM is released or to have multiple model sizes). However, assuming to need only one MLLM version, the need to have a separate training phase for the aggregator can be considered a major drawback, given that additional computational resources and time is required.

Moreover, as already mentioned, this approach - that operates in isolation from the textual modality - may struggle to capture relevant relationships and information within the visual features needed by LLM to perform well in downstream tasks.



(a) Text-Agnostic Vision-Only Aggregation pre-training pipeline inspired by DINOSAUR [40].



(b) Text-Agnostic Vision-Only Aggregation plugged into LLaVA pipeline [3].

3.2.1 Multi-Encoder

By reducing the number of visual features through an aggregator, the computational cost of the LLM is significantly reduced. With this freed computational budget, it is reasonable to hypothesize that LLMs could achieve performance improvements by using visual features extracted by different vision encoders or by using visual features extracted by different layers of the same vision encoder, given that different layers or models can provide different levels of abstraction of the visual information, so they can be more or less effective for different downstream tasks.

The *multi-encoder* approach is based on the idea of using multiple vision encoders or multiple layers of the same vision encoder to extract visual features, and then aggregate them together using the aggregator module, before passing them to the decoder (Figure 3.6). For example, the aggregated visual information obtained by a DINO-like and a CLIP-like vision encoder can be combined together to have a more complete representation of the visual information, given that DINO-like encoders are able to capture object-level features and CLIP-like encoders are trained to capture language alignment visual features.

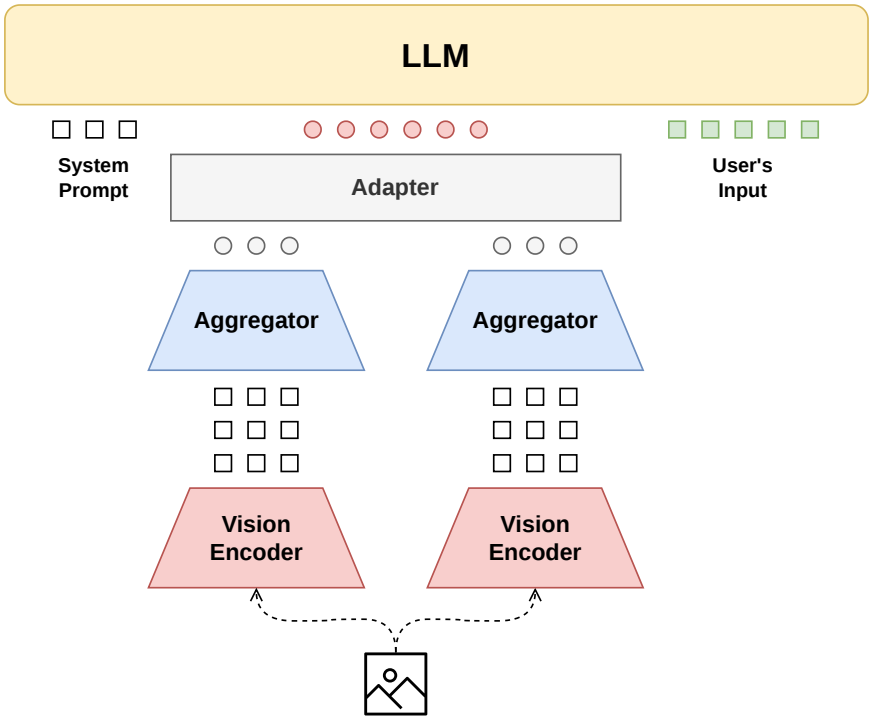


Figure 3.6: Multi-encoder architecture.

3.3 Text-influenced Vision-centric Aggregation

One step forward from the text-agnostic vision-only approach described in Section 3.2 is to allow the text to influence the visual feature selection process, without directly guiding the visual aggregation itself with the text features. This approach, which can be referred to as *text-influenced vision-centric* aggregation, aims to synthesize visual features, in a text-aware scenario, while still maintaining a main focus on the visual information.

In this approach the text actively influences the loss function used to train the aggregator, conditioning the aggregation process to select visual features that are more relevant for the downstream tasks - instead of merely compressing them - without directly conditioning the aggregation process on the text features. This can be achieved by incorporating a text-based loss component that encourages the model to select visual features that are semantically aligned with the textual input, while still allowing the visual aggregation process to operate independently from the text.

The most common approach to have a text-based loss is to rely on a language model decoder to generate text and optimize following a standard language modeling loss (e.g., cross-entropy loss) between the generated text and the ground-truth text, which is available during the training phase of the aggregator. LLMs are powerful models that can be exploited as text decoders in this kind of pipelines and, fortunately, LLaVA training pipeline itself is based on a language modelling loss and already incorporates a vision branch that is partially trained together with the LLM.

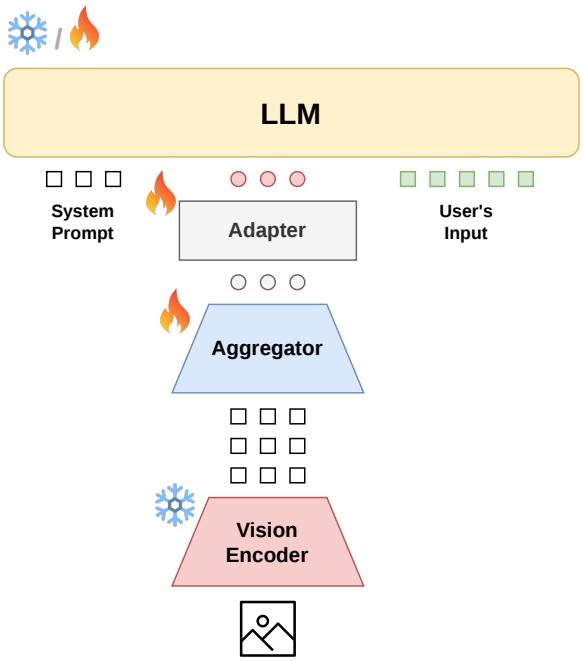
This allows the training pipeline to be simplified, the aggregator can be trained *together* with the LLM in a LLaVA-like training pipeline, without any need for a separate training phase and without any need to design a custom decoder for the aggregator - which is instead replaced by the LLM itself. In other words, the aggregator is trained from scratch during the first and second stage of the LLaVA training pipeline. In this way the aggregator module is trained to select visual features that are more relevant for the MLLM's downstream tasks, while still maintaining a main focus on the visual information. This is a major shift from text-agnostic approach (Section 3.2) where the aggregator is trained in isolation (becoming a simple agnostic visual feature compressor).

Training the aggregator together with the LLM - instead of training it in isolation - results in a more efficient training pipeline that reduces the required computational resources and time by

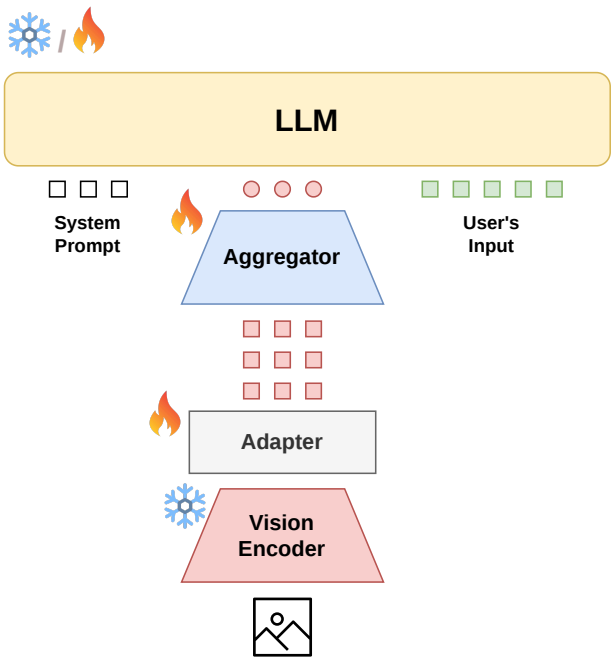
eliminating the need for a separate training phase. However, removing the additional training phase, the aggregator module is trained on a smaller amount of data and this could lead to a suboptimal performance attributable to *underfitting*. Moreover, the aggregator module (trained from scratch) must be plugged in the LLaVA pipeline carefully to prevent that its initial random weights may lead to a *catastrophic forgetting* of the rest of the model, which is instead already pre-trained and has already learned useful features for the downstream tasks.

In original LLaVA architecture, projector - which aligns the visual features to the LLM hidden states - is inserted between the vision encoder and the LLM. When plugging a new module into the pipeline - i.e. the aggregator - multiple choices are available: 1) to insert the aggregator before the projector (Figure 3.7b), allowing it to operate on the original visual features extracted by the vision encoder; 2) to insert the aggregator after the projector (Figure 3.7a), allowing it to operate on the projected visual features which are already aligned to the LLM hidden states; 3) add a new projector in order to insert the aggregator between two adapters (Figure 3.7c), where the first is responsible for projecting the original visual features to the LLM hidden states and the second is responsible for post-processing the aggregated visual features to be aligned to the LLM hidden states. Adding a new MLP, the number of parameters in the model is increased and it is theoretically supposed to provide more expressiveness for processing information.

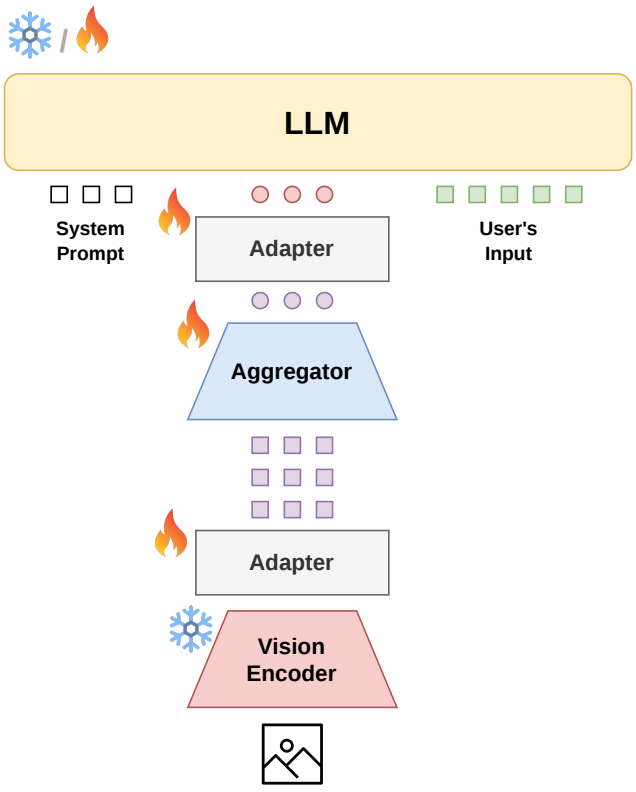
However, none of these theorized different pipeline variants could be considered better than the others, as they all have their own advantages and disadvantages. For this reason, the best choice is to experiment with all of them and evaluate their performance on downstream tasks, to understand which one is the best for the specific scenario.



(a) Text-influenced Vision-centric Aggregation with adapter only after.



(b) Text-influenced Vision-centric Aggregation with adapter only before.



(c) Text-influenced Vision-centric Aggregation with adapter both after and before.

3.4 Text-guided Vision Aggregation

In the previously described approaches (Section 3.2 and Section 3.3), visual features are aggregated without direct and active guidance from the text. Consequently, the visual aggregator must infer how to synthesize the most relevant visual information for the downstream tasks without access to the textual features. While the visual aggregation module learns generalized patterns for data synthesis during training, it can not dynamically adapt to the specific textual context of individual inputs. This inflexibility can lead to suboptimal performance, particularly in scenarios where the relevance of visual features heavily depends on the accompanying text.

This motivates the exploration of a more integrated approach - which can be referred to as *text-guided vision* aggregation - where the visual aggregation process is directly guided by the textual features (Figure 3.9). In this paradigm, the text features are not only used to influence the loss function during training but also play an active role in guiding the aggregation of visual features, allowing for a more dynamic and context-aware synthesis of visual information.

Especially in LLaVA's second stage, where the model is trained to answer complex questions about images, the text features can provide crucial contextual information that can help the aggregator to focus on the most relevant visual features for answering the question, potentially leading to a significant improvement in performance on downstream tasks. For example, considering the image in Figure 3.8 picked from MS COCO dataset, two different questions about the same image - e.g., *"What time is it?"* and *"How many people are in the image?"* - would require the aggregator to focus on different visual features of the image, such as the clock face with an OCR-like oriented downstream task for the first question and the people in the background for an object detection-like downstream task for the second question. In previous approaches, the aggregator would have produced the same aggregated visual features - presumably a general representation of the image - for both questions. In contrast, with a text-guided vision aggregation approach, the aggregator can dynamically adapt its visual feature selection based on the specific textual input, allowing it to synthesize different visual representations that are more relevant for each question, potentially leading to a significant improvement in performance on downstream tasks.

Figure 3.8: Example image from MS COCO dataset in which there is a clock on foreground and some people in the background [41].

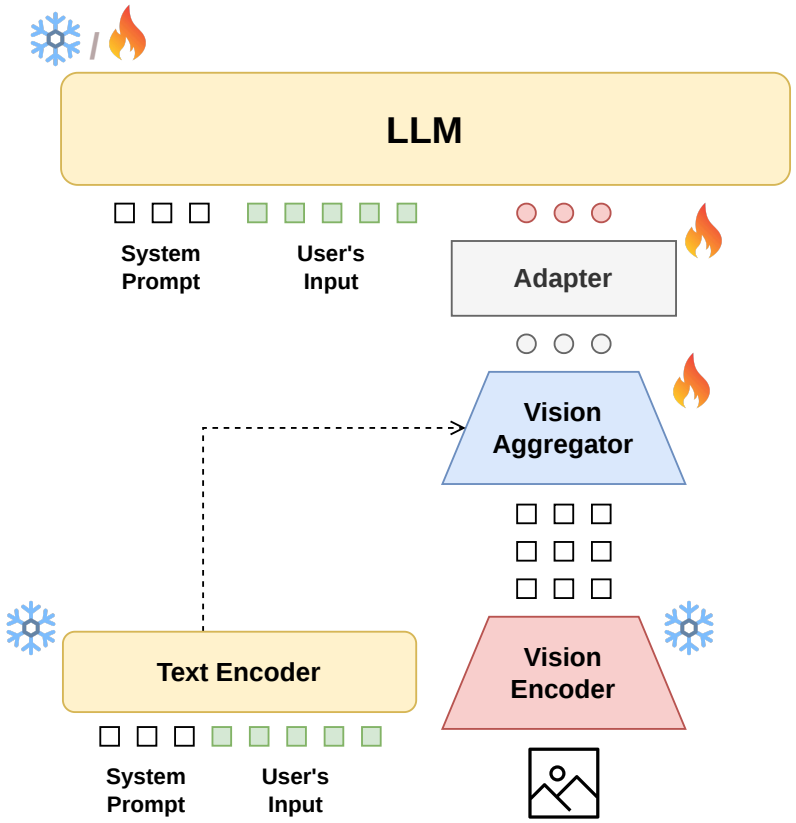


Figure 3.9: Text-guided Vision Aggregation architecture.

3.4.1 Text Encoder

The text features referred to in text-guided vision aggregation approach can be derived from different sources and the best choice may depend on the specific scenario and the architecture of the model. Possible solutions include: 1) a CLIP's text encoder, 2) a dedicated text encoder and 3) the LLM itself as text encoder.

When CLIP (Section 2.3.3) is used as vision encoder, the text features can be extracted from the coupled CLIP text encoder itself, which is trained to produce semantically meaningful representations of text that are aligned with the visual features extracted by the paired CLIP vision encoder. These features can provide a rich and informative representation of the textual input, which can be used to guide the visual aggregation process effectively. However, CLIP text encoder has two main limitations: 1) it is trained in contrastive learning setting to align images and their corresponding captions, so it may not be optimal for extracting text features that are meaningful for guiding the visual aggregation process when the downstream task is different from image captioning and may require a more fine-grained understanding of the text such as visual question answering; 2) the available pre-trained CLIP text encoders were trained using a relatively small encoder model (typically a BERT-base architecture), which has a limited context length (in the original released model only 77 tokens could be processed) that generally is not sufficient to capture the full textual context of the input, especially when the input text is a long and complex user's question.

Alternatively, instead of relying on the CLIP text encoder - for example when other vision encoders are used such as DINO (Section 2.3.4) - a dedicated (and pre-trained) text encoder can be used to extract the text features, which can be trained specifically for the purpose of guiding the visual aggregation process. This approach allows for more flexibility in the design of the text encoder and can potentially lead to better performance, as the text features can be optimized for the specific downstream tasks. Additionally, a dedicated text encoder can be chosen based on the specific requirements of the downstream task, potentially considering a model with a larger context length than the LLM decoder. However, this approach requires 1) additional computational resources and time to train the dedicated text encoder; 2) a larger amount of training data to achieve good performance and 3) more weights need to be stored and managed.

Instead of using a separate text encoder, the LLM itself can be used to extract the text features, leveraging its powerful language understanding capabilities to produce rich and informative representations of the textual input. This approach avoids the need for a separate text encoder

model which could have context length limitations and more resource requirements, as the LLM is already part of the architecture - it is already trained together with the rest of the components - and it does not have any context length limitation naturally.

Nevertheless, using the LLM as text encoder introduces additional complexity in the training process. In fact, if the LLM is involved both in the text encoding and in the final decoding process, the cost of back-propagation is approximately doubled, given that the gradients must be computed through the LLM twice: once for the text encoding and once for the final decoding, doubling the number of activations that must be stored during the forward pass and the number of computations required during the backward pass. Doubling the complexity of the back-propagation - that is the most resource-intensive part of the training process - the training time can increase significantly. Moreover, if the LLM's weights are modified during the training to be used as text encoder, their distribution may be destroyed in a classical *catastrophic forgetting* process, leading to a suboptimal performance in decoding part.

In order to mitigate this issue, the LLM can be used as text encoder in a frozen way, preventing any modification of its weights during the training process, eventually appending a learnable projector to align text encoder output distribution to next module. However, this approach may lead to a suboptimal performance, as the LLM is not optimized to produce text features that are meaningful for guiding the visual aggregation process, and simply adding a projector may not be enough to capture the specific nuances and requirements of the downstream tasks effectively.

Text Encoder Layer Selection

In all previous described cases, similar to the vision encoder, determining which text encoder layer to use for extracting text features is a crucial design choice that can significantly impact the performance of the model.

Because different layers may capture different levels of abstraction and information about the text, the most relevant features for guiding the visual aggregation process may be found in specific layers rather than others - first layers could be too focussed on semantic and grammar information, on the other hand last layers could be too focussed on pre-training objective (e.g., next token prediction). Especially when using the LLM itself as text encoder, the choice of which hidden states to use as text features can be critical because the deeper the hidden states are, the more computation is required to extract them, and the training time could increase significantly. For this reason, careful experimentation should be required to identify the optimal

layer for extracting text features that can effectively guide the visual aggregation process while balancing the computational cost.

A possible way to solve the issue of selecting the optimal layer for extracting text features X_{text} is to use a learnable mechanism which is fitted to select the most meaningful text encoder's layer. For example, the simplest approach is to learn a set of weights $W = [w_1, w_2, \ldots, w_L]$ over the L text encoder layers and to compute a weighted sum of the hidden states $H_1, H_2, \ldots, H_L$ of all the layers (Figure 3.10) - eventually after applying a *softmax* - allowing the model to learn how to combine information from different layers to produce a more informative representation of the text that can effectively guide the visual aggregation process:

$$X_{text} = \sum_{i=1}^{L} \sigma(w_i) \cdot H_i$$

However, this approach may introduce additional training instability and it requires computing the hidden states for all layers of the text encoder, which can be resource-intensive, especially for large models with many layers. In fact, if not properly regularized and handled, approaches that rely on combination of multiple layers may lead to instability during training and performance loss - due to the possible distinct distribution of vectors in the different hidden states. In fact, it is widely possible that different layers of a (Transformer) model produce embeddings which have different magnitude and direction, and simply applying a weighted sum to combine them may lead to a representation that is dominated, for instance, by the layers with larger magnitude.

A possible solution to mitigate this layer distribution incompatibility issue is to apply a light manipulation ϕ of the hidden states before applying the weighted sum - such as a normalization or a (linear) projection to a common space - allowing the model to learn how to effectively combine information from different layers while maintaining stability during training and ensuring that the resulting representation is meaningful for guiding the visual aggregation process:

$$X_{text} = \sum_{i=1}^{L} \sigma(w_i) \cdot \phi(H_i)$$

Alternatively, more sophisticated approaches were explored in literature, such as in ReT where a LSTM (*Long-Short Term Memory*, a type of *Recurrent Neural Network*) was used to process the hidden states of all layers sequentially - allowing the model to learn how to capture the temporal dependencies and relationships between the layers - potentially leading to a more informative representation of the text that can effectively guide the visual aggregation

process [65]. However, this approach may introduce additional complexity and computational overhead, as it requires processing the hidden states sequentially through an RNN, which can be resource-intensive, especially for large models with many layers.

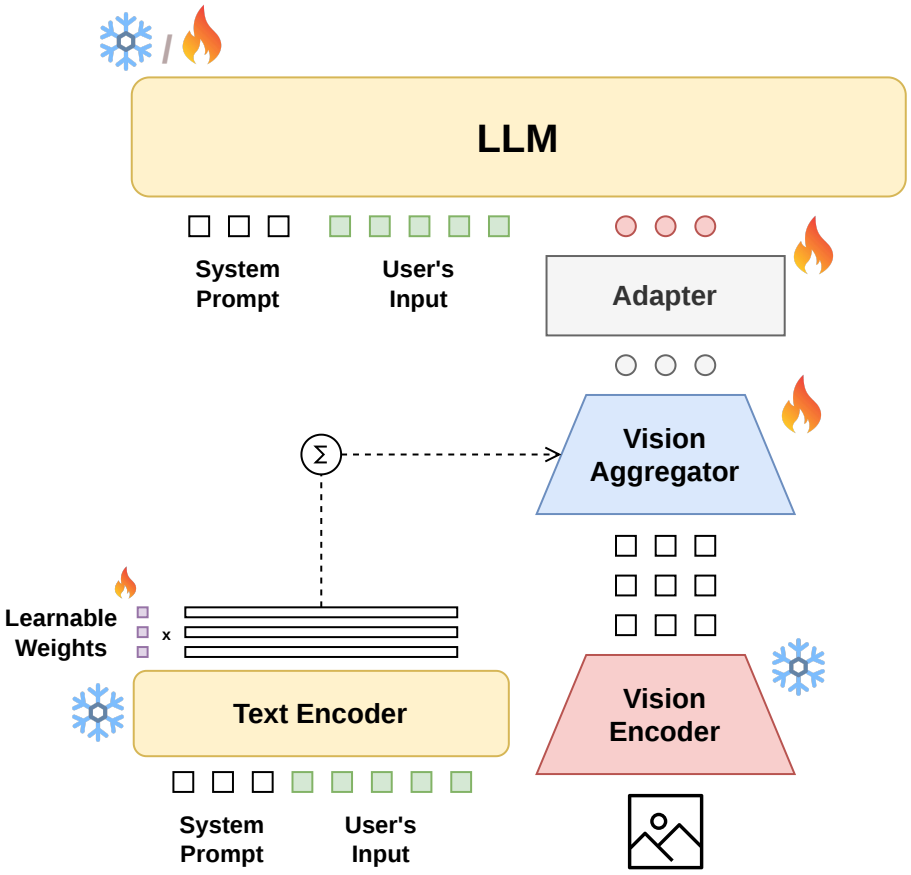


Figure 3.10: Text encoder layer selection through learnable weights that modulate the contribution of each layer.

3.4.2 Text Guidance Implementation

In order to implement a text-guided vision aggregation approach, a specific architectural design is required to allow the text features to directly influence the visual aggregation process. This can be achieved in several ways which can be grouped into two main categories: 1) conditioning the visual features before the aggregation process and 2) conditioning the aggregation process itself on the text features.

In the first case, the text features X_{text} are used to condition the visual features X_{vis} before they are passed to the aggregator. This can be achieved, for instance, through a Feature-wise Linear

Modulation (FiLM) mechanism, originally proposed for visual reasoning tasks [66]. Rather than simply concatenating the modalities, FiLM applies an affine transformation to dynamically adjust the visual representation based on the textual context (Figure 3.11b). This modifies the original visual features - vectors' magnitude and orientation are altered toward the interested region of the latent space - in a way that emphasizes or suppresses certain aspects of the visual information according to the text, allowing the model to focus on the most relevant visual features for the downstream tasks. Specifically, the model learns two functions, f and h, which map the text features to a set of scaling (γ) and shifting (β) parameters:

$$\gamma = f(X_{text}) \qquad \beta = h(X_{text})$$

These parameters are then used to apply a feature-wise affine transformation to the original visual features:

$$X'_{vis} = \gamma \odot X_{vis} + \beta$$

where $\odot$ denotes element-wise multiplication. Through this formulation, the model is able to dynamically modulate the visual representation according to the provided textual context prior to the final aggregation process.

This approach allows the text to influence the visual feature selection process indirectly, by modulating the visual features before they are passed to the aggregator. However, for instance in the FiLM case, affine transformation can only "turn off" or "scale up" the original visual features, but it can not introduce new information or relationships between the visual features that are not already present in the original representation. This can limit the model's ability to capture complex interactions between the modalities, especially in scenarios where the relevance of visual features is highly dependent on the specific textual context.

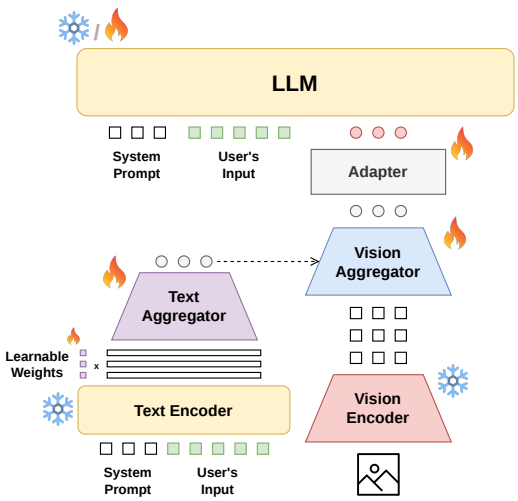
Instead, in the second case, the aggregation process itself is directly conditioned on the text features, allowing for a more integrated and dynamic interaction between the modalities during the aggregation. This can be achieved, for instance, by incorporating cross-attention mechanisms that allow the aggregator to attend to both the visual features and the text features simultaneously (similarly to what happens in BLIP-2's text encoder described in Section 2.6.2) [6] or using a set of learnable queries which are derived from the text features in a cross-attention mechanism - such as a perceiver - allowing the model's text extraction component to directly guide the visual aggregation process by providing initial queries that are already informed by the textual

context, potentially leading to a more effective selection of visual features that are relevant for the downstream tasks.

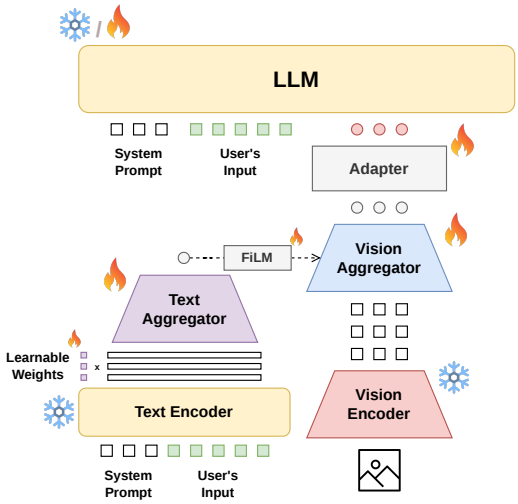
Even in this last case, there are several ways to initialize the queries from the text features and use them to aggregate the visual features. Considering that generally the text features have a shape that is neither compatible with the visual aggregator nor to simple linear/MLP projector, a way to bridge this gap must be implemented. For example, a text aggregator could be employed to summarize the text features into a compact set of vectors that can be used as queries for the visual aggregation process. For instance, if the text aggregator is a perceiver, a set of learnable queries with the same size of the visual aggregator's queries can be used to attend to the text features, allowing each query to capture different aspects of the textual input and produce a more informative starting point that can effectively guide the next visual aggregation process. As another option, a single dedicated query can be used to attend to the text features and it can be summed to a set of learnable queries equipped to visual perceiver aggregator, allowing the model to capture the overall textual context while still maintaining a set of learnable queries that can capture specific aspects of the visual features, potentially leading to a more effective guidance of the visual aggregation process. However, it is important to note that when an additional component such as the perceiver as text aggregator is introduced to process the text features before using them to guide the visual aggregation, the computational overhead of the model can increase significantly, gradient vanishing problem can arise and in any case the text aggregator could struggle to lean how to both capture the relevant information from the text and produce queries that are meaningful for guiding the visual aggregation process, especially in scenarios where the textual input is long or complex and the most relevant information is not easily identifiable.

Alternatively, if the LLM itself is used as text encoder, thanks to its causal language modeling training objective, the hidden states corresponding to the last token of the input text can be used as a compact representation of the entire textual input, which can be directly used as a query for the visual aggregation process. However, due to the tendency of LLMs to favor recent content - phenomenon also called *recency bias* [67] - hidden states of the last token may not always capture the most relevant information for guiding the visual aggregation process, especially in scenarios where the input text is long or complex and the most relevant information is not located at the end of the input. In such cases, relying solely on the last token's hidden states may lead to suboptimal performance, as it may not provide a comprehensive representation of the textual

context needed to effectively guide the visual aggregation process.



(a) Vision aggregation guidance using the output of text aggregator, eventually as initial queries for a visual perceiver.



(b) Vision aggregation guidance through a FiLM-like mechanism starting from the output of text aggregator.

3.4.3 Vision Encoder Layer Selection Guided By Text Input

Generally, the traditional LLaVA architecture relies on a predefined vision encoder layer to extract the visual features - which are then passed to the LLM through an adapter. For example, when CLIP (Section 2.3.3) is used as vision encoder, in literature, the penultimate layer has become the most selected layer to extract the visual features, given that the last layer is more specialized for the contrastive learning objective of CLIP, while the penultimate layer is more general and contains richer visual information that can be useful for downstream tasks [68].

Nevertheless, this choice is based on a combination of general intuition and some empirical experiments, but it may not be optimal for all scenarios, especially when the downstream task is complex, such as visual question answering. In fact, different layers of the vision encoder may capture different levels of abstraction and information about the visual input, and the most relevant features for guiding the visual aggregation process may be found in specific layers rather than others. For example, last layers may contain relevant features for more abstract downstream tasks, while earlier layers may provide low-level information useful in spatial reasoning. Considering Figure 3.8, if the question was *"What is the t-shirt color of the person on the left?"*, the most relevant features may be found in earlier layers - that capture more low-level

visual information such as colors and shapes - while if the question was *"What is represented in the image?"*, the most relevant features may be found in the last layers, which capture more high-level semantic information about the objects in the image.

Therefore, text features can be leveraged to dynamically select - thanks to a *router* module - the most relevant layers of the vision encoder for each specific input, allowing the model to extract visual features that are more aligned with the textual context and the downstream task, potentially leading to a significant improvement in performance on downstream tasks. The router module can be implemented in different ways, eventually following the most prominent approaches in the literature for Mixture of Experts models [38] - which are components that in this case can be used to dynamically select different layers of the model ("experts") based on the input, allowing the model to adapt its processing to the specific characteristics of each input and potentially improving performance on a wide range of tasks.

Similarly to what is described for text features aggregation, the most simple approach to implement a router is to use a simple MLP that takes an aggregation - or the hidden state related to last token when LLM is used - of the text features as input and produces a set of weights over the layers of the vision encoder, which are then used to compute a weighted sum of the features extracted from each layer, allowing the model to dynamically select the most relevant visual features for each specific input based on the textual context. Additionally, to mitigate the risk of instability potentially caused by the unbound distribution of the weights - that could lead to an explosion of the resulting feature magnitudes if the MLP produces large weights - a softmax or sigmoid function can be applied to the output of the MLP to ensure that the weights are normalized and bounded, preventing any single layer from dominating the resulting representation and allowing the model to effectively combine information from multiple layers when necessary. These two distinct approaches - softmax vs sigmoid - have their own advantages and disadvantages: softmax introduces a shared "budget" that encourages competition among layers, which can be more effective in scenarios where it is desirable to select few layers as the most relevant, while sigmoid can be more effective in scenarios where it is desirable to combine information from many multiple layers, allowing a more flexible and nuanced selection of visual features based on the textual context, but risking introducing a lot of noise if the router does not understand to "turn off" irrelevant layers.

Furthermore, even in this case, different layers can potentially have distinct distributions of features, and simply applying a weighted sum to combine them may lead to a representation

that is dominated by the layers with larger magnitude - leading to suboptimal performance. Solutions to mitigate this issue can be similar to the ones described for text features aggregation, such as applying a simple normalization or a (linear) projection to a common space before applying the weighted sum, alternatively following more sophisticated approaches which rely on RNNs [65] or other mechanisms to capture the relationships between the layers and produce a more informative representation of the visual features.

3.4.4 Turn-by-turn Text-guided Vision Aggregation

In the canonical visual aggregation approaches, the visual features are aggregated **once**, producing a single set of aggregated visual features that are then used throughout the entire generation process. This approach is based on the fact that there are no reasonable ways to dynamically change the visual features during the generation process, as the visual features are extracted and aggregated before the generation starts, and they remain fixed during the entire generation process.

However, in scenarios where the LLM can not rely on the entire set of visual features extracted by vision encoder (e.g., in a visual aggregation setup) and supposing a *multi-turn conversion* where the input text is long, divided into multiple parts that require complex understanding - such as LLaVA-like visual question answering - the visual information relevant for a specific question in the conversation may be different from the visual information relevant for another turn *within the same conversation*. For this reason, a single set of aggregated visual features represents a limitation: it may not be sufficient to capture all the relevant information needed for each turn.

For example, in a multi-turn conversation about an image, the first turn may require information about the overall scene and objects in the image, while the second turn may require more specific information about a particular object or region in the image. In this case, under the assumption that the entire set of original visual features can not be aggregated into a smaller set without any information loss, having a single set of aggregated visual features that is used for both turns may lead to suboptimal performance - as it may not provide the necessary level of detail and specificity needed for each turn - given that the visual aggregation module learns to generate a set of dense visual information that is reasonably useful for different types of user questions.

Relying on the user's input in the text-guided vision aggregation scenario, the visual features

can be coupled with each related user's question, allowing the vision extraction component of the model to adapt the visual representation to the specific requirements of each conversation turn and on downstream task. This can be achieved by implementing a mechanism that allows the model to aggregate the visual features based on the related textual context at each turn, effectively creating a *turn-by-turn* text-guided vision aggregation process that can capture the evolving needs of the conversation and provide more relevant visual information for each turn.

In particular, the vision encoder is used to extract visual features from the input image *only once* at the beginning of the conversation and these original "raw" visual features are stored as starting point for all turn aggregations. In the same way, during the training phase, the text features are extracted from the input text only once at the beginning in a single forward pass - if LLM is used, causal mask can be exploited to prevent data leakage - and then, for each turn of the conversation, the text-guided vision aggregation process is applied to the stored visual features, allowing the model to aggregate visual representation based on the specific textual input for that turn, without the need to re-extract the visual features from the image, which can be computationally expensive.

This approach allows for a more flexible and dynamic interaction between the text and visual modalities throughout the conversation, potentially leading to better performance on downstream tasks that require a nuanced understanding of both modalities. During inference, the same behavior is maintained with the only difference that the text features are extracted at each new turn of the conversation, given that the full input text is unknown initially - slightly increasing the computational cost during inference, as the text features must be extracted at each turn instead of only once at the beginning.

In other words, this *turn-by-turn* training process is comparable to train the model on synthetic data derived from the original data, where each synthetic sample corresponds to a single turn of the conversation, with its related text features and its related aggregated visual features.

More in detail, considering an original batch size B, a number of turns T_i for each conversation i in the batch and a (maximum) sequence length L, the text encoder forward pass produces an output X_{text} with a shape of (B, L, D) where D is the dimension of the text features and depends on the specific text encoder model. Then, for each sample $i \in \{1, 2, ..., B\}$, $X_{text,i}$ is divided into a set of T_i chunks - identifiable for instance by a specific sentinel token in the input text that separates the turns - with each chunk corresponding to the text features of a specific turn of the

conversation, resulting in a set of $\bar{B} = \sum_{i=1}^{B} T_i \geq B$ chunks (Figure 3.12).

After that, an independent set of aggregated visual features is produced from each chunk (i.e., turn), synthesizing the visual features F_{vis} based on the text features of that specific turn, resulting in $\bar{B}$ sets of aggregated visual features - which are then passed to the LLM together with the corresponding text features for each turn to produce the final output (e.g., the answer of related question).

In this scenario, the LLM used as text encoder is particularly effective, because it can be used to extract the text features for each turn without any data leakage, thanks to its causal language modeling the token embeddings are not influenced by the next turns, allowing to perform only a single forward pass for each sample. In contrast, if a separate text (bidirectional) encoder were used, the text features for each turn would need to be extracted through multiple forward passes, one for each turn, which can significantly increase the computational cost during both training and inference. Naturally, in the same way thanks to causal mask, the LLM as decoder can attend only to the text features of the current turn and to the corresponding aggregated visual features, preventing any data leakage.

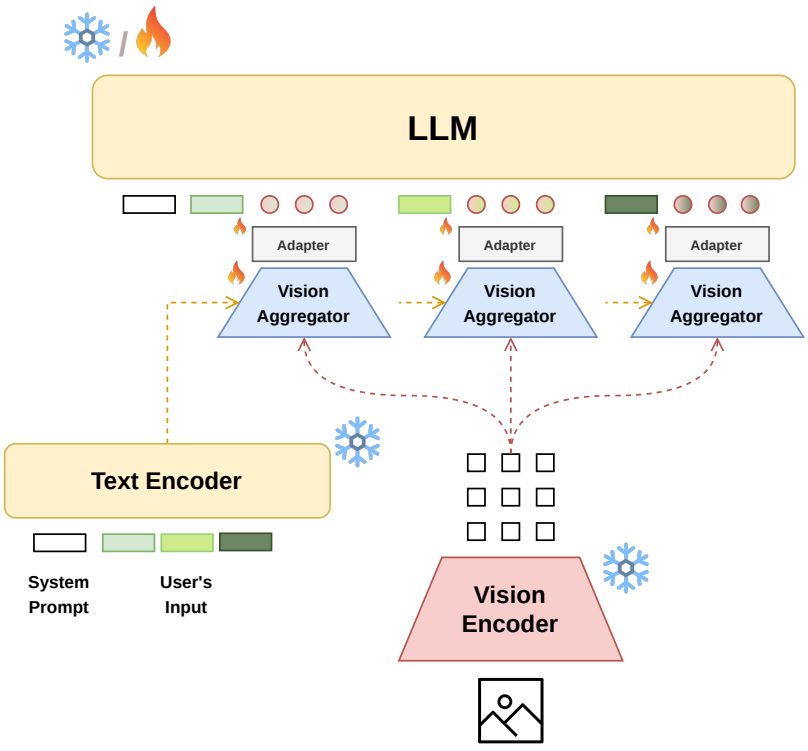


Figure 3.12: Multi-turn text-guided vision aggregation architecture.

3.4.5 Mitigating Computational Overhead Through KV Caching

The auto-regressive nature of Transformer model (Section 2.2) inference introduces a significant computational bottleneck. Since each new token x_t is generated based on the entire preceding sequence $x_1, x_2, ..., x_{t-1}$, the theoretical implementation would require the model to re-process the entire context at every step.

To mitigate this inefficiency, a technique called *Key-Value (KV) Caching* is employed. During the generation process, the key and value matrices computed for each token in the input sequence are stored in a cache. When generating the next token, instead of recomputing the attention scores for the entire sequence, the model can simply retrieve the relevant keys and values from the cache, significantly reducing the computational overhead and enabling faster inference.

This exact approach can be applied to the text-guided vision aggregation scenario when the LLM is used both as text encoder and as final decoder. In fact, in this configuration, the LLM should process the input text twice: once to extract the text features - that will be used to guide the visual aggregation process - and once to generate the final output based on the aggregated visual features and the input text. By leveraging KV caching, the model can avoid redundant computations during the second forward pass through the LLM, as the keys and values computed during the first pass can be re-used during the second one, significantly reducing the computational overhead and enabling faster inference.

In standard LLaVA architecture, the visual features extracted by the vision encoder and projected to the LLM model dimension are concatenated **before** the text token embeddings of the question - as if they were a visual integration of text context - which are then passed together to the LLM. Remembering the causal LLM behavior, this approach was chosen to allow the visual features to be attended by all next tokens of the question, allowing the model to effectively integrate visual information with the textual context from the very beginning of the generation process.

In the turn-by-turn text-guided vision aggregation scenario when the LLM is used as text encoder, during the initial forward pass, the model processes exclusively the conversational text tokens - visual features must be omitted at this stage, given that their aggregation inherently depends on this processing. Consequently, the text embeddings computed during this initial phase attend solely to the preceding textual context of the conversation. For this reason, in order to re-use the text features computed during the first forward pass in the second one thanks to cache mechanism, the visual features must be concatenated **after** the text token embeddings of

the question - as if they were a provided "visual answer" to the question - and before the answer that will have to be generated by the LLM.

In this way, the overhead introduced by the additional forward pass through the LLM is significantly reduced during inference, as the keys and values computed during the first pass can be re-used during the second one. Overall, compared to proposed text-influenced vision-centric aggregation approach (Section 3.3), in inference the turn-by-turn text-guided vision aggregation approach with LLM as text encoder and KV caching only introduces the additional computational cost to have multiple independent visual aggregation guided by the text features of each turn, but apart from second LLM function call - negligible with respect LLM forward pass - it does not introduce any additional computational cost regarding the double LLM forward pass, which is the most resource-intensive part of the inference process.

Nevertheless, during training with a frozen LLM in first forward pass, even if theoretically the KV caching mechanism could be applied, from a technical point of view, the back-propagation process in common deep learning frameworks (e.g., PyTorch) does not allow computing gradients through the cache mechanism - as it is designed to be used during inference and not during training. For this reason, unfortunately, the power of KV caching to reduce the computational overhead of the double forward pass through the LLM can be fully exploited only during inference. However, even if the KV caching mechanism can be applied only at inference time, during the training phase the back-propagation dominates the computational cost over the frozen LLM first forward pass, making the additional computational cost of the double forward pass acceptable and manageable.

3.4.6 Limitations

Despite the potential benefits of a turn-by-turn text-guided vision aggregation approach, it also introduces additional complexity in the model architecture and training process. First of all, if in all turns new visually aggregated features are injected, then the number of visual features that are passed to the LLM can increase significantly - potentially surpassing the original total amount of visual features fed to the LLM in a standard LLaVA architecture.

For this reason, if the aggregation module is less capable of understanding the visual features and to synthesize them in a more compact representation compared to the LLM itself, the number of aggregated visual features must be carefully selected to balance the trade-off between providing sufficient visual information for each turn and maintaining a manageable

computational cost: too many visual features can lead to a significant increase in computational cost and memory usage, while too few visual features may not provide enough information for the LLM to perform well on downstream tasks. For example, when *CLIP-Vit-Large-336* is used (576 visual features), a reasonable choice could be between 4 and 16 aggregated visual features for each turn, in order to have between 36 and 144 turns before surpassing the original amount of visual features fed to the LLM in a standard LLaVA architecture.

Otherwise, if the aggregation module is capable of synthesizing and pre-elaborate the visual features - in order to provide a more exploitable representation to the LLM - then the disadvantage of accumulating visual features at each turn can be minimized by the boost in performance that the model can achieve thanks to the more effective visual representation provided by the aggregator, which can allow the model to perform better on downstream tasks even with a smaller number of visual features, potentially leading to a significant improvement in performance while still maintaining a manageable computational cost.

Nevertheless, generally to reduce the computational cost and the latency during inference of additional components, aggregator module must be designed to be as simple as possible, with a (relatively) small number of parameters, which may limit its ability to effectively synthesize the visual features and provide a more exploitable representation to the LLM. For this reason, maintaining an aggregator that allows to reduce global latency is more reasonable than others.

Additionally, as already described previously, the training process becomes more complex and time-consuming compared to text-influenced vision-centric aggregation approach (Section 3.3), because the *turn-by-turn* processing needs to consider a bigger synthetic batch size - which requires a larger amount of computation resources, especially the GPU memory.

4. Experiments

In this chapter, a wide range of experiments is presented, in order to evaluate the efficacy of the methods proposed in Chapter 3 and to better understand which design choices are more effective to reach performance comparable to the baselines.

Traditional token pruning models exposed in Section 2.7 were evaluated on a relatively small set of benchmarks, generally a subset of the famous Cambrian dataset collection [69] - which was initially proposed to evaluate the performance of vision-centric models and it contains a wide variety of tasks that require different levels of visual understanding and reasoning, such as image captioning, visual question answering, visual reasoning and so on.

In this work, proposed models are evaluated against the standard LLaVA architecture - which represents the baseline - and against the most prominent token pruning models in literature, such as OC-VTP (Section 2.8) and LLaVA-Mini (Section 2.11) on a wide set of Cambrian dataset benchmarks, in order to better explore in which scenarios the proposed methods struggle with visual feature aggregation and in which they can achieve comparable performance with other models. Additionally, LLaVA with only CLS token is sometimes used as a very simple aggregation baseline, in order to understand the goodness of the proposed method compared to a very simple approach.

To comprehensively evaluate the proposed methods, several distinct LLMs are considered. First, *LLaMA 3.2 1B* (parameters) serves as an initial prober within a smaller, highly controllable setting. This choice enables rapid iteration over various design configurations, facilitating a deeper understanding of the methods' foundational behavior. Second, *Vicuna 7B* is included given its historical significance in the original LLaVA paper. Testing on this larger and more capable architecture helps determine how effectively the methods scale with model size. Consequently, it allows for a detailed assessment of whether the aggregators process visual features less, equally or more effectively than the intrinsic attention layers of the LLM itself. Finally, *Qwen 3* family models are examined across multiple sizes, which helps to understand how the proposed methods perform on different small- and medium-sized architectures.

4.1 Experiments on Text-agnostic Vision-only Aggregation

Described in Section 3.2, the text-agnostic vision-only aggregation method is a natural evolution of traditional token pruning techniques presented in Section 2.7 where the aggregator is a trainable component. Given that approaches with a similar philosophy such as OC-VTP (Section 2.8) emerged exploiting the Slot Attention mechanism (Section 3.1.4) as a promising direction, an investigation of this method - being fully compatible with the proposed pipeline - is conducted thanks to a set of experiments exposed in this section.

Similarly to OC-VTP, a Slot Attention-based aggregator is developed to distill the visual information into a fixed number of slots, which are then used to provide visual features for the LLM - following standard LLaVA framework. However, unlike OC-VTP - which prunes visual features based on the attention weights - the produced slots are directly used as input for the LLM, shifting the focus from pruning to aggregation. This design choice is motivated by the fact that Slot Attention mechanism is a powerful tool for learning how to aggregate visual features into a compact visual concept object-related representation, which can then be effectively used by the LLM to perform downstream tasks.

Generally, OCL-based architectures rely on the DINO family vision encoder (Section 2.3.4) - generally small ones are sufficient - to extract visual features, given the emergent capability to produce semantically meaningful features related to objects in the images - without any supervision, which means that the produced features were not affected by any language bias during training. Vision encoders that rely on supervised learning with language supervision (e.g., CLIP, described in Section 2.3.3) are expected to produce features that are already biased towards the language modality, generating an embedding space in which concepts that are close in the language space are also close in the visual space.

For example, an image of a *dog* and an image of a *park* could be closely embedded in the visual space because they are often close in the language space, even if they are not visually similar.

For this reason initial experiments are conducted with DINOv2 vision encoder, and then a comparison with CLIP is performed to understand the impact of this choice on the performance of the Slot Attention-based aggregator.

4.1.1 Preliminary Experiments Between CLIP and DINO Vision Encoders

First of all, a preliminary explorative evaluation is conducted to measure performance gap between CLIP and DINOv2 vision encoder both on OCL and MLLM setups, in order to understand how an image-text pre-alignment of the visual features can affect the performance of the models in various tasks.

OCL models are typically evaluated on *instance segmentation* metrics because they are designed to learn to associate visual features with specific objects in the image, and the produced slots are expected to correspond to specific objects or regions in the image, so it is important to evaluate how well the produced slots correspond to the ground truth object masks. Specifically, several metrics are considered: 1) *Adjusted Rand Index* (ARI) which measures the similarity between the predicted slots and the ground truth object masks; 2) *Foreground ARI* (FG-ARI) which is a variant of ARI that is focused only on the foreground objects - i.e., main objects in the scene; 3) *Mean Best Overlap* (mBO) which measures the average overlap between the predicted slots and the ground truth object masks; 4) *Mean Intersection over Union* (mIoU) which measures the average intersection over union between the predicted slots and the ground truth object masks.

Generally, given that OCL models are designed to produce permutation invariant slots, association between the produced slots and the ground truth object masks is performed by using the *Hungarian algorithm* [70], which finds the optimal one-to-one matching between the predicted slots and the ground truth object masks based on a cost function that measures the similarity between them.

Among the most powerful Slot Attention-based models with dynamic slot allocation, MetaSlot (Section 2.5.3) is initially considered to train a Text-agnostic Vision-only Aggregator. Object masks are provided by the *alpha mask* of MetaSlot's *Spatial Broadcast Decoder*, but similar results are obtained also expanding to image resolution the attention weights of the slots on the visual features, so each slot is associated with a specific region of the image, which is expected to correspond to a specific object or part of an object in the image.

In Table 4.1 - and for a better understanding in Figure 4.1 - a comparison of OCL metrics described above is presented, varying number of considered slots and using the original proposed hyperparameters. Intuitively, DINO-based models outperform models of CLIP-family on OCL metrics, in all settings, thanks to its emergent capability to naturally focus on coherent semantic regions.

In particular, DINO-based models achieve a significant improvement - doubling the performance of CLIP - in all provided metrics, demonstrating their effectiveness in object discovery and segmentation. Multiple sizes are considered for DINO-based models, but there are not significant differences in performance between them, confirming the fact that even small DINO vision encoders are effective in producing semantically meaningful features that can be used by the Slot Attention-based aggregator to learn to associate visual features with specific objects in the image.

Moreover, it is particularly evident in Figure 4.1 that regardless of the size of the model, DINO-based models exhibit a slight performance degradation trend when the number of slots is increased in ARI and mIoU metrics, while CLIP-based models performance tends to remain more stable varying the number of slots.

This could be explained by the fact that DINO-based models are more effective in learning to associate visual features with specific objects, so when the number of slots is increased - even if MetaSlot provided a masking mechanism to avoid the association of multiple slots with the same object - some slots could be associated with multiple small parts in the background, which could lead to a loss of performance in terms of ARI and mIoU metrics, this is supported by the fact that the performance remain stable in FG-ARI and mBO metrics, which are more focused on the foreground objects, so the association of multiple slots with small parts in the background might not affect the performance in these metrics.

CLIP-based models, on the other hand, are less effective in learning to associate visual features with specific objects, so even when the number of slots is increased, performance is not affected by the object association degradation. Additionally, it is interesting to note that - following the expectation - even in these scenarios, the penultimate CLIP layer is more effective than the last one, thanks to the fact that the last layer of CLIP is more affected by the language bias, so the produced features are more aligned with the language space, which could lead to a loss of performance in terms of OCL metrics - which are more focused on the visual modality - while the penultimate layer is less affected by this bias, so it can produce features that are more effective for OCL tasks.

Table 4.1: MetaSlot instance segmentation OCL metrics.

Model	Resolution	Patch	Slot	ARI ↑	FG-ARI ↑	mBO ↑	mIoU ↑
DINOv3-L	224x224	16x16	5	23.40	37.54	28.59	22.89
			7	21.16	37.70	29.45	23.06
			9	19.19	40.20	**29.96**	21.98
DINOv3-L	256x256	16x16	5	24.36	40.05	28.93	23.30
			7	21.34	42.39	29.29	22.95
			9	19.34	40.77	29.54	22.41
DINOv3-S	224x224	16x16	5	24.67	38.26	28.16	**24.16**
			7	22.45	41.38	29.16	22.51
			9	19.07	38.12	29.66	22.07
DINOv3-S	256x256	16x16	5	23.65	37.33	28.52	24.01
			7	21.59	39.62	29.47	22.33
			9	19.25	42.55	28.87	22.70
DINOv2-L	224x224	14x14	5	**25.73**	37.26	29.21	22.46
			7	21.59	40.88	29.49	23.08
			9	17.89	38.24	29.46	22.53
DINOv2-S	224x224	14x14	5	24.44	37.55	28.76	23.14
			7	21.83	44.61	28.55	23.14
			9	19.43	**42.98**	29.20	22.02
CLIP-ViT-L	224x224	14x14	5	2.50	15.00	12.50	17.00
Last Layer			7	1.50	16.50	12.00	20.00
			9	2.00	13.50	12.50	15.50
CLIP-ViT-L	224x224	14x14	5	11.50	22.00	16.50	15.00
Penultimate Layer			7	10.50	23.00	17.00	14.00
			9	12.00	25.50	18.00	14.50

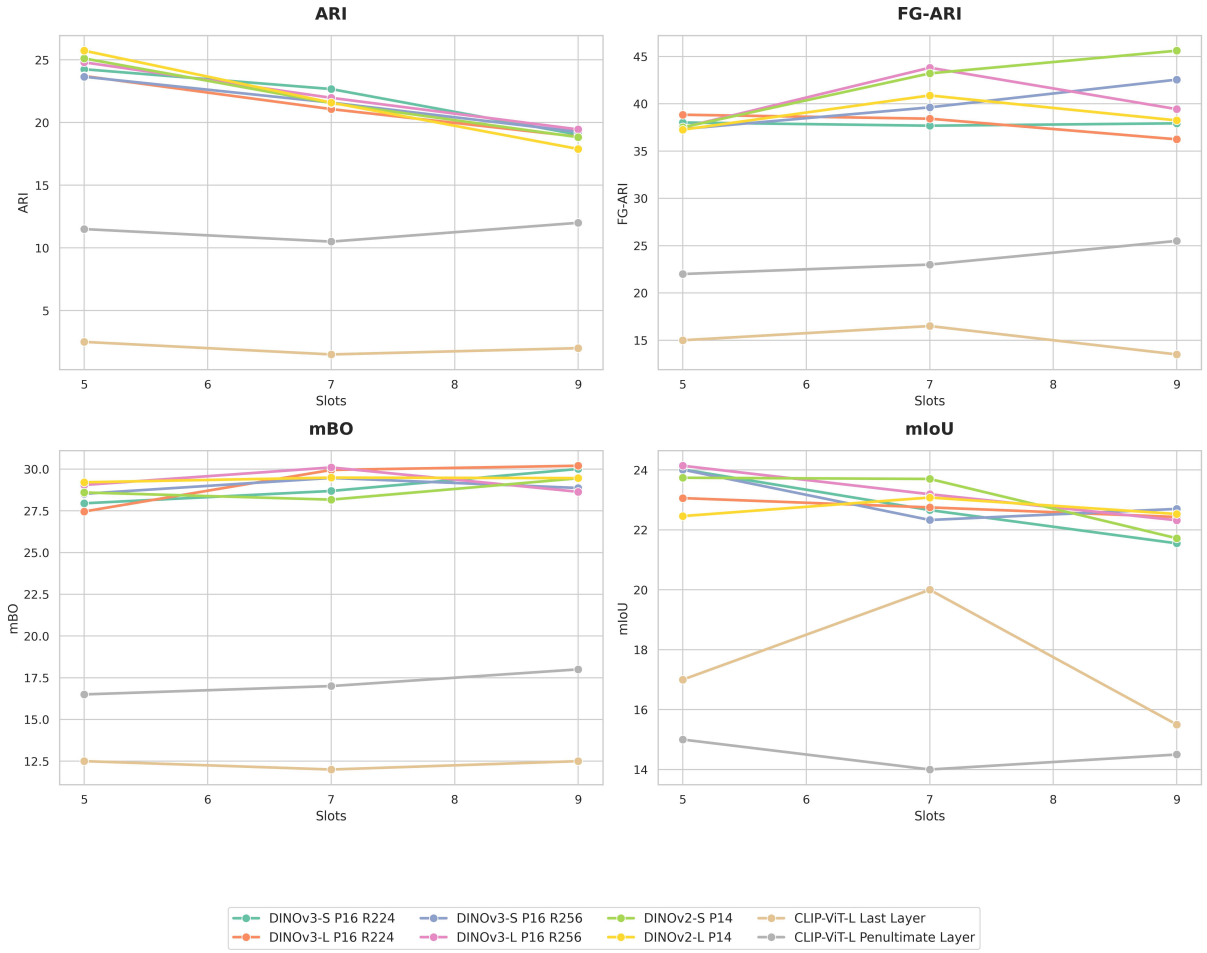


Figure 4.1: MetaSlot OCL metrics on several configurations.

Furthermore in Table 4.2, standard LLaVA architecture based on LLaMA 3.2 1B with CLIP ViT Large (penultimate layer) vision encoder is compared with a standard LLaVA architecture with DINOv2 Small - with a resolution of 224×224 pixels and 16×16 patches, excluding CLS token - vision encoder and with a baseline without vision (i.e., LLaMA 3.2 1B only).

Overall, it is immediate apparent that the standard LLaVA architecture with CLIP vision encoder outperforms the one with DINOv2 vision encoder, which in turn outperforms the baseline without vision, confirming the importance of the choice of the vision encoder and the impact of the image-text pre-alignment on the performance of the model.

The importance of the pre-alignment with the language modality is reasonable, as already mentioned, given that the standard LLaVA architecture relies on a simple projection to align the visual features with the language space, so if the produced visual features are already aligned with the language space, it is easier for the model to learn to associate visual features with specific concepts in the language space.

Moreover, it is relevant to notice that there are not correlations between the OCL metrics and the performance in MLLM tasks, given that DINO-based models outperform CLIP-based models in OCL metrics, but they are outperformed by CLIP-based models in MLLM tasks, demonstrating that the emergent capability of DINO vision encoders to produce semantically meaningful features related to objects in the images is not enough to achieve a good performance in MLLM tasks.

Table 4.2: Comparison between different vision encoders and a text-only baseline using LLaMA-3.2-1B in standard LLaVA setup.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
CLIP-ViT-Large-336 (Standard LLaVA)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
CLIP-ViT-L-336 MetaSlot With Masking	55.4	49.8	75.3	**1161.6**	42.0	51.9	38.6	**62.6**	**34.0**	8.4	49.3	23.6	10.2	2.7	40.0	41.6	**32.8**	42.2	5.0	27.3	**56.7**	51.0	29.1	51.7	58.2	55.5	60.2
	-15.9%	-14.4%	-10.3%	-11.9%	-30.1%	-15.4%	-2.4%	-6.3%	-0.4%	+3.6%	+0.5%	-30.1%	-15.2%	-90.7%	-18.9%	-8.3%	-10.6%	-12.1%	-61.0%	-30.8%	+22.0%	-2.4%	-13.3%	-8.3%	+13.0%	-6.5%	-0.1%
CLIP-ViT-L-336 MetaSlot Without Masking	56.1	50.4	76.1	**1161.6**	42.6	**53.2**	38.4	61.9	33.6	8.7	49.3	23.7	10.1	2.8	40.0	42.1	32.6	42.7	3.8	27.6	56.3	**52.1**	29.1	52.3	**61.0**	55.0	**63.0**
	-14.9%	-13.4%	-9.3%	-11.9%	-29.1%	-13.2%	-3.0%	-7.4%	-1.6%	+7.1%	+0.5%	-29.7%	-16.1%	-90.0%	-18.9%	-7.3%	-11.3%	-11.1%	-70.8%	-30.1%	+21.0%	-0.3%	-13.3%	-7.2%	+18.5%	-7.4%	+4.4%
CLIP-ViT-L-336 CLS only	56.5	49.8	72.7	**1161.6**	**48.7**	**53.2**	38.8	61.9	33.2	**9.3**	**50.9**	25.6	**10.4**	**8.5**	**41.0**	42.4	31.1	**43.2**	2.5	24.0	54.5	50.6	**29.8**	51.7	56.9	55.0	59.5
	-14.2%	-14.4%	-13.3%	-11.9%	-19.0%	-13.2%	-1.8%	-7.4%	-2.5%	+14.3%	+3.6%	-24.3%	-13.5%	-70.3%	-16.8%	-6.5%	-15.4%	-10.1%	-80.5%	-39.2%	+17.3%	-3.1%	-11.2%	-8.3%	+10.5%	-7.4%	-1.3%
DINOv2-S MetaSlot With Masking	53.2	50.6	**78.6**	1068.0	37.0	46.6	37.1	61.9	32.5	6.5	47.6	23.4	9.6	1.7	37.9	44.4	31.9	41.1	**6.5**	**39.4**	40.8	49.6	28.2	52.2	54.0	58.5	55.3
	-19.2%	-13.1%	-6.3%	-19.0%	-38.4%	-24.0%	-6.1%	-7.3%	-4.7%	-19.8%	-3.1%	-30.8%	-20.0%	-94.0%	-23.1%	-2.2%	-13.0%	-14.4%	-49.6%	-0.3%	-12.3%	-5.0%	-16.1%	-7.4%	+4.9%	-1.5%	-8.3%
DINOv2-S MetaSlot Without Masking	53.9	**50.9**	**78.6**	1061.4	39.5	47.2	37.2	62.2	33.2	5.3	47.9	23.7	9.5	1.9	38.1	**45.1**	31.7	41.3	5.9	38.3	41.3	49.0	28.3	**52.8**	50.7	**60.0**	53.0
	-18.3%	-12.5%	-6.3%	-19.5%	-34.3%	-23.0%	-6.0%	-6.9%	-2.6%	-34.6%	-2.4%	-30.0%	-20.8%	-93.3%	-22.7%	-0.7%	-13.7%	-14.0%	-54.3%	-3.0%	-11.2%	-6.3%	-15.8%	-6.4%	-1.6%	+1.0%	-12.1%
LLaMA 3.2 1B only	38.5	39.9	48.0	764.7	24.3	42.0	35.0	59.3	33.2	8.7	38.8	21.3	9.3	2.3	36.2	37.4	28.0	38.5	0.0	22.9	50.5	45.3	26.0	46.7	54.6	50.0	49.2
	-41.6%	-31.5%	-42.8%	-42.0%	-59.6%	-31.5%	-11.4%	-11.2%	-2.5%	+7.1%	-21.0%	-36.9%	-22.1%	-91.8%	-26.5%	-17.6%	-23.9%	-19.8%	-100.0%	-42.1%	+8.5%	-13.3%	-22.5%	-17.2%	+6.0%	-15.8%	-18.4%

4.1.2 MetaSlot-based Aggregator with DINO Vision Encoder

Focusing again on the Slot Attention-based aggregator provided by MetaSlot, in Table 4.3 a comparison between the standard LLaVA architecture with the already considered DINOv2 Small vision encoder and the related MetaSlot-based aggregator with DINOv2 Small vision encoder is presented. MetaSlot with originally proposed hyperparameters is considered.

In particular, the number of slots is 7 - the average number of objects in MS COCO dataset's images - with a slot dimension of 256 - which is the same dimension of the visual features produced by DINOv2 Small vision encoder (no projection is needed to align the visual features with the slot space, but the projection to LLM embedding space still remains). 50,000 training steps are needed to reach convergence with a batch size of 32 and a learning rate of $2 \cdot 10^{-4}$. Number of refinement iterations is set to 3, which is the de facto standard in most of the OCL related works that leverage Slot Attention, and the training objective is based on the MSE loss over the visual features, which is the same used in DINOSAUR [40] (Section 2.5.1).

Based on the results, the gap between the standard LLaVA architecture with LLaMA 3.2 1B and DINOv2 Small and the proposed method with the pre-trained MetaSlot-based aggregator is

very small and in some cases the MetaSlot-based aggregator **outperforms** the baseline - which relies on *36 times* more visual tokens - demonstrating the validity of aggregation technique in learning to distill the most relevant information into the dense visual concept which can be then directly used in downstream tasks, simplifying the elaboration process in the LLM layers (Figure 4.2).

Despite the surprising aggregation capability, this approach is not able to outperform the standard LLaVA architecture in which visual features are completely replaced with the single CLS token, demonstrating that CLS token still remains a powerful starting point to provide a sort of summary of the visual information to perform downstream tasks.

This could be reasonable because the CLS token was trained together with other visual features in the DINO pre-training phase in which many more images are elaborated by the model with respect to the pre-training phase of the Slot Attention-based aggregator.

Additionally, thanks to the capability of MetaSlot to provide also a masking mechanism - to avoid the association of multiple slots with the same object - both variants - with and without the masking mechanism - are considered in the comparison. This comparison is particularly interesting to understand if a single slot is enough to aggregate the visual information of the related object.

Unexpectedly, the variant without the masking mechanism outperforms the one with the masking mechanism. This counter-intuitive result could be explained by the fact that even when multiple slots map to the same object, they may capture distinct aspects of that object, thereby providing complementary information, which can lead to a better performance in downstream tasks.

Moreover, this demonstrates that even if multiple slots are associated with the same object, the model is still able to learn to distill the most relevant information into the dense visual concepts, which can then be effectively used by the LLM to perform downstream tasks.

In other words, it is demonstrated that in the presented configuration, a Slot Attention-based aggregator - in particular MetaSlot - is effective in aggregating the majority of the visual information, preserving the most relevant features for downstream tasks, but it is not able to distill all the relevant information of a region of the image (e.g., an object) into a single slot. For this reason, aggregators that do not force a strict association between a slot and an object can be more effective in learning to distill the most relevant information into the dense visual concepts.

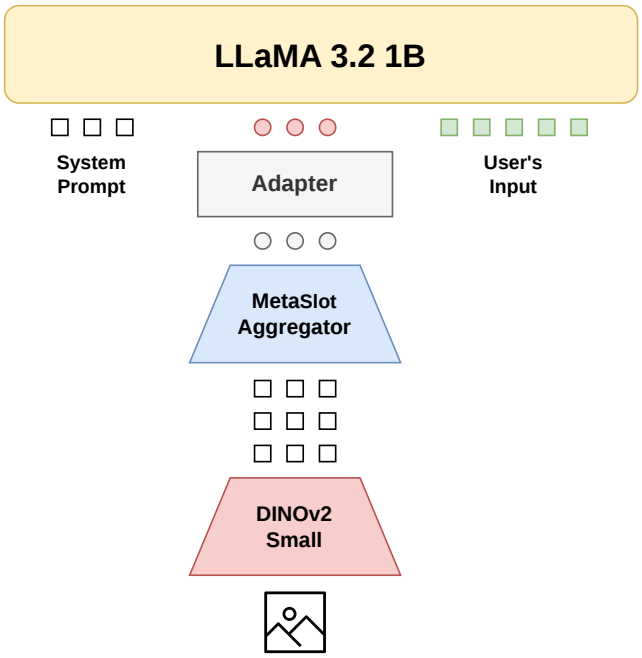


Figure 4.2: LLaVA with LLaMA 3.2 1B and DINOv2-Small vision encoder aggregated by MetaSlot.

Table 4.3: Comparison considering LLaVA with DINOv2-Small and LLaMA 3.2 1B of proposed MetaSlot configurations with 7 slots (both unmasked and masked variants) against standard and CLS-only variants. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	58.6	55.3	81.4	1160.3	45.1	53.3	38.1	62.8	33.5	7.8	48.1	21.8	9.4	2.3	37.5	38.1	32.8	43.7	6.5	36.3	44.6	53.9	30.6	54.7	59.7	58.8	65.8
MetaSlot (With Masking)	53.2 -9.2%	50.6 -8.5%	**78.6** -3.4%	1067.5 -8.0%	37.0 -17.9%	46.6 -12.6%	37.1 -2.6%	61.9 -1.4%	32.5 -3.0%	**6.5** -16.7%	47.6 -1.0%	23.4 +7.3%	**9.6** +2.1%	1.7 -26.1%	37.9 +1.1%	44.4 +16.5%	32.0 -2.4%	41.1 -5.9%	**6.5** 0.0%	**39.4** +8.5%	40.8 -8.5%	**49.6** -8.0%	28.2 -7.8%	52.2 -4.6%	**54.0** -9.5%	58.5 -0.5%	**55.3** -16.0%
MetaSlot Without Masking	**53.9** -8.0%	**50.9** -7.9%	**78.6** -3.4%	1061.7 -8.5%	**39.5** -12.4%	**47.2** -11.4%	**37.2** -2.4%	**62.2** -1.0%	**33.2** -0.9%	5.3 -32.1%	47.9 -0.4%	23.7 +8.7%	9.5 +1.1%	1.9 -17.4%	**38.1** +1.6%	45.1 +18.4%	31.7 -3.4%	41.3 -5.5%	5.9 -9.2%	38.3 +5.5%	41.4 -7.2%	49.0 -9.1%	**28.3** -7.5%	**52.8** -3.5%	50.7 -15.1%	**60.0** +2.0%	53.0 -19.5%
CLS only	52.2 -10.9%	48.9 -11.6%	72.9 -10.4%	**1073.3** -7.5%	38.8 -14.0%	46.9 -12.0%	**37.2** -2.4%	61.5 -2.1%	32.3 -3.6%	6.4 -17.9%	**48.6** +1.0%	**24.0** +10.1%	9.4 0.0%	**2.6** +13.0%	36.9 -1.6%	**47.1** +23.6%	**33.0** +0.6%	**44.1** +0.9%	5.2 -20.0%	34.8 -4.1%	**47.7** +6.9%	48.0 -10.9%	28.0 -8.5%	51.1 -6.6%	52.5 -12.1%	52.9 -10.0%	**55.3** -16.0%

4.1.3 MetaSlot-based Aggregator with CLIP Vision Encoder

Similarly to experiments conducted on DINOv2-based models, a comparison considering CLIP-based models is presented. In Table 4.4, the standard LLaVA architecture with CLIP-ViT-Large-

336 is compared with the related MetaSlot-based aggregator with the same vision encoder - even in this case, preserving original proposed MetaSlot's hyperparameters during training.

In this case, the gap between the standard LLaVA architecture and the MetaSlot-based aggregator is more significant compared to DINOv2-based models, demonstrating that the aggregation capability of the Slot Attention-based aggregator is less effective when CLIP vision encoder is used.

Nevertheless, differently from DINOv2-based models, the MetaSlot-based aggregator with CLIP vision encoder is able to obtain comparable results to the standard LLaVA architecture with CLS token, demonstrating the goodness of the aggregation technique also when the vision encoder is a text-image pre-aligned one.

Table 4.4: Comparison considering LLaVA with CLIP-ViT-Large-336 and LLaMA 3.2 1B of proposed MetaSlot configurations with 7 slots (both unmasked and masked variants) against standard and CLS-only variants. DINOv2-Small and LLM variants are also considered. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA (CLIP-ViT-L-336)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
CLIP-ViT-L-336 MetaSlot With Masking	55.4 -15.9%	49.8 -14.4%	75.3 -10.3%	**1161.6** -11.9%	42.0 -30.1%	51.9 -15.4%	38.6 -2.4%	**62.6** -6.3%	**34.0** -0.4%	8.4 +3.6%	49.3 +0.5%	23.6 -30.1%	10.2 -15.2%	2.7 -90.7%	40.0 -18.9%	41.6 -8.3%	**32.8** -10.6%	42.2 -12.1%	5.0 -61.0%	27.3 -30.8%	**56.7** +22.0%	51.0 -2.4%	29.1 -13.3%	51.7 -8.3%	58.2 +13.0%	55.5 -6.5%	60.2 -0.1%
CLIP-ViT-L-336 MetaSlot Without Masking	56.1 -14.9%	50.4 -13.4%	76.1 -9.3%	**1161.6** -11.9%	42.6 -29.1%	**53.2** -13.2%	38.4 -3.0%	61.9 -7.4%	33.6 -1.6%	8.7 +7.1%	49.3 +0.5%	23.7 -29.7%	10.1 -16.1%	2.8 -90.0%	40.0 -18.9%	42.1 -7.3%	32.6 -11.3%	42.7 -11.1%	3.8 -70.8%	27.6 -30.1%	56.3 +21.0%	**52.1** -0.3%	29.1 -13.3%	52.3 -7.2%	**61.0** +18.5%	55.0 -7.4%	**63.0** +4.4%
CLIP-ViT-L-336 CLS only	**56.5** -14.2%	49.8 -14.4%	72.7 -13.3%	**1161.6** -11.9%	**48.7** -19.0%	**53.2** -13.2%	**38.8** -1.8%	61.9 -7.4%	33.2 -2.5%	**9.3** +14.3%	**50.9** +3.6%	**25.6** -24.3%	**10.4** -13.5%	**8.5** -70.3%	**41.0** -16.8%	42.4 -6.5%	31.1 -15.4%	**43.2** -10.1%	2.5 -80.5%	24.0 -39.2%	54.5 +17.3%	50.6 -3.1%	**29.8** -11.2%	51.7 -8.3%	56.9 +10.5%	55.0 -7.4%	59.5 -1.3%
DINOv2-S MetaSlot With Masking	53.2 -19.2%	50.6 -13.1%	**78.6** -6.3%	1068.0 -19.0%	37.0 -38.4%	46.6 -24.0%	37.1 -6.1%	61.9 -7.3%	32.5 -4.7%	6.5 -19.8%	47.6 -3.1%	23.4 -30.8%	9.6 -20.0%	1.7 -94.0%	37.9 -23.1%	44.4 -2.2%	31.9 -13.0%	41.1 -14.4%	**6.5** -49.6%	**39.4** -0.3%	40.8 -12.3%	49.6 -5.0%	28.2 -16.1%	52.2 -7.4%	54.0 +4.9%	58.5 -1.5%	55.3 -8.3%
DINOv2-S MetaSlot Without Masking	53.9 -18.3%	**50.9** -12.5%	**78.6** -6.3%	1061.4 -19.5%	39.5 -34.3%	47.2 -23.0%	37.2 -6.0%	62.2 -6.9%	33.2 -2.6%	5.3 -34.6%	47.9 -2.4%	23.7 -30.0%	9.5 -20.8%	1.9 -93.3%	38.1 -22.7%	**45.1** -0.7%	31.7 -13.7%	41.3 -14.0%	5.9 -54.3%	38.3 -3.0%	41.3 -11.2%	49.0 -6.3%	28.3 -15.8%	**52.8** -6.4%	50.7 -1.6%	**60.0** +1.0%	53.0 -12.1%
LLaMA 3.2 1B only	38.5 -41.6%	39.9 -31.5%	48.0 -42.8%	764.7 -42.0%	24.3 -59.6%	42.0 -31.5%	35.0 -11.4%	59.3 -11.2%	33.2 -2.5%	8.7 +7.1%	38.8 -21.0%	21.3 -36.9%	9.3 -22.1%	2.3 -91.8%	36.2 -26.5%	37.4 -17.6%	28.0 -23.9%	38.5 -19.8%	0.0 -100.0%	22.9 -42.1%	50.5 +8.5%	45.3 -13.3%	26.0 -22.5%	46.7 -17.2%	54.6 +6.0%	50.0 -15.8%	49.2 -18.4%

As already mentioned, OCR and document understanding tasks are particularly challenging for aggregation techniques - due to their difficulty to preserve small details. When excluding them from the evaluation, the gap between the standard LLaVA architecture and others with aggregation techniques is overall reduced, demonstrating that 1) the aggregation techniques are more effective in learning how to distill the most relevant information for visual understanding and reasoning tasks; 2) the performance gap is mainly due to the fact that small details tends to be discarded during the aggregation process; 3) MLLMs do not really need all the visual

features provided by vision encoder - offering frequently redundant information, increasing only the computational cost - being in many cases sufficient to have few but good visual features to perform downstream tasks.

Despite the comparable performance at this point, this approach has a training overhead with respect to the standard LLaVA architecture with only CLS token, which could be considered a good enough way to provide visual features to the LLM when the computational budget is limited, providing a sort of summary of the visual information that is effective for the LLM to perform downstream tasks - without the need for an additional training phase for the aggregator, which can be computationally expensive and time-consuming.

4.1.4 Multi-Encoder

Described in Section 3.2.1, the multi-encoder approach is based on the idea of using multiple vision encoders or multiple layers of the same vision encoder to extract different visual features, and then combine them using the aggregator module, before passing them to the decoder. Multi-encoder which combines DINOv2 (Small) and CLIP (ViT-Large-336) vision encoders with MetaSlot aggregator is presented in Table 4.5 with LLaMA 3.2 1B (Figure 4.3).

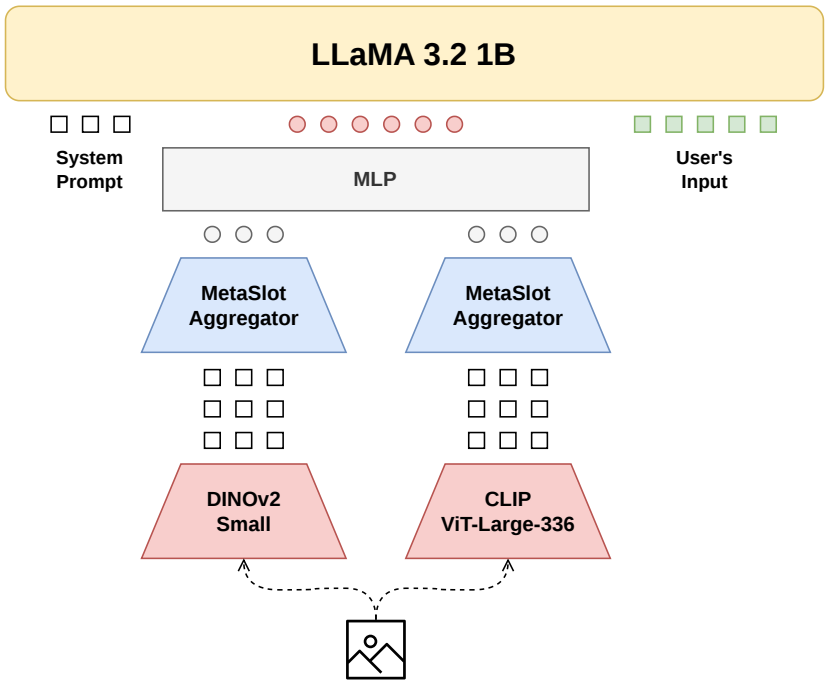


Figure 4.3: Multi-Encoder architecture with DINOv2 (Small) and CLIP (ViT-Large-336) vision encoders.

Multi-encoder in the presented configuration is able to outperform the standard LLaVA architecture with only CLS token, demonstrating that the combination of DINO and CLIP vision encoders can provide a more complete representation of the visual information, leading to a better performance in downstream tasks with respect to the use of a single vision encoder, given that the combination of different vision encoders can provide complementary information about the visual content.

Nevertheless, considering computational cost, the multi-encoder approach requires training multiple vision aggregators and if there are not enough resources to parallelize the pipeline during inference, it can lead to an increment of the inference time compared to single aggregator approach. In conclusion, the multi-encoder methodology is a valid solution to improve the performance of the model when multiple aggregators were already trained - and there are enough resources at inference time - thanks to its capability to provide a more complete representation of the visual information.

Table 4.5: Comparison considering LLaVA LLaMA 3.2 1B Multi-encoder with CLIP-ViT-Large-336 and DINOv2-Small of proposed MetaSlot configurations with 7 slots (unmasked variant) for each vision encoder (14 slots overall) against standard and CLS-only variants. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						**Knowledge**					**OCR**					**Vision**					**Other**					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Multi-encoder	**60.5**	**52.7**	**76.1**	**1246.0**	**52.9**	**58.8**	**40.8**	**65.3**	34.3	**10.1**	**53.5**	**28.4**	**11.0**	**16.0**	**42.5**	**44.1**	32.0	40.8	3.8	25.9	**57.4**	50.3	**31.9**	**54.1**	55.8	49.0	60.9
DINO+CLIP	-8.1%	-9.5%	-9.3%	-5.5%	-12.0%	-4.1%	+3.3%	-2.2%	+0.6%	+24.7%	+9.0%	-16.0%	-8.3%	-43.9%	-13.8%	-2.9%	-12.9%	-15.0%	-70.5%	-34.4%	+23.4%	-3.7%	-5.1%	-4.1%	+8.3%	-17.5%	+1.0%
CLIP-ViT-L-336	58.2	50.4	72.7	1232.8	48.7	57.4	40.3	64.6	**34.7**	9.3	52.5	25.7	10.7	8.5	41.5	42.0	**32.6**	**42.2**	**4.4**	**28.2**	55.6	**52.4**	30.2	52.3	**59.3**	**55.0**	**65.0**
CLS only	-11.7%	-13.4%	-13.3%	-6.5%	-19.0%	-6.4%	+1.9%	-3.3%	+1.8%	+14.8%	+6.9%	-24.0%	-10.8%	-70.2%	-15.8%	-7.5%	-11.2%	-12.1%	-65.9%	-28.6%	+19.6%	+0.2%	-10.1%	-7.3%	+15.1%	-7.4%	+7.8%

4.1.5 Training Instability

Results provided in previous sections demonstrate the effectiveness of Slot Attention-based aggregators to reduce the number of visual features, distilling the most relevant information into dense visual concepts - generally related to objects. However, despite the surprisingly aggregation capability and other benefits described in Section 3.2 - such as the possibility to train the aggregator once and use it for all LLMs thanks to its agnostic nature - this approach demonstrates some training instability issues during experiments which must be addressed

carefully in order to reach good performance.

Training instability mainly derive from the nature of the Slot Attention (Section 3.1.4) mechanism: 1) it is based on an iterative refinement process; 2) it exploits a softmax function to explicitly force the association of distinct slot with specific regions of the image; 3) it randomly initializes the slots.

These characteristics make Slot Attention very sensitive to model precision - preferring higher precision formats, which require more resources - and due to the initialization of the slots jointly to the iterative softmax competition, it is prone to the problem of dead slots (described in Section 3.1.4).

The most effective found solutions to mitigate the training instability were 1) to use a higher precision format - up to full precision - for the Slot Attention module; 2) to use a warmup phase during training, in which the learning rate is gradually increased from a small value to the target value, which can help to mitigate the problem of dead slots by allowing the model to explore how to use all the slots, avoiding a premature slot collapse due to higher learning rates; 3) to use small learning rates - such as $5 \cdot 10^{-5}$; 4) to use *gradient clipping* to limit the magnitude of the gradients and preventing the model from diverging during training. However, higher precision format leads to an increment of the training time, therefore it is recommended to try to mitigate the training instability issues with the other two solutions before using higher precision format (Figure 4.4)).

Figure 4.4: MetaSlot different training curves from left to right: 1) with training instability issues; 2) applying gradient clipping; 3) using a warmup phase during training and a smaller learning rate ($5 \cdot 10^{-5}$).

4.1.6 Pure Slot Attention

In previous sections, the Slot Attention-based aggregator provided by MetaSlot was considered, in order to evaluate the effectiveness of Slot Attention-based aggregators and to understand if the provided masking mechanism is effective in learning to distill the most relevant information into the dense visual concepts.

Even if in OCL scenarios dropping duplicated slots improves performance, empirical results on typical MLLM benchmarks (Section 4.1.2) demonstrate that masking mechanism does not help; on the contrary, they degrade performance in downstream tasks.

For this reason, a pure Slot Attention aggregator was considered to replace MetaSlot. As discussed above, this decision is motivated by the fact that masking mechanism is pointless in MLLM scenarios and simplifying the architecture training instability issues would be mitigated - this allows the use of a higher number of slots (e.g., 8, 16 or 32) or increase the learning rate (e.g., $4 \cdot 10^{-4}$), which can lead to less training time, reaching convergence faster.

In particular, pure Slot Attention aggregator is considered, trained for 35,000 steps (enough to reach convergence) with a batch size of 32 and a learning rate of $4 \cdot 10^{-4}$, based on CLIP (ViT-Large-336) with 8 slots - which is a more standard choice with respect to 7 proposed by MetaSlot - and then trained using LLaMA 3.2 1B in LLaVA pipeline. In the same way of MetaSlot, the number of refinement iterations is set to 3 - which is the de facto standard in most of the OCL related works that leverage Slot Attention - and the slot dimension is set equal to CLIP output dimension (i.e., 1024). Both MLP before the Slot Attention and final Spatial Broadcast Decoder has two hidden layers with dimension 1024, while the feed-forward network in the inner Slot Attention module has a dimension of 4096. The training objective is based on the MSE loss over visual features extracted by vision encoder (the same used in DINOSAUR [40]).

The only difference with respect to standard Slot Attention mechanism is the slot initialization: instead of initializing the slots randomly from a learnable Gaussian distribution, the slots are learnable vectors. This design choice is motivated by the fact that the random initialization of the slots leads to permutation invariant property, which is a desirable property in OCL scenarios, but it is not necessary in MLLM tasks - where the ordering is important for spatial reasoning.

As illustrated in Table 4.6, the pipeline described above - which retains just 1.4% of the original visual features - is able to achieve surprisingly good performance given the significant reduction of the visual features - especially in visual question answering tasks - outperforming CLIP's CLS only variant and obtaining comparable results to the standard LLaVA architecture

based on DINOv2 Small - which uses all the visual features provided by the vision encoder. This result is particularly interesting because it confirms again that the majority of the visual information provided by the vision encoder is not considered by the LLM to perform downstream tasks - which are really needed only when small details are important, such as in OCR and document understanding tasks - and only few dense visual concepts are enough to reach good enough performance in most scenarios, **outperforming** the baseline some benchmarks.

Table 4.6: Comparison between standard LLaVA and the proposed model with pure Slot Attention aggregator (8 slots, 3 iterations, MSE loss for 35,000 steps). Both with LLaMA 3.2 1B and CLIP-Vit-Large-336. Values within 10% gap are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA (CLIP-ViT-L-336)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Pure Slot Attention	58.6	52.7	79.4	1126.0	51.1	53.4	**38.6**	**65.6**	**35.0**	6.1	**47.9**	26.3	4.2	**26.6**	39.1	35.3	32.7	42.0	5.2	**37.0**	**46.5**	**52.9**	29.8	**55.7**	**55.3**	**58.9**	**64.8**
	-11.1%	-9.4%	-5.4%	-14.6%	-14.9%	-12.9%	-2.2%	-1.8%	+2.7%	-25.0%	-2.5%	-22.2%	-64.9%	-6.8%	-20.7%	-22.2%	-11.0%	-12.5%	-60.0%	-6.3%	0.0%	+1.3%	-11.4%	-1.2%	+7.4%	-0.8%	+7.5%

Ablation Study: MSE vs Cosine Similarity

As already mentioned, the training objective of the traditional Slot Attention-based approaches is based on the MSE loss, which is a common choice for regression tasks and originally employed in OCL models such as DINOSAUR [40] - but it is not the only one.

CLIP vision encoder was trained with a contrastive loss based on the cosine similarity, so it is reasonable to think that using a training objective based on the cosine similarity could be more effective in learning to distill the most relevant information into the dense visual concept when CLIP vision encoder is used.

Cosine similarity $\frac{a \cdot b}{||a|| \, ||b||}$ is a measure of similarity between two non-zero vectors a, b, also defined as the cosine of the angle between them. This loss function is particularly effective in scenarios where the magnitude of the vectors is not as important as their direction, which is often the case in high-dimensional spaces such as those produced by vision encoders.

Table 4.7 presents a comparison between the MSE loss and the cosine similarity loss, using a pure Slot Attention aggregator DINOSAUR-based [40] with a CLIP-ViT-Large-336 vision encoder. This includes an MLP projector with two hidden layers that match the visual feature

dimension of 1024. The inner Slot Attention module features a feed-forward network with a dimension of 4096, applying 3 iterations to extract 8 slots. During the aggregator pre-training phase, a Spatial Broadcast Decoder equipped with a two-layer MLP (dimension 1024) reconstructs the input visual features. The objective loss - either MSE or cosine similarity - is computed between original and reconstructed features in the vision encoder's output space. Both variants are trained for 35,000 steps (enough to reach convergence) during pre-training with a batch size of 32 and a learning rate of $4 \cdot 10^{-4}$.

There is no single winner for all benchmarks, but it is possible to notice that the MSE loss has the better performance overall, outperforming the cosine similarity loss in most benchmarks.

This could be explained by the fact that MSE loss is more effective in learning to distill the most relevant information into the dense visual concepts, given that it is a regression loss that directly optimizes the distance between the original and reconstructed features, while cosine similarity loss is a contrastive loss that optimizes the angle between the original and reconstructed features, which could not be as effective in learning to aggregate the most relevant information.

Table 4.7: Comparison between standard CLIP ViT-Large reference, Slot Attention with MSE loss, and Slot Attention with Cosine Similarity loss. Both slot models are trained for 35,000 steps with 8 slots and 3 iterations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA (CLIP-ViT-L-336)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Pure Slot Attention MSE	58.6	52.7	**79.4**	1126.0	51.1	**53.4**	**38.6**	**65.6**	**35.0**	**6.1**	47.9	**23.8**	**10.6**	**10.0**	39.1	35.3	**32.7**	42.0	5.2	**37.0**	46.5	**53.2**	**31.3**	**55.7**	**55.3**	58.9	**64.8**
	-11.1%	-9.4%	-5.4%	-14.6%	-14.9%	-12.9%	-2.2%	-1.8%	+2.7%	-25.0%	-2.5%	-29.7%	-11.4%	-64.9%	-20.7%	-22.2%	-11.0%	-12.5%	-60.0%	-6.3%	0.0%	+1.9%	-6.8%	-1.2%	+7.4%	-0.8%	+7.5%
Pure Slot Attention Cosine Similarity	**58.6**	**53.1**	78.1	**1135.2**	**51.9**	53.0	37.8	64.3	33.4	4.9	**48.5**	23.6	10.4	8.5	**39.6**	**36.0**	32.7	**43.0**	4.5	**37.5**	45.8	52.8	**32.4**	54.8	55.0	**59.5**	62.5
	-11.1%	-8.8%	-6.9%	-13.9%	-13.6%	-13.5%	-4.4%	-3.8%	-2.0%	-39.3%	-1.2%	-30.1%	-13.5%	-70.3%	-19.6%	-20.8%	-10.9%	-10.4%	-65.0%	-5.0%	-1.4%	+1.1%	-3.5%	-2.9%	+6.7%	+0.1%	+3.7%

Ablation Study: More Parameters

Additional ablation study was conducted to understand if increasing the number of weights across all components can lead to a better performance, given that they can provide a more powerful transformation of the visual features before and after the Slot Attention module, which can help to learn to aggregate the most relevant information into the dense visual concepts.

Reference configuration is the one described in Section 4.1.6, in proposed ablative model the MLP before the Slot Attention and the final Spatial Broadcast Decoder have three layers: input

layer with 1024 to be compatible with CLIP output dimension and hidden layers with dimension 2048, while the feed-forward network in the inner Slot Attention module has double dimension - i.e., 8192. The training objective is based on the MSE loss, which is the best performing one based on previous ablation study. Even in this case, both variants are trained for 35,000 steps (enough to reach convergence) during pre-training with a batch size of 32 and a learning rate of $4 \cdot 10^{-4}$.

In the results presented in Table 4.8 with LLaMA 3.2 1B, increasing the number of parameters allows the model to achieve better performance in some benchmarks - but not in all of them - specifically in visual question answering, while both achieve comparable performance in purely vision and general benchmarks. Instead, in OCR and document understanding benchmarks, the model with more parameters performs worse than the reference one, losing some percentage points.

Performance degradation in benchmark that require to understand small details in the image - such as OCR and document understanding tasks - could be explained by the fact that increasing the number of parameters can lead to a more powerful transformation of the visual features before and after the Slot Attention module, which can help to learn most relevant dense visual information, but it can also lead to a loss of the small details in the image, given that the model can learn to focus on the most relevant features for downstream tasks, which are often related to main subjects in the image, discarding small details that could be important for OCR and document understanding tasks.

Table 4.8: Comparison between standard CLIP ViT-Large reference, standard Slot Attention, and a larger Slot Attention aggregator variant. Both slot models are trained for 35,000 steps with 8 slots and 3 iterations using MSE loss. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA (CLIP-ViT-L-336)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Slot Attention	**58.6** -11.1%	52.7 -9.4%	**79.4** -5.4%	**1126.0** -14.6%	51.1 -14.9%	**53.4** -12.9%	**38.6** -2.2%	**65.6** -1.8%	**35.0** +2.7%	**6.1** -25.0%	47.9 -2.5%	23.8 -29.7%	**10.6** -11.4%	**10.0** -64.9%	**39.1** -20.7%	35.3 -22.2%	32.7 -11.0%	**42.0** -12.5%	**5.2** -60.0%	**37.0** -6.3%	46.5 0.0%	53.2 +1.9%	**31.3** -6.8%	**55.7** -1.2%	55.3 +7.4%	58.9 -0.8%	**64.8** +7.5%
Slot Attention Larger	58.1 -11.8%	**52.8** -9.3%	77.9 -7.1%	1106.2 -16.1%	**51.9** -13.6%	52.8 -13.9%	37.8 -4.2%	64.1 -4.1%	34.0 -0.3%	5.1 -37.5%	**48.1** -2.0%	**24.4** -27.7%	9.8 -18.0%	7.0 -75.3%	38.3 -22.3%	**42.6** -6.1%	**33.0** -10.1%	41.8 -13.0%	**5.2** -60.0%	36.5 -7.5%	**48.5** +4.2%	**53.7** +2.9%	31.0 -7.6%	55.5 -1.6%	**57.2** +11.0%	**61.7** +3.9%	63.3 +4.9%

Ablation Study: Spatial Sorting

Slot Attention produces slots that are permutation invariant [10], so there is not a specific intrinsic order in which the produced slots are arranged, and they can be associated with different regions of the image in different iterations of the refinement process.

In literature, architecture such as CORE[58] (Section 2.9) leverages a spatial sorting mechanism to sort the produced slots based on their spatial position in the image, in order to help the model to better understand the spatial relationships between the produced slots.

In order to understand if a spatial sorting mechanism can help LLM to increase the performance in downstream tasks, the CORE's centroid-guided sorting mechanism is implemented and applied to the produced slots of the pure Slot Attention-based aggregator described in Section 4.1.6, and then trained with CLIP-ViT-Large-336 and LLaMA 3.2 1B in LLaVA pipeline with same hyperparameters and training objective.

Results are presented in Table 4.9, showing that the spatial sorting mechanism does not significantly improve the performance of the model. This could be explained by two factors. First, the LLM can exploit spatial relationships through the information intrinsically present in the visual features; for instance, ViTs append a positional encoding to these features before the Transformer layers. Second, the learnable slots - unlike randomly initialized ones - provide a consistent positional anchoring. Since these slots maintain the same order during training, the model can learn to associate specific slots with specific image regions.

Table 4.9: Comparison between standard CLIP ViT-Large reference, standard Slot Attention, and Slot Attention with spatial sorting. All slot configurations use 8 slots, 3 iterations, and are trained with MSE loss for 35,000 steps.

	General						**Knowledge**					**OCR**					**Vision**					**Other**					
Model	**Avg**	GQA	POPE	MME	MMB $_{(EN)}$	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA (CLIP-ViT-L-336)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Slot Attention	58.6	52.7	79.4	1126.0	51.1	**53.4**	**38.6**	**65.6**	35.0	**6.1**	**47.9**	**23.8**	10.6	**10.0**	**39.1**	**35.3**	32.7	42.0	5.2	**37.0**	46.5	53.2	**31.3**	55.7	**55.3**	58.9	64.8
	-11.1%	-9.4%	-5.4%	-14.6%	-14.9%	-12.9%	-2.2%	-1.8%	+2.7%	-25.0%	-2.5%	-29.7%	-11.4%	-64.9%	-20.7%	-22.2%	-11.0%	-12.5%	-60.0%	-6.3%	0.0%	+1.9%	-6.8%	-1.2%	+7.4%	-0.8%	+7.5%
Slot Attention + Spatial Sorting	**59.2**	**53.7**	**80.4**	**1139.2**	**52.2**	53.0	38.6	65.5	**35.1**	6.0	47.7	23.6	**10.7**	9.7	38.9	34.9	**32.9**	**42.4**	**5.3**	36.9	**47.2**	**53.5**	31.2	**56.1**	54.8	**59.8**	**65.5**
	-10.1%	-7.8%	-4.2%	-13.6%	-13.2%	-13.6%	-2.4%	-2.0%	+3.0%	-25.9%	-2.8%	-30.3%	-10.9%	-65.8%	-21.0%	-23.1%	-10.3%	-11.6%	-59.2%	-6.6%	+1.4%	+2.3%	-7.2%	-0.6%	+6.4%	+0.6%	+8.6%

4.1.7 Self Slot Attention

By definition the Slot Attention mechanism treats slots independently, so there is not any interaction between them during the refinement process and they compete actively to be associated

with specific regions of the image. Given the concrete possibility to have dead slots - specifically when the number of slots is increased - Self-Slot Attention (described in Section 3.1.4) is considered as replacement of the pure Slot Attention aggregator in the configuration used in Section 4.1.6.

Given that Self-Slot Attention introduces an interaction between slots, those that have larger magnitude could highly influence other ones, homogenizing the produced slots and leading to a loss of performance. For this reason, in this case a cosine similarity loss - already proposed in Section 4.1.6 as MSE replacement - is used as training objective, given that it ignores the magnitude, considering only the direction of the vectors.

LLaMA 3.2 1B is trained with the Self-Slot Attention-based aggregator which was pre-trained with a batch size of 32, a learning rate of $4 \cdot 10^{-4}$ and for 35,000 steps with a CLIP-ViT-Large-336 vision encoder and 8 slots (with a dimensionality of 1024), and then compared with the pure Slot Attention-based aggregator with the same configuration and training objective (MSE). Both rely on the same architecture of the MLP before the Slot Attention and final Spatial Broadcast Decoder, which have two hidden layers with dimension 1024, while the feed-forward network in the inner Slot Attention module has a dimension of 4096.

Table 4.10: Comparison between standard CLIP ViT-Large reference, standard Slot Attention, and Self-Slot Attention aggregators. Both models are evaluated with 8 slots, 3 iterations, and MSE loss for 35,000 steps. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA (CLIP-ViT-L-336)	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Pure Slot Attention	58.6	52.7	79.4	1126.0	51.1	53.4	**38.6**	**65.6**	35.0	**6.1**	**47.9**	**23.8**	**10.6**	**10.0**	**39.1**	35.3	32.7	42.0	5.2	37.0	46.5	53.2	31.3	55.7	55.3	58.9	64.8
	-11.1%	-9.4%	-5.4%	-14.6%	-14.9%	-12.9%	-2.2%	-1.8%	+2.7%	-25.0%	-2.5%	-29.7%	-11.4%	-64.9%	-20.7%	-22.2%	-11.0%	-12.5%	-60.0%	-6.3%	0.0%	+1.9%	-6.8%	-1.2%	+7.4%	-0.8%	+7.5%
Self-Slot Attention	**59.8**	**53.8**	**80.4**	**1157.6**	**52.3**	**54.5**	38.2	64.7	34.9	5.9	47.5	23.5	10.4	9.5	37.9	**36.1**	**33.4**	**42.5**	**5.4**	**37.9**	47.8	54.2	**31.9**	56.5	56.4	60.6	65.5
	-9.3%	-7.5%	-4.2%	-12.2%	-12.9%	-11.1%	-3.3%	-3.2%	+2.2%	-27.3%	-3.2%	-30.5%	-13.7%	-66.5%	-23.1%	-20.4%	-9.0%	-11.4%	-57.8%	-4.1%	+2.9%	+3.8%	-5.2%	+0.2%	+9.6%	+2.1%	+8.7%

Results are presented in Table 4.10. Self-Slot Attention aggregator outperforms the pure Slot Attention one in visual question answering and general benchmarks, while it performs worse in OCR and document understanding tasks, demonstrating that allowing interaction between slots during the refinement process can help to learn to distill the most relevant information into the dense visual concepts, which can be effective in visual question answering and general

benchmarks, but it can also lead to a loss of information about small details in the image, given that the model can learn to focus on the most relevant features for downstream tasks, which are often related to main subjects in the image, discarding small details that could be important for OCR and document understanding tasks.

This architecture overall provides an improvement with respect to the pure Slot Attention-based aggregator, paving the way for aggregation techniques that are focused on VQA tasks, demonstrating that allowing interaction between slots during the refinement process can be effective in learning to distill the most relevant information into the dense visual concepts.

4.1.8 More Slots and Iterations

As already mentioned, 8 slots represent only a very small fraction - approximately 1.4% - of the original visual features provided by the vision encoder, so it is reasonable to think that increasing the number of slots can lead to a better performance, given that more visual information can be preserved during the aggregation process, which can help to learn to distill the most relevant information into the dense visual concepts.

Nevertheless, when the number of slots is increased, the hypothesis to associate each slot with a specific object in the image is lost - given that the higher number of slots leads to a higher probability to have multiple slots associated with the same object - and in this case the Slot Attention aggregator becomes just another attention-based mechanism to reduce the number of visual features (similarly to a perceiver-based aggregator).

This is particularly problematic in OCL scenarios, where the main task is to learn in a self-supervised way to identify and segment objects in the image. However, MLLM scenarios are different, given that the main task is to learn to aggregate the most relevant information into dense visual concepts, which can be effectively used by the LLM to perform downstream tasks, so the loss of the association between slots and objects might not be a problem. This intuition is supported by the fact that when MetaSlot was considered (Section 4.1.3) the higher performance on MLLM benchmarks was obtained by the variant without the masking mechanism, which allows multiple slots to be associated with the same object, while the variant with the masking mechanism - forcing a strict association between a slot and an object - performs worse in MLLM benchmarks, even if it is more effective in OCL benchmarks.

In order to explore this direction, the first experiments rely on a pure Slot Attention-based aggregator with 32 slots - only 5.5% of the original visual features provided by the vision encoder

- which is trained considering again the traditional OCL hyperparameters, such as 3 iterations and MSE loss. MLP before the Slot Attention and final Spatial Broadcast Decoder have two hidden layers with dimension 1024, while the feed-forward network in the inner Slot Attention module has a dimension of 4096. During pre-training both variants are trained with a batch size of 32 and a learning rate of $4 \cdot 10^{-4}$; however, the variant with 32 slots was trained for only 30,000 steps, given that it reaches convergence faster with respect to the variant with 8 slots, which was trained for 35,000 steps. As already done in previous setups, pre-trained aggregators are then trained with LLaMA 3.2 1B and compared with the standard LLaVA architecture (576 visual features).

Table 4.11: Comparison between standard CLIP ViT-Large reference and the updated slot attention variants. Gap percentages and absolute values have been recalculated proportionally to the CLIP ViT-Large-336 baseline. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Slot Attention	58.6	52.7	79.4	1126.0	51.1	53.4	38.6	65.5	**35.0**	**6.1**	47.9	23.8	**10.6**	10.0	39.1	35.3	**32.7**	42.0	**5.2**	**37.0**	**46.5**	53.2	31.3	**55.7**	55.3	58.9	64.8
8 Slots, 3 Iters	-11.1%	-9.4%	-5.4%	-14.6%	-14.9%	-12.9%	-2.3%	-1.9%	+2.7%	-25.0%	-2.5%	-29.7%	-11.4%	-64.9%	-20.7%	-22.2%	-11.0%	-12.5%	-60.0%	-6.3%	0.0%	+1.8%	-6.8%	-1.2%	+7.4%	-0.8%	+7.5%
Slot Attention	**60.4**	**54.8**	**81.6**	**1148.4**	**53.4**	**54.8**	**38.7**	**65.9**	34.8	5.8	**48.2**	**26.7**	10.5	**13.9**	**40.5**	**41.9**	29.7	**42.6**	4.5	33.6	38.1	**54.3**	**31.4**	54.7	**59.5**	**60.6**	**65.4**
32 Slots, 3 Iters	-8.3%	-5.8%	-2.7%	-12.9%	-11.2%	-10.6%	-2.2%	-1.3%	+2.0%	-28.6%	-1.8%	-21.0%	-12.1%	-51.3%	-17.9%	-7.8%	-19.1%	-11.2%	-65.0%	-15.0%	-18.1%	+4.0%	-6.6%	-3.1%	+15.5%	+2.1%	+8.4%

The quantitative results of this comparative analysis are presented in Table 4.11. As the data indicates, the pure Slot Attention-based aggregator configured with 32 slots consistently outperforms the baseline 8-slot variant across all evaluated benchmarks. Notably, this significant performance gain is achieved through a relatively moderate architectural adjustment - simply increasing the slot capacity - demonstrating the high impact of this specific hyperparameter. The consistent improvement across diverse tasks suggests that increasing the number of slots directly correlates with enhanced representational power. From a mechanistic perspective, a higher number of slots provides a broader informational bandwidth, ensuring that a greater amount of fine-grained visual information is preserved during the initial aggregation phase. By reducing the constraints of the informational bottleneck, the model is less prone to discarding subtle yet critical visual details. Consequently, this expanded capacity facilitates a more effective distillation process, allowing the architecture to optimally encode the most relevant features into highly informative and dense visual concepts.

Following the modification of the slot number, further experiments are conducted to evaluate the stability of the remaining hyperparameters. Specifically, the objective is to determine whether the standard configurations typically adopted in OCL scenarios remain optimal for this expanded architecture, or if recalibration is necessary to fully leverage the higher number of slots.

A primary hyperparameter of interest in this context is the number of Slot Attention iterations, which dictates the depth of the iterative refinement process. By increasing the iteration count, the slots are afforded additional computational steps to dynamically update their representations. This extended refinement phase theoretically enables the mechanism to better extract and isolate subtle, fine-grained details from the original visual features.

The empirical findings, illustrated in Table 4.12, reveal a striking improvement. Scaling the number of iterations from the standard 3 to 16 yields a remarkable performance boost. The 16-iteration model not only decisively outperforms the 3-iteration baseline across every single evaluated benchmark, but it also achieves a highly significant milestone: it drastically narrows the performance gap with the computationally heavy standard LLaVA architecture, which relies on an extensive set of 576 CLIP visual features. Most remarkably, on some tasks, this refined Slot Attention variant **outperforms** the standard LLaVA baseline, demonstrating an exceptional capability to resolve complex visual queries despite its highly compressed visual representation.

Table 4.12: Comparison between standard CLIP ViT-Large reference and the updated slot attention variants. Gap percentages and absolute values have been recalculated proportionally to the CLIP ViT-Large-336 baseline. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Slot Attention	**60.4**	54.8	**81.6**	1148.4	53.4	54.8	**38.7**	**65.9**	**34.8**	5.8	**48.2**	26.7	**10.5**	13.9	40.5	**41.9**	29.7	42.6	4.5	33.6	38.1	54.3	31.4	54.7	59.5	60.6	65.4
32 Slots, 3 Iters	-8.4%	-5.8%	-2.7%	-12.9%	-11.2%	-10.6%	-2.2%	-1.3%	+2.0%	-28.6%	-1.8%	-21.0%	-12.1%	-51.3%	-17.9%	-7.8%	-19.1%	-11.2%	-65.0%	-15.0%	-18.1%	+4.0%	-6.6%	-3.1%	+15.5%	+2.1%	+8.4%
Slot Attention	**61.5**	**55.8**	81.5	**1188.0**	**55.1**	**55.5**	38.6	65.4	34.4	**6.9**	47.7	**26.8**	10.3	**14.8**	**40.9**	41.1	**32.8**	**43.4**	**9.7**	**38.0**	**40.0**	**56.3**	**32.9**	**55.1**	**61.0**	**63.6**	**68.7**
32 Slots, 16 Iters	-6.8%	-4.2%	-2.9%	-9.9%	-8.3%	-9.5%	-2.3%	-2.1%	+1.0%	-14.3%	-2.9%	-20.8%	-13.8%	-48.0%	-17.1%	-9.4%	-10.8%	-9.6%	-25.0%	-3.8%	-13.9%	+7.7%	-2.1%	-2.3%	+18.4%	+7.0%	+14.0%

From a computational perspective, it is important to emphasize that increasing the number of iterations does not introduce any additional trainable parameters. The primary drawback involves a moderate increase in inference latency and a higher memory footprint required to store intermediate activations. However, in light of the extraordinary performance gains achieved, this computational cost represents a highly favorable trade-off. Even with the extended iteration

count, the proposed configuration remains vastly more memory- and compute-efficient than processing the full set of 576 visual features required by the standard LLaVA baseline.

Longer Refinement Process Analysis

Given the significant performance improvement obtained by increasing the number of iterations, it is interesting to understand the inner model behavior when more iterations are used, in order to understand how the model is able to extract more fine-grained information from the original visual features and distill it into the dense visual concepts, which can then be effectively used by the LLM to perform downstream tasks.

First, the OCL capabilities of the 32-slot architecture are evaluated in Table 4.13, comparing the 3-iteration and 16-iteration variants. The objective is to determine whether an extended refinement process can mitigate the inherent difficulties of preserving strict object-slot associations when the total number of slots vastly exceeds the average number of objects present in a typical image.

The empirical comparison reveals a stark lack of improvement: both variants exhibit uniformly sub-optimal OCL performance, with no significant differences in their metrics. This poor baseline behavior is directly attributable to the severe structural mismatch between the high slot capacity (32) and the naturally lower object count in standard visual scenes - on average 7 objects for each image considering MS COCO dataset. Consequently, the surplus of slots inevitably forces the model into over-segmentation - splitting single entities across multiple slots - or dedicating capacity to uninformative background noise, which inherently breaks strict object-centric binding.

Crucially, these results demonstrate that merely increasing the number of iterations cannot overcome this fundamental architectural limitation. While the 16-iteration model has more computational steps to extract features, it simply refines this fragmented representation rather than consolidating it. Ultimately, the extended processing time is insufficient to recover proper OCL capabilities when the slot-to-object ratio remains overwhelmingly disproportionate.

Table 4.13: MetaSlot instance segmentation OCL metrics.

Model	Slot	Iteration	ARI	FG-ARI	mBO	mIoU
Slot Attention with CLIP-ViT-L-336	32	3	0.0414	0.1878	0.1832	0.1898
Slot Attention with CLIP-ViT-L-336	32	16	0.0407	0.1859	0.1834	0.1901

Additionally, Figure 4.5, detailing the activation dynamics of a 32-slot architecture across a 16-step refinement process, reveals a distinct multiphasic behavior. Initially, a rapid convergence is observed, wherein the number of active slots decreases precipitously from near maximum capacity at the initial state to a stabilized plateau of approximately 13 active slots by the fourth iteration, indicating a successful preliminary object-binding phase where the model effectively partitions the feature space into distinct, meaningful entities. A subsequent perturbation occurs around the sixth iteration, characterized by a temporary spike in slot activation, which suggests an underlying algorithmic struggle to maintain representational distinctiveness without yielding to localized feature fragmentation. Following this, the number of active slots rapidly ascends, ultimately culminating in complete capacity saturation by the twelfth iteration. This terminal phase, during which all 32 slots become persistently active, strongly signifies a phenomenon of profound over-segmentation.

Given the better performance of the 16-iteration model with respect to the 3-iteration one, it is reasonable to hypothesize that the first part is a sort of routing phase, in which the model decides the subset of slots which are specialized to extract information from specific objects in the image, then the selected slots are refined and extract more fine-grained information thanks to the higher number of iterations. In the last iterations the model has learnt to exploit unused slots to help the main slots to extract the most complex additional information which are not extracted by the main slots - such as unusual details in the regions. Then the LLM will be able to merge the information extracted by the main slots with the one extracted by the helper slots to perform better in downstream tasks, especially in visual question answering tasks, which require a deeper understanding of the image.

Furthermore, this phenomenon raises the suspicion that as the number of refinement iterations increases, the underlying dynamics may gradually shift toward a behavior more akin to standard cross-attention. In this extended iterative regime, the competitive bottleneck - which is theoretically designed to enforce strict, object-centric segregation - may become increasingly diluted.

Consequently, rather than maintaining isolated and static bindings to distinct entities, the slots could begin to alternate their focus across various regions of the image. This would allow them to function collectively to aggregate global spatial context, effectively blurring the line between discrete object representation and traditional dense attention mechanisms.

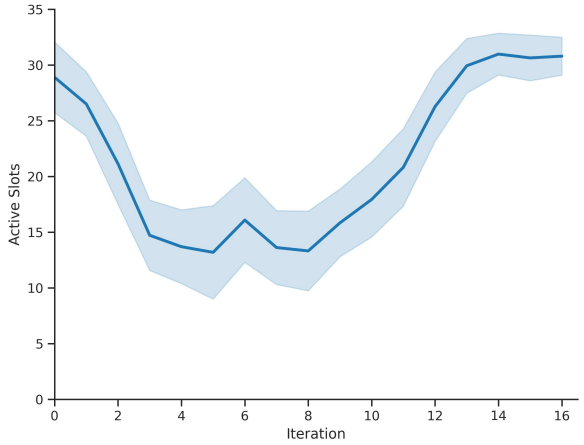


Figure 4.5: Mean number of active slots across iterations in a 32-slot architecture with 16 iterations over a batch of 32 random images from the MS COCO dataset [41] (the area around the strong line represents the standard deviation).

Heuristic Right Number of Iterations

To systematically investigate the broader implications of the refinement process on representational quality, a wider array of architectural configurations are evaluated. This involved strategically scaling the number of iterations and number of slots across several experiments.

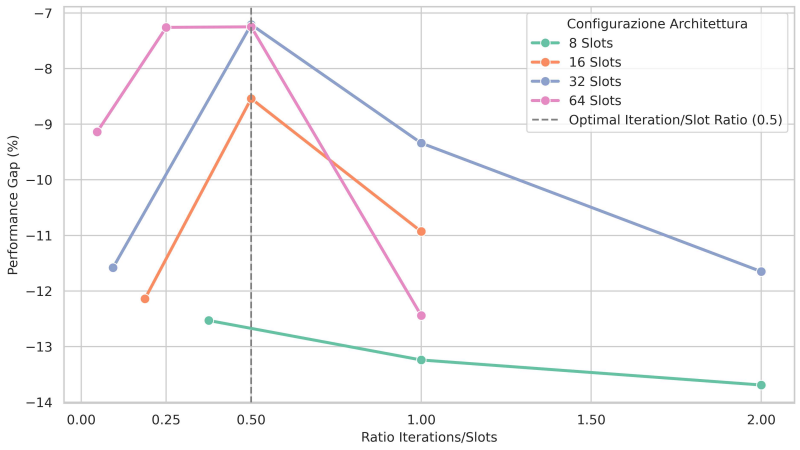


Figure 4.6: Slot Attention number of iterations trends across several configurations.

In particular, preserving the same hyperparameters of the model with 32 slots and 16 iterations described in Section 4.1.8, several variants are compared: 1) 8 slots with 3/8/16 iterations; 2) 16 slots with 3/8/16 iterations; 3) 32 slots with 3/16/32/64 iterations and 4) 64 slots with 3/16/32/64 iterations. The objective is to understand if the performance improvement obtained by increasing the number of iterations is consistent across different slot capacities, and

if there is a correlation between the number of slots and the number of iterations.

Table 4.14: Ablation study on the number of slots (S) and iterations (I) using the Slot Attention aggregator with CLIP-ViT-Large-336 and LLaMA 3.2 1B compared to the standard LLaVA (first row). Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined.

			General						Knowledge					OCR					Vision					Other					
S	I	TS ×1000	Avg	GQA	POPE	MME	MMB (EN)	SEED	Avg	SQA	MMMU	MathVista	AI2D	Avg	ChartQA	OCR_{bench}	TextVQA	VizWiz	Avg	RWQA	MMVP	Blink	V*	Avg	MMStar	QBench	ADE	Omni3D	COCO
-	-	-	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
8	3	35	**58.6**	**52.7**	**79.4**	**1126.0**	51.1	**53.4**	**38.6**	**65.6**	**35.0**	**6.1**	**47.9**	23.8	**10.6**	**10.0**	39.1	35.3	32.7	42.0	5.2	37.0	46.5	**53.2**	**31.3**	**55.7**	**55.3**	58.9	**64.8**
			-11.1%	-9.4%	-5.4%	-14.6%	-14.9%	-12.9%	-2.2%	-1.8%	+2.7%	-25.0%	-2.5%	-29.7%	-11.4%	-64.9%	-20.7%	-22.2%	-11.0%	-12.5%	-60.0%	-6.3%	0.0%	+1.9%	-6.8%	-1.2%	+7.4%	-0.8%	+7.5%
8	8	25	58.2	52.6	78.4	1123.4	51.2	52.7	37.8	64.9	33.2	5.7	47.5	23.8	10.5	9.1	39.2	36.2	33.0	42.1	5.4	37.3	47.2	52.9	31.1	55.0	54.8	59.4	64.0
			-11.6%	-9.6%	-6.5%	-14.8%	-14.8%	-14.0%	-4.3%	-2.8%	-2.5%	-30.2%	-3.3%	-29.7%	-12.4%	-67.9%	-20.5%	-20.2%	-10.2%	-12.3%	-58.2%	-5.5%	+1.4%	+1.2%	-7.4%	-2.4%	+6.5%	-0.0%	+6.1%
8	16	20	58.1	**52.7**	77.5	**1126.0**	**51.7**	52.1	36.9	64.4	30.8	5.1	47.2	**24.0**	10.4	8.1	**39.6**	**37.9**	**33.7**	**42.5**	**5.8**	**38.0**	**48.5**	52.7	31.0	54.4	54.4	**60.5**	63.1
			-11.9%	-9.5%	-7.6%	-14.6%	-14.0%	-15.0%	-6.7%	-3.6%	-9.7%	-37.5%	-3.8%	-29.0%	-13.2%	-71.7%	-19.6%	-16.5%	-8.2%	-11.4%	-55.0%	-3.8%	+4.2%	+0.9%	-7.6%	-3.5%	+5.7%	+1.8%	+4.7%
16	3	25	59.4	53.7	80.5	1136.5	52.2	54.1	38.6	65.7	34.9	5.9	48.0	25.2	10.6	11.9	39.7	38.5	31.2	42.3	4.8	35.3	42.3	53.7	31.3	55.2	57.4	59.8	65.1
			-9.8%	-7.8%	-4.1%	-13.8%	-13.1%	-11.8%	-2.2%	-1.6%	+2.3%	-26.9%	-2.2%	-25.5%	-11.8%	-58.2%	-19.4%	-15.1%	-15.1%	-11.9%	-62.6%	-10.7%	-9.1%	+2.9%	-6.8%	-2.2%	+11.4%	+0.6%	+7.9%
16	8	25	**61.9**	**55.5**	**83.1**	**1209.1**	**54.3**	**56.1**	**39.7**	**67.5**	**35.5**	**6.5**	**49.3**	**26.1**	**10.8**	**12.3**	**41.4**	**39.9**	**33.0**	**43.7**	6.0	37.8	**44.5**	**55.2**	**32.2**	**56.7**	**58.3**	**62.3**	**66.5**
			-6.1%	-4.6%	-1.0%	-8.3%	-9.7%	-8.5%	+0.4%	+1.0%	+4.1%	-20.3%	+0.4%	-22.8%	-9.8%	-57.0%	-16.0%	-12.1%	-10.2%	-9.0%	-53.3%	-4.3%	-4.4%	+5.6%	-4.1%	+0.5%	+13.3%	+4.8%	+10.2%
16	16	25	60.3	53.8	80.5	1203.8	52.7	54.4	38.4	65.2	34.1	**6.5**	47.6	25.0	10.3	10.9	40.1	38.6	32.6	42.2	**6.5**	**38.0**	43.9	53.5	31.1	54.8	56.2	61.2	64.2
			-8.4%	-7.5%	-4.1%	-8.7%	-12.3%	-11.2%	-2.9%	-2.4%	0.0%	-19.6%	-3.0%	-26.1%	-13.8%	-61.7%	-18.6%	-15.0%	-11.1%	-12.0%	-50.0%	-3.8%	-5.6%	+2.5%	-7.4%	-2.8%	+9.2%	+3.1%	+6.5%
32	3	30	60.4	54.8	**81.6**	1148.4	53.4	54.8	**38.7**	**65.9**	34.8	5.8	**48.2**	26.7	10.5	13.9	40.5	**41.9**	29.7	42.6	4.5	33.6	38.1	54.3	31.4	54.7	59.5	60.6	65.4
			-8.3%	-5.8%	-2.7%	-12.9%	-11.2%	-10.6%	-2.1%	-1.3%	+2.0%	-28.6%	-1.8%	-21.0%	-12.1%	-51.3%	-17.9%	-7.8%	-19.1%	-11.2%	-65.0%	-15.0%	-18.1%	+4.0%	-6.6%	-3.1%	+15.6%	+2.1%	+8.4%
32	16	30	61.4	**55.8**	81.5	**1188.0**	**55.1**	55.5	38.6	65.4	34.4	**6.9**	47.7	26.8	10.3	14.8	40.9	41.1	32.8	43.4	**9.7**	**38.0**	40.0	**56.3**	**32.9**	55.1	**61.0**	**63.6**	**68.7**
			-6.7%	-4.2%	-2.9%	-9.9%	-8.3%	-9.5%	-2.3%	-2.1%	+1.0%	-14.3%	-2.9%	-20.7%	-13.8%	-48.0%	-17.1%	-9.4%	-10.7%	-9.6%	-25.0%	-3.7%	-13.9%	+7.7%	-2.1%	-2.3%	+18.4%	+7.0%	+14.0%
32	32	30	**61.5**	55.2	**81.6**	1181.4	**55.1**	**56.8**	38.4	**65.9**	34.4	5.5	48.0	**27.3**	**10.8**	**16.1**	**41.6**	40.6	**33.7**	**44.5**	7.7	36.5	**45.8**	52.5	30.8	54.8	56.0	58.4	62.3
			-6.6%	-5.1%	-2.8%	-10.4%	-8.3%	-7.4%	-2.7%	-1.4%	+1.0%	-32.1%	-2.3%	-19.3%	-10.4%	-43.4%	-15.7%	-10.5%	-8.3%	-7.2%	-40.0%	-7.5%	-1.4%	+0.4%	-8.2%	-2.9%	+8.8%	-1.7%	+3.3%
32	64	30	59.3	52.8	80.4	1169.5	51.7	53.3	37.8	63.9	**35.1**	6.4	46.0	24.7	10.0	9.8	39.6	39.5	33.3	44.1	7.1	37.0	45.2	52.7	31.3	54.7	56.0	59.6	61.9
			-9.9%	-9.3%	-4.2%	-11.3%	-13.9%	-13.1%	-4.3%	-4.4%	+3.0%	-21.4%	-6.4%	-26.9%	-16.6%	-65.6%	-19.7%	-13.0%	-9.2%	-8.2%	-45.0%	-6.3%	-2.8%	+0.9%	-6.8%	-3.0%	+8.8%	+0.3%	+2.6%
64	3	50	62.0	55.6	82.1	1210.4	55.4	56.3	**39.4**	65.5	**35.6**	**8.0**	**48.5**	26.7	**10.9**	16.3	41.5	38.0	31.5	42.0	**7.1**	**38.0**	38.7	51.7	30.6	54.4	53.3	**60.9**	59.6
			-5.9%	-4.4%	-2.1%	-8.2%	-7.9%	-8.1%	-0.3%	-1.9%	+4.4%	-1.8%	-1.3%	-21.1%	-9.3%	-42.7%	-15.9%	-16.2%	-14.3%	-12.5%	-45.0%	-3.8%	-16.7%	-1.0%	-9.0%	-3.6%	+3.5%	+2.5%	-1.2%
64	16	50	**62.9**	**56.6**	**82.9**	**1227.5**	56.1	**57.5**	38.8	64.8	34.7	7.8	47.7	**27.4**	10.7	**17.6**	**42.1**	39.4	33.4	44.3	5.8	36.5	**47.2**	54.4	**32.5**	56.4	55.9	60.7	**66.4**
			-4.6%	-2.8%	-1.2%	-6.9%	-6.7%	-6.2%	-2.0%	-3.0%	+1.7%	-3.6%	-2.8%	-18.8%	-10.7%	-38.4%	-14.6%	-13.3%	-8.9%	-7.7%	-55.0%	-7.5%	+1.4%	+4.1%	-3.3%	0.0%	+8.5%	+2.2%	+10.1%
64	32	50	62.0	55.3	82.6	1164.2	**56.9**	56.9	39.2	**66.3**	34.2	7.8	**48.5**	27.3	**10.9**	17.4	41.3	**39.7**	**33.5**	**45.6**	6.5	36.0	45.8	**54.5**	31.6	**56.5**	**59.5**	59.6	65.4
			-5.9%	-4.9%	-1.6%	-11.7%	-5.4%	-7.2%	-0.8%	-0.7%	+0.3%	-3.5%	-1.3%	-19.2%	-9.0%	-39.1%	-16.3%	-12.5%	-8.9%	-5.1%	-50.0%	-8.8%	-1.4%	+4.4%	-5.9%	+0.2%	+15.5%	+0.4%	+8.4%
64	64	45	58.8	52.1	81.3	1108.9	51.8	53.3	38.7	65.0	35.2	6.8	47.9	24.0	10.5	8.4	39.5	37.6	31.7	43.2	6.5	34.0	43.3	52.6	31.4	54.1	57.7	56.0	63.8
			-10.8%	-10.4%	-3.1%	-15.9%	-13.8%	-13.1%	-2.0%	-2.7%	+3.3%	-16.1%	-2.5%	-29.0%	-12.8%	-70.6%	-19.8%	-17.2%	-13.6%	-10.1%	-50.0%	-13.8%	-6.9%	+0.7%	-6.6%	-4.0%	+12.0%	-5.7%	+5.8%

Results are presented in Table 4.14, showing an interesting correlation between the number of slots and the number of iterations: when the number of iterations and slots are not increased in tandem, there are no significant performance improvements. Specifically, when the number of iterations is increased a lot with respect to the number of slots (e.g., 8 slots with 16 iterations), there is a degradation of the performance. In the same way, if the number of slots is increased without increasing the number of iterations (e.g., 32 slots with 3 iterations), there is not a significant performance improvement.

Empirically, results in Figure 4.6 suggest that best performance is obtained when the number of iterations is around half of the number of slots (e.g., 16 slots with 8 iterations), which could be explained by the fact that when the number of iterations is increased in tandem with the

number of slots, the model is able to effectively leverage the higher slot capacity to extract more fine-grained information from the original visual features, while avoiding the over-segmentation problem that arises when the number of slots is increased without increasing the number of iterations.

This configuration allows us to find a good balance between preserving a sufficient amount of visual information and maintaining a strong object-centric bias, which can lead to a better performance in downstream tasks.

4.1.9 Comparison with OC-VTP

Given the remarkable performance obtained by the pure Slot Attention-based aggregator with 32 slots and 16 iterations, it is interesting to compare this configuration with one of its main competitors: the OC-VTP [11] architecture described in Section 2.8, which exploits Slot Attention to select relevant visual features, without fine-tuning the LLM after the pre-training of the aggregator, but requiring a full training of the standard LLaVA.

Table 4.15: Comparison between standard LLaVA with Vicuna-7B and CLIP-ViT-Large-336 (baseline), OC-VTP and proposed Slot Attention aggregator models across selected benchmarks. Two average across benchmarks are reported: one for all benchmarks and one for a subset without text-centric tasks. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

Model	**Avg**	**Avg$_{\text{subset}}$**	GQA	MMB (EN)	MME	POPE	SEED	SQA	TextVQA	VizWiz	MMMU
Standard LLaVA	65.6	66.8	64.9	64.6	1469.0	87.5	70.8	70.3	64.3	58.0	36.3
OC-VTP 64 Slots	62.7	63.6	58.4	59.7	1347.1	84.5	70.0	69.7	60.3	**58.6**	35.6
	-4.4%	-4.8%	-10.0%	-7.6%	-8.3%	-3.4%	-1.2%	-0.9%	-6.2%	+1.0%	-1.9%
OC-VTP 128 Slots	**64.2**	65.5	60.8	62.1	**1389.7**	**86.9**	73.6	69.8	**62.1**	56.8	**36.1**
	-2.1%	-1.9%	-6.3%	-3.9%	-5.4%	-0.7%	+4.0%	-0.7%	-3.4%	-2.0%	-0.6%
Slot Attention 64 Slots 32 Iters	63.6	**66.2**	**62.5**	**62.8**	1383.8	86.3	**75.7**	71.8	55.3	53.6	35.4
	-3.0%	-0.9%	-3.7%	-2.8%	-5.8%	-1.4%	+6.9%	+2.2%	-14.0%	-7.6%	-2.5%
Slot Attention 32 Slots 16 Iters	62.8	65.3	61.8	60.8	1385.3	84.4	73.6	**71.9**	54.7	54.2	35.1
	-4.2%	-2.4%	-4.8%	-5.9%	-5.7%	-3.5%	+3.9%	+2.3%	-15.0%	-6.6%	-3.4%

As illustrated in the comprehensive evaluation (Table 4.15) across multiple visual reasoning benchmarks proposed by the authors of OC-VTP themselves, the proposed purely Slot Attention-driven mechanism exhibits a highly efficient use of its representational capacity.

Specifically, the configuration utilizing merely 32 slots and 16 refinement iterations with the fine-tuning of Vicuna 7B demonstrates highly competitive performance metrics against the OC-VTP architecture employing double the number of slots (i.e., 64 slots).

This finding is particularly salient because it empirically validates that proposed architectural refinements successfully achieve equivalent representational power while effectively halving the requisite number of visual tokens - or slots - that must be processed by the language model - even considering larger LLMs.

A nuanced analysis of the performance breakdown across individual datasets reveals a specific vulnerability inherent to this highly compressed object-centric approach: tasks heavily reliant on OCR, such as *TextVQA* and *VizWiz*. Avoiding the selection of raw visual features, but aggregating all of them, in these specific domains, the proposed architecture faces challenges in encoding dense, fine-grained, and low-level textual artifacts into a strictly limited number of slots, leading to a degradation in the overall average score when these metrics are factored into the global evaluation.

Crucially, when these text-centric visual benchmarks are excluded from the evaluation the true efficacy of the proposed model for general visual understanding, spatial reasoning, and object-relational tasks becomes markedly apparent. Under these filtered conditions, the streamlined configuration of just 32 slots and 16 iterations yields an aggregated performance that is strikingly comparable to the significantly heavier baseline model equipped with 128 slots - that due to the quadratic attention complexity means more than sixteen times the computational cost.

This near-parity in performance on core visual reasoning tasks strongly underscores the efficiency of the proposed iterative refinement strategy. It suggests that for broad semantic understanding, a well-calibrated, lower-capacity Slot Attention module can effectively rival the representational fidelity of much larger, parameter-heavy aggregators, provided that dense textual reading is not the primary objective of the downstream task.

4.1.10 2D Pooling Aggregator

In previous sections, the pure Slot Attention-based aggregator was considered with several configurations - 8, 16, 32 and 64 slots - showing that it is able to achieve good performance

in downstream tasks with an extreme reduction of the visual features provided by the vision encoder. When the computational budget is enough to allow a higher number of slots, the benefits of the Slot Attention - which was originally designed to learn to identify only a small number of objects in the image - tend to vanish.

The 2D average pooling aggregator (Section 3.1.1) is considered as an alternative to Slot Attention-based one when the number of slots is increased - for example 64 - and its effectiveness was already exploited in previous works such as Gemma-4 [53]. Pooling can be effective in reducing the number of visual features, preserving a good amount of information when the number of retained visual tokens is enough. In fact, when a too aggressive pooling is applied - for example, to obtain a final resolution of only 2×2 or 4×4 patches - it could tend to amalgamate highly heterogeneous visual features, producing poorly informative visual concepts.

In order to evaluate the effectiveness of pooling-based aggregators with a higher number of visual tokens, an average pooling aggregator with a final resolution of 8×8 is considered against a pure Slot Attention aggregator with 64 slots and 32 iters. Both aggregators are trained with the same post vision encoder MLP layers and the same Spatial Broadcast Decoder - both with a 1024-dimensional input layer and two hidden layers with dimension 2048.

Both variants are trained with a MSE loss up to convergence - 50,000 training steps for the pooling-based variant and 25,000 for the Slot Attention-based variant - with a batch size of 32, a learning rate of 2×10^{-5} and a gradient clipping threshold of 1.0. After pre-training using CLIP (ViT-Large-336) vision encoder, both aggregators are trained with LLaMA 3.2 1B and compared with the standard LLaVA architecture (576 visual features).

Results in Table 4.16 show first of all that when the number of slots is increased to 64, the pooling-based aggregator is able to outperform the Slot Attention-based one in visual question answering and OCR/document understanding - maintaining comparable performance on other tasks - demonstrating that when the computational budget allows the use of a higher number of slots, the benefits of the Slot Attention tend to vanish and a simpler pooling-based approach can be more effective in learning to distill the most relevant information into the dense visual concept, which can then be effectively used by the LLM to perform downstream tasks. Moreover, surprisingly, the pooling-based aggregator with 64 visual tokens is able to **outperform** the standard LLaVA architecture in some tasks.

Additionally, having increased the number of visual tokens to 64 (approximately 11.11% of the total visual features), the gap with the standard LLaVA architecture is drastically reduced,

reaching an overall performance gap of only 5% considering also the text-centric visual benchmarks - which are the most challenging ones for aggregation approaches - and less than 3.5% when they are excluded from the evaluation, demonstrating that when the computational budget allows the use of a higher number of slots, the benefits of the Slot Attention tend to vanish and a simpler pooling-based approach can be more effective in learning to aggregate the most relevant information into the dense visual concepts, which can then be effectively used by the LLM to perform downstream tasks.

Table 4.16: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336 in different configurations to evaluate pooling strategies. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Pooling 8x8 MSE	61.8	54.6	78.7	**1236.7**	56.4	**57.5**	39.1	66.1	33.7	**8.0**	**48.6**	**28.7**	10.2	**24.2**	**41.9**	38.5	32.2	42.1	**11.3**	34.6	40.8	53.7	**34.6**	**58.0**	53.0	**61.1**	62.0
	-6.2%	-6.2%	-6.2%	-6.2%	-6.2%	-6.2%	-1.1%	-1.1%	-1.1%	-1.1%	-1.1%	-15.1%	-15.1%	-15.1%	-15.1%	-15.1%	-12.3%	-12.3%	-12.3%	-12.3%	-12.3%	+2.9%	+2.9%	+2.9%	+2.9%	+2.9%	+2.9%
Pooling 8x8 Cosine Similarity	60.9	53.8	77.6	1219.6	55.6	56.7	38.1	64.3	32.8	7.8	47.3	26.3	9.3	22.1	38.3	35.3	31.0	40.5	10.9	33.3	39.2	52.2	33.6	56.3	51.4	59.3	60.2
	-7.5%	-7.5%	-7.5%	-7.5%	-7.5%	-7.5%	-3.7%	-3.7%	-3.7%	-3.7%	-3.7%	-22.3%	-22.3%	-22.3%	-22.3%	-22.3%	-15.6%	-15.6%	-15.6%	-15.6%	-15.6%	-0.1%	-0.1%	-0.1%	-0.1%	-0.1%	-0.1%
Slot Attention 64 Slots, 32 Iters	**62.0**	**55.3**	**82.6**	1164.2	**56.9**	56.9	**39.2**	**66.3**	**34.2**	7.8	48.5	27.3	**10.9**	17.4	41.3	**39.7**	**33.5**	**45.6**	6.5	**36.0**	**45.8**	**54.5**	31.6	56.5	**59.5**	59.6	**65.4**
	-5.9%	-4.9%	-1.6%	-11.7%	-5.4%	-7.2%	-0.8%	-0.7%	+0.3%	-3.5%	-1.3%	-19.2%	-9.0%	-39.1%	-16.3%	-12.5%	-8.9%	-5.1%	-50.0%	-8.8%	-1.4%	+4.4%	-5.9%	+0.2%	+15.5%	+0.4%	+8.4%

These good results can be explainable by the fact that CLIP's visual features are text-aware and highly exploitable by the LLM in the LLaVA paradigm - which is the main reason of the impressive performance reached by LLaVA models. Pooling acts exactly as a feature *compressor* - therefore there are no ways to increase the performance of the model, unless the LLM can properly exploit all visual features. Given that pooling preserves the visual encoder output distribution, the LLM can effectively leverage the text-aware visual features provided by CLIP, even if they are significantly reduced in number, to perform downstream tasks, in the same way as standard LLaVA. Moreover, pooled features preserve the spatial order of the original visual features - while Slot Attention produces a set of permutation invariant slots [10] - which can help the LLM in spatial reasoning tasks.

Furthermore, from a training perspective, pooling-based aggregators are much easier to train with respect to Slot Attention-based ones, given that they do not have additional parameters, do not have the problem of dead slots - which can be a problem especially when the number of slots is increased - and they do not require a careful tuning of hyperparameters, such as the number of iterations, which can be crucial for Slot Attention-based aggregators to reach good

performance. This makes pooling-based aggregators more robust and easier to use in practice, especially when the computational budget allows the use of a higher number of slots.

Nevertheless, the good performance of pooling-based aggregator with 64 visual tokens presented above was obtained with a pre-training phase of 50,000 steps, which is much higher than the 25,000 steps used for the Slot Attention-based variant. However, thanks to its simplicity - there are no additional parameters and no iterative process - pooling-based aggregators are faster to train, requiring fewer computation resources. Additionally, thanks to its higher training stability, pooling-based aggregators can be trained with a higher learning rate, which can further reduce the training time, reaching the convergence faster. This makes pooling-based aggregators more efficient to train in practice when the computational budget allows the use of a higher number of slots.

4.2 Experiments on Text-influenced Vision-centric Aggregation

Described in Section 3.3, text-influenced vision-centric aggregation relies on the LLM to use cross-entropy loss to learn how to aggregate the visual features provided by the vision encoder. This approach is more appealing from a training perspective, given that it does not require a pre-training phase of the aggregator, but directly exploits the standard LLaVA training pipeline with the same training objective of the standard architecture (cross-entropy loss). Moreover, this approach can be more effective in learning to distill the most relevant information into the dense visual concepts, given that it directly optimizes the performance on downstream tasks, while the other approaches optimize a proxy objective - such as MSE or cosine similarity - which may not be perfectly aligned with the final performance on downstream tasks. Additionally, thanks to the visual features reduction obtained through the aggregator, this approach significantly reduces time and computational resources of the standard LLaVA training and inference phase significantly, which is a crucial aspect in practice, given that standard LLaVA might be unfeasible in resource-constrained environments such as real-time robotic applications and edge devices.

Results presented in Section 4.1.7 and Section 4.1.8 suggested that interaction between slots and more iterations help to obtain better performance in downstream tasks. On the other hand, Slot Attention has some training stability issues which can be amplified when the training signal comes from a language model - which provides a much noisier training signal compared to the

MSE or cosine similarity loss for visual features aggregation.

For this reason, in this section the more training-stable perceiver resampler and pooling-based aggregators are mainly considered. As already mentioned, LLaMA 3.2 1B is used as prober, while larger LLMs - such as Vicuna 7B - are used for a more realistic evaluation of the performance of the proposed approaches in downstream tasks. Additionally, Qwen3 family of models is considered for a more comprehensive evaluation with different LLMs.

4.2.1 LLaVA LLaMA 3.2 1B with Pooling-based Aggregator

Standard LLaVA architecture exploits all 576 visual tokens provided by the vision encoder. The simplest approach to reduce the number of visual tokens is to apply a pooling-based aggregator (as described in Section 3.1.1) to the original visual features, reducing them to a fixed number of visual tokens, such as 8, 16 or 64. This approach is more appealing from a training point of view, given that it does not have any additional parameter and it does not require a careful tuning of hyperparameters, such as the number of iterations in Slot Attention-based aggregators.

First experiments rely on a average pooling-based aggregator with a final resolution of 2×4 (8 visual tokens) - higher resolution is assigned to the horizontal dimension given that there are more horizontal images than vertical ones in the training dataset - which is plugged in the standard LLaVA architecture with a LLaMA 3.2 with 1B parameters and CLIP-ViT-Large-336.

As illustrated in Section 3.3 there are three different ways to plug an aggregator in the LLaVA pipeline: 1) before the MLP after the vision encoder, 2) after the MLP and vision decoder, and 3) between two MLPs. All these configurations are evaluated and compared in Table 4.17 where MLPs have a hidden state with the double of the dimensionality of the input - on the contrary, standard LLaVA uses a hidden dimensionality equal to the input dimensionality, but in this case more parameters were added to help the compression phase. The learning rate used for MLPs in all these experiments is $1 \cdot 10^{-3}$ with a global batch size of 256 in the first stage and 128 in the second one - exactly as specified in the original LLaVA training pipeline.

It is clear that the best performance is obtained when the pooling-based aggregator is plugged between two MLPs, beating other two variants in almost all benchmarks. This may be explained by the fact that the MLP before the aggregator can help to extract more relevant information from the original visual features, which can be then effectively aggregated by the pooling-based approach, while the MLP after the pooling-based aggregator can help to refine the aggregated visual features, aligning them to LLM language space.

It is interesting to notice that the variant with the MLP before the pooling is able to **outperform** the standard LLaVA architecture in visual question answering tasks - such as MMVP in which there is an improvement up to more than 60%. This could be explained by the fact that MLP before the pooling-based aggregator is able to extract the most relevant information, discarding noise and irrelevant features - which can be then effectively aggregated by the pooling-based approach - reducing the amount of visual information to be processed by the LLM, which can focus only on the most relevant visual concepts. This allows the LLM to perform better in downstream tasks, especially in visual question answering tasks, which require a deeper high-level understanding of the image instead of small details.

Table 4.17: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336 in different configurations to evaluate 2×4 pooling strategies. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Pre and Post MLPs	**57.0**	**47.5**	**71.8**	**1251.2**	**58.2**	**45.1**	**29.5**	**31.2**	**32.3**	**7.8**	**46.5**	23.2	**10.2**	8.9	34.3	39.2	**37.2**	**43.8**	**20.7**	**40.9**	**43.2**	**44.9**	**32.2**	**52.2**	**42.7**	**50.5**	**46.8**
Figure 3.7c	-13.5%	-18.4%	-14.4%	-5.1%	-3.2%	-26.4%	-25.3%	-53.3%	-5.3%	-3.7%	-5.3%	-31.4%	-15.0%	-68.8%	-30.4%	-13.7%	+1.4%	-8.8%	+60.5%	+3.5%	-7.1%	-14.0%	-4.2%	-7.4%	-17.1%	-15.0%	-22.4%
Post MLP	55.3	47.2	69.2	1207.2	57.1	42.7	29.1	31.1	31.9	7.3	46.0	**25.5**	9.6	**13.6**	**35.7**	**42.9**	34.6	41.0	16.0	39.5	41.9	43.8	31.7	51.0	42.0	50.3	43.9
Figure 3.7a	-16.1%	-18.9%	-17.5%	-8.4%	-5.0%	-30.3%	-26.3%	-53.4%	-6.5%	-9.9%	-6.3%	-24.6%	-20.0%	-52.3%	-27.6%	-5.5%	-5.7%	-14.6%	+24.0%	0.0%	-9.9%	-16.1%	-5.7%	-9.6%	-18.4%	-15.3%	-27.2%
Pre MLP	54.1	45.5	68.3	1163.7	57.3	41.4	28.8	29.6	31.8	7.5	46.1	21.4	9.7	2.9	33.1	39.9	33.6	40.1	11.3	40.0	43.1	42.4	31.8	50.5	39.2	49.2	41.2
Figure 3.7b	-17.9%	-21.8%	-18.6%	-11.7%	-4.7%	-32.5%	-27.1%	-55.7%	-6.7%	-7.4%	-6.1%	-36.7%	-19.2%	-89.8%	-32.9%	-12.1%	-8.4%	-16.5%	-12.4%	+1.3%	-7.3%	-18.8%	-5.4%	-10.5%	-23.9%	-17.2%	-31.7%

These preliminary results demonstrate that text-influenced vision-centric aggregation are effective in learning to distill the most relevant information into the dense visual concepts - reaching also better performance in visual question answering tasks - which can then be effectively used by the LLM to perform downstream tasks, while reducing the amount of visual information to be processed by the LLM, which can reduce the training time and computational resources significantly.

Ablation on Pooling Resolution

In order to evaluate the effectiveness of pooling-based aggregators with different numbers of visual tokens, average pooling-based aggregators with a final resolution of 2×4 (8 visual tokens), 4×4 (16 visual tokens) and 8×8 (64 visual tokens) are considered against the standard LLaVA architecture with 576 visual features. All these variants are trained with the same vision encoder (CLIP-ViT-Large-336) and LLM (LLaMA 3.2 1B). Differently from Section 4.2.1, the MLPs

before and after the aggregator have a single hidden layer with the same dimensionality of input - following the standard LLaVA approach. Even in this case learning rate used for MLPs in all these experiments is $1 \cdot 10^{-3}$ with a global batch size of 256 in the first stage and 128 in the second one.

Additionally, perceiver resampler is based on a set of learnable queries - which would learn to aggregate *different* aspects of the original visual features - to prevent a query collapse to the same representation an orthogonal query initialization is used, which forces the queries to be different from each other at the beginning of the training, which can help to learn to aggregate different aspects of the original visual features during warm-up phase.

Results reported in Table 4.18 show that 1) even if MLPs have a smaller number of parameters, the approach with only 8 pooled visual features - approximately 1.4% of the original - reaches competitive performance with standard LLaVA, losing only few percentage points in vision benchmarks and less than 10% in general ones; 2) when the number of visual tokens is increased to 16 and 64, the performance of pooling-based aggregators is improved in almost all benchmarks, **outperforming** the standard LLaVA architecture in some tasks, especially in visual question answering ones; 3) the performance gap using 16 pooled visual features - approximately 2.8% of the original - is lower than 64 variant in some benchmarks.

Table 4.18: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336 in different configurations to evaluate pooling strategies. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						**Knowledge**					**OCR**					**Vision**					**Other**					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Average Pooling 2×4	60.7	**49.5**	73.2	1358.5	**66.6**	46.3	28.4	31.8	**31.7**	4.2	46.0	24.8	9.8	11.1	37.4	**40.8**	36.1	43.4	13.3	40.1	47.6	42.9	30.5	52.5	38.1	50.3	43.0
	-7.9%	-14.9%	-12.8%	+3.0%	+10.8%	-24.5%	-28.1%	-52.4%	-7.0%	-48.1%	-6.3%	-26.6%	-18.3%	-61.1%	-24.1%	-10.1%	-1.6%	-9.6%	+3.1%	+1.5%	+2.3%	-17.8%	-9.2%	-6.9%	-26.0%	-15.3%	-28.7%
Average Pooling 4×4	**62.0**	**49.5**	**74.8**	**1439.8**	65.7	**47.9**	**28.8**	31.8	30.7	**5.9**	46.9	**26.5**	**10.8**	**14.7**	**39.8**	40.5	**37.3**	**44.4**	**14.7**	**41.0**	**49.2**	**45.0**	**30.9**	**53.6**	**44.2**	**50.7**	45.5
	-5.9%	-14.9%	-10.8%	+9.2%	+9.3%	-21.9%	-27.1%	-52.4%	-10.0%	-27.2%	-4.5%	-21.6%	-10.0%	-48.4%	-19.3%	-10.8%	+1.6%	-7.5%	+14.0%	+3.8%	+5.9%	-13.8%	-8.0%	-5.0%	-14.2%	-14.6%	-24.5%
Average Pooling 8×8	60.7	**49.5**	73.5	1361.2	66.0	46.2	**28.8**	**32.0**	31.6	4.4	**47.0**	24.2	10.6	10.6	37.0	38.7	37.0	43.9	**14.7**	40.6	48.9	43.9	30.7	52.7	39.2	50.2	**46.6**
	-7.9%	-14.9%	-12.4%	+3.2%	+9.8%	-24.6%	-27.1%	-52.1%	-7.3%	-45.7%	-4.3%	-28.4%	-11.7%	-62.8%	-24.9%	-14.8%	+0.8%	-8.5%	+14.0%	+2.8%	+5.2%	-15.9%	-8.6%	-6.6%	-23.9%	-15.5%	-22.7%

This last observation can be explained by the fact that when the number of pooled visual features is increased, the total amount of samples needed to train the aggregator could be greater. In this case 64 pooled visual features variant could suffer under-fitting due to the limited number of training samples. This behavior is different from the one observed in Section 4.1.8 with Slot Attention-based aggregators - where increasing the number of slots and iterations improved

the performance in almost all benchmarks - but this is reasonable because in that case the aggregator is pre-trained with enough samples to reach the convergence, while in this current case the aggregator is trained from scratch with the LLaVA training pipeline, which could not be enough to reach the convergence when the number of pooled visual features is increased.

Max Pooling

In previous sections average pooling-based aggregators were considered under the hypothesis that they are able to preserve the vision encoder's final distribution, which is text-aware and simple to be aligned with the LLM language space. However, it is reasonable to think that max pooling - which intrinsically selects the most relevant features provided by first MLP - could be more effective in learning to distill the most relevant information into the dense visual concepts, which can then be effectively used by the LLM to perform downstream tasks.

For this reason, max pooling-based aggregators with a final resolution of 2×4 (8 visual tokens) and 4×4 (16 visual tokens) are considered against the standard LLaVA architecture with 576 visual features. All other hyperparameters are the same as those of the average pooling-based aggregators described in Section 4.2.1.

Table 4.19: Comparison considering LLaVA LLaMA 3.2 1B with different pooling configurations. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	MMBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Average Pooling 2×4	60.7	**49.5**	73.2	1358.5	**66.6**	46.3	**28.4**	**31.8**	**31.7**	4.2	46.0	24.8	9.8	11.1	37.4	40.8	36.1	**43.4**	13.3	40.1	47.6	42.9	30.5	52.5	38.1	50.3	43.0
	-7.9%	-14.9%	-12.8%	+3.0%	+10.8%	-24.5%	-28.1%	-52.4%	-7.0%	-48.1%	-6.3%	-26.6%	-18.3%	-61.1%	-24.1%	-10.1%	-1.6%	-9.6%	+3.1%	+1.5%	+2.3%	-17.8%	-9.2%	-6.9%	-26.0%	-15.3%	-28.7%
Max Pooling 2×4	**62.1**	49.0	**73.6**	**1489.9**	66.4	**46.8**	27.8	31.2	28.4	**4.9**	**46.7**	**26.2**	**10.3**	**12.2**	**38.9**	**43.5**	**38.0**	42.4	**16.7**	**43.3**	**49.5**	**44.8**	**31.5**	**53.0**	**43.0**	**51.3**	**45.0**
	-5.8%	-15.8%	-12.3%	+13.0%	+10.5%	-23.7%	-29.6%	-53.3%	-16.7%	-39.5%	-4.9%	-22.5%	-14.2%	-57.2%	-21.1%	-4.2%	+3.5%	-11.7%	+29.5%	+9.6%	+6.4%	-14.2%	-6.2%	-6.0%	-16.5%	-13.6%	-25.4%
Average Pooling 4×4	**62.0**	**49.5**	74.8	**1439.8**	65.7	**47.9**	**28.8**	**31.8**	**30.7**	**5.9**	**46.9**	26.5	10.8	**14.7**	39.8	40.5	37.3	**44.4**	14.7	41.0	49.2	**45.0**	**30.9**	**53.6**	**44.2**	50.7	**45.5**
	-5.9%	-14.9%	-10.8%	+9.2%	+9.3%	-21.9%	-27.1%	-52.4%	-10.0%	-27.2%	-4.5%	-21.6%	-10.0%	-48.4%	-19.3%	-10.8%	+1.6%	-7.5%	+14.0%	+3.8%	+5.9%	-13.8%	-8.0%	-5.0%	-14.2%	-14.6%	-24.5%
Max Pooling 4×4	61.9	49.3	**75.0**	1425.0	**66.3**	47.7	28.2	31.1	30.6	4.9	46.0	**26.6**	**11.1**	**14.7**	**39.9**	**40.7**	**38.9**	43.8	**16.7**	**44.9**	**50.3**	44.2	29.1	53.3	41.7	**52.4**	44.6
	-6.1%	-15.3%	-10.6%	+8.1%	+10.3%	-22.2%	-28.6%	-53.4%	-10.3%	-39.5%	-6.3%	-21.3%	-7.5%	-48.4%	-19.1%	-10.4%	+6.0%	-8.8%	+29.5%	+13.7%	+8.2%	-15.3%	-13.4%	-5.5%	-19.0%	-11.8%	-26.0%

In Table 4.19 are reported results showing that max pooling-based aggregators are able to outperform average pooling-based ones especially in document understanding and visual tasks. In particular, in visual question answering tasks, max pooling-based aggregator with 16 pooled visual features **outperforms** the standard LLaVA architecture with an average improvement of 6 percentage points, demonstrating that max pooling - which intrinsically selects the most

relevant features provided by first MLP - is more effective in learning to distill the most relevant information into the dense visual concepts, which can then be effectively used by the LLM in downstream tasks, especially in visual question answering tasks, which require a deeper understanding of the image.

Normalization Before LLM Injection

In Gemma-4 [53] was introduced a normalization layer which is applied before the visual features LLM injection, in order to help to fit the language distribution that is normalized in turn, simplifying the learning process of the last MLP, which can avoid learning how to scale the visual features.

In this section a normalization layer is applied before the LLM injection in the pooling-based aggregator with a final resolution of 2×4. In order to be more consistent with the proposed Gemma-4 architecture, average pooling-based aggregator is considered. All other hyperparameters are the same of the average pooling-based aggregators described in previous sections with two MLPs.

The results reported in Table 4.20 show that the normalization layer is able to improve the performance in some tasks, but there is not a clear improvement in all benchmarks, demonstrating that the normalization layer can be helpful in some cases, but it is not a crucial component to boost the performance of proposed approach.

Table 4.20: Comparison considering LLaVA LLaMA 3.2 1B with different pooling configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Average Pooling 2×4	**60.7**	**49.5**	**73.2**	**1358.5**	**66.6**	**46.3**	28.4	**31.8**	31.7	4.2	46.0	**24.8**	9.8	**11.1**	**37.4**	40.8	36.1	**43.4**	**13.3**	40.1	47.6	42.9	**30.5**	**52.5**	38.1	**50.3**	**43.0**
	-7.9%	-14.9%	-12.8%	+3.0%	+10.8%	-24.5%	-28.1%	-52.4%	-7.0%	-48.1%	-6.3%	-26.6%	-18.3%	-61.1%	-24.1%	-10.1%	-1.6%	-9.6%	+3.1%	+1.5%	+2.3%	-17.8%	-9.2%	-6.9%	-26.0%	-15.3%	-28.7%
Average Pooling 2×4 with Normalization	57.5	47.1	70.7	1309.6	60.7	43.4	**28.7**	30.8	**31.8**	**4.8**	**47.4**	22.9	**9.9**	6.6	33.6	**41.5**	**36.6**	42.0	**13.3**	**43.0**	**48.2**	**43.1**	29.3	50.6	**44.4**	49.0	42.1
	-12.7%	-19.1%	-15.7%	-0.7%	+1.0%	-29.2%	-27.3%	-53.9%	-6.7%	-40.7%	-3.5%	-32.2%	-17.5%	-76.8%	-31.8%	-8.6%	-0.3%	-12.5%	+3.1%	+8.9%	+3.6%	-17.4%	-12.8%	-10.3%	-13.8%	-17.5%	-30.2%

2 Epochs in Stage 1

Previously described experiments rely on a standard LLaVA training pipeline in which aggregation components are plugged in the architecture between the vision encoder and the LLM.

The first stage of a standard LLaVA pipeline consists of only one epoch of training - freezing the LLM - which is relatively fast, especially compared with the second one. Given that the aggregation components are trained from scratch, it is reasonable to think that increasing the number of epochs in the first stage could help the model to learn better how to aggregate the visual features provided by the vision encoder, which can lead to a better performance in downstream tasks in the second stage, when the LLM is unfrozen and can provide a more informative training signal to the aggregation components.

For this reason the pooling-based aggregator with a final resolution of 4×4 is trained with 2 epochs in the first stage, while all other hyperparameters are the same of the average pooling-based aggregators described in previous sections with two MLPs.

Table 4.21: Comparison considering LLaVA LLaMA 3.2 1B with different pooling configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						**Knowledge**					**OCR**					**Vision**					**Other**					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Max Pooling 4×4	**61.9**	**49.3**	**75.0**	1425.0	66.3	**47.7**	28.2	**31.1**	30.6	**4.9**	**46.0**	26.6	**11.1**	14.7	**39.9**	40.7	**38.9**	**43.8**	**16.7**	**44.9**	**50.3**	44.2	29.1	**53.3**	41.7	**52.4**	44.6
1 Epoch in Stage 1	-6.1%	-15.3%	-10.6%	+8.1%	+10.3%	-22.2%	-28.6%	-53.4%	-10.3%	-39.5%	-6.3%	-21.3%	-7.5%	-48.4%	-19.1%	-10.4%	+6.0%	-8.8%	+29.5%	+13.7%	+8.2%	-15.3%	-13.4%	-5.5%	-19.0%	-11.8%	-26.0%
Max Pooling 4×4	61.8	49.2	73.9	**1436.9**	**66.8**	47.5	**28.9**	30.7	**34.9**	4.4	45.7	**27.6**	10.8	**16.2**	39.8	**43.4**	32.6	41.6	12.7	39.6	46.5	**44.5**	**30.3**	51.8	**41.9**	50.9	**47.5**
2 Epochs in Stage 1	-6.2%	-15.5%	-11.9%	+9.0%	+11.1%	-22.5%	-26.8%	-54.0%	+2.3%	-45.7%	-6.9%	-18.3%	-10.0%	-43.2%	-19.3%	-4.4%	-11.2%	-13.3%	-1.6%	+0.3%	-0.1%	-14.8%	-9.8%	-8.2%	-18.6%	-14.3%	-21.2%

The results reported in Table 4.21 show that increasing the number of epochs in the first stage can lead to a better performance in some downstream tasks, especially in general ones, demonstrating that increasing the number of epochs in the first stage can help the model to learn better how to aggregate the visual features. However, this variant reaches worse performance in vision-centric tasks, which can be explained by the fact that the model could overfit to the training data in the first stage - and it is not able to recover from it in the second stage where the learning rate is much lower - maybe shifting from in-distribution data to more abstract and less understandable data due to the increased training time. This can be a problem especially for vision-centric tasks, which require a deeper understanding of the image, while for general tasks the model can rely more on the language understanding capabilities of the LLM, which can be less affected by the overfitting problem in the first stage.

4.2.2 LLaVA LLaMA 3.2 1B with Perceiver Resampler Aggregator

In contrast to simple pooling-based aggregators, the perceiver resampler (Section 3.1.5) - sometimes also called *attention pooling* - is a more complex and powerful approach to reduce the number of visual features, given that it has learnable parameters and it can learn to extract the most relevant information from the original visual features, which can be then effectively aggregated into the dense visual concepts, which can then be effectively used by the LLM to perform downstream tasks.

Similarly to pooling-based aggregators, different architectural configurations are evaluated to understand the best way to plug the perceiver resampler in the LLaVA pipeline, comparing 1) two MLPs with hidden size equal to input dimensionality before and after the perceiver resampler; 2) MLP with hidden size equal to input dimensionality only after the aggregator; 3) MLP with a hidden size which is double the input dimensionality only after the aggregator. In all these experiments the learning rate used for MLPs is $1 \cdot 10^{-3}$, while the learning rate used for the perceiver resampler is $2 \cdot 10^{-4}$, with a global batch size of 256 in the first stage and 128 in the second one - exactly as specified in the original LLaVA training pipeline.

Results are reported in Table 4.22 and on the contrary of what happens for pooling-based aggregators, the best performance is mainly obtained by the variants with only one MLP, where the one with hidden size equal to input dimensionality beats others especially in visual question answering tasks.

Table 4.22: Comparison considering LLaVA LLaMA 3.2 1B with different perceiver resampler with 6 blocks configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR$_{Bench}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Perceiver 8 Queries MLP only after	60.1	48.0	**72.9**	1349.9	65.6	46.4	28.8	**31.5**	**33.4**	4.1	46.1	26.1	9.5	14.7	**38.1**	**42.2**	**37.4**	**44.1**	14.7	**43.0**	**47.6**	42.1	**31.1**	**53.0**	38.4	48.3	39.9
	-8.8%	-17.5%	-13.1%	+2.4%	+9.2%	-24.3%	-27.1%	-52.8%	-2.1%	-49.4%	-6.1%	-22.8%	-20.8%	-48.4%	-22.7%	-7.0%	+1.9%	-8.1%	+14.0%	+8.9%	+2.4%	-19.3%	-7.4%	-6.0%	-25.4%	-18.7%	-33.8%
Perceiver 8 Queries MLP only after Double Dim.	**61.1**	**48.7**	71.9	**1418.1**	**67.0**	**46.8**	28.5	31.4	32.2	4.7	45.7	**26.3**	10.7	**15.5**	37.9	41.1	34.8	42.6	11.3	39.1	46.2	**44.4**	30.1	52.1	**44.2**	**49.7**	**45.7**
	-7.3%	-16.3%	-14.3%	+7.6%	+11.5%	-23.7%	-27.8%	-53.0%	-5.6%	-42.0%	-6.9%	-22.2%	-10.8%	-45.6%	-23.1%	-9.5%	-5.2%	-11.3%	-12.4%	-1.0%	-0.7%	-14.9%	-10.4%	-7.6%	-14.2%	-16.3%	-24.2%
Perceiver 8 Queries Two MLPs (Before and After)	57.7	46.4	68.6	1363.7	61.3	44.0	**29.0**	31.2	33.2	**4.9**	**46.7**	22.3	**10.8**	5.3	33.5	39.5	36.7	43.5	**16.7**	39.7	46.8	42.6	30.1	52.0	37.0	49.3	44.5
	-12.4%	-20.3%	-18.2%	+3.4%	+2.0%	-28.2%	-26.6%	-53.3%	-2.6%	-39.5%	-4.9%	-34.0%	-10.0%	-81.4%	-32.0%	-13.0%	0.0%	-9.4%	+29.5%	+0.5%	+0.6%	-18.4%	-10.4%	-7.8%	-28.2%	-17.0%	-26.2%

This can be explained by the fact that perceiver resampler - which has learnable parameters - is able to extract the most relevant information from the original visual features, which can be then effectively aggregated into the dense visual concepts, which can then be effectively used

by the LLM to perform downstream tasks, without the need for an additional MLP before the aggregator, which could introduce noise in the input of the perceiver resampler, making it harder for it to learn how to extract the most relevant information from the original visual features. Moreover, additional MLP could introduce a training instability, especially if it has a learning rate which is much higher than the one used for the perceiver resampler.

In fact when the input visual features change too fast, the perceiver resampler might not be able to learn how to extract the most relevant information from the original visual features, given that it is not able to adapt to the fast changing input visual features, which can lead to a worse performance in downstream tasks.

Perceiver Resampler's Number of Blocks Analysis

Perceiver resampler is a more complex, but potentially more powerful approach compared to pooling-based aggregators, given that it has learnable parameters and it can learn to extract the most relevant information from the original visual features, which can be then effectively aggregated into the dense visual concepts, which can then be effectively used by the LLM to perform downstream tasks.

One of the most important hyperparameters of the perceiver resampler is the number of blocks, which can affect the capability to learn complex visual representations. In order to evaluate the impact of this hyperparameter, two variants of the perceiver resampler with 3 and 6 blocks are considered against the standard LLaVA architecture with 576 visual features. All other hyperparameters are the same of the best variant described in previous section with only one MLP after the aggregator.

In particular, the perceiver resampler with 6 blocks was trained with a learning rate of $2 \cdot 10^{-4}$, while the one with 3 blocks was trained with a learning rate of $5 \cdot 10^{-4}$, given that the one with 3 blocks has less parameters and it is able to learn faster, while the one with 6 blocks has more parameters and it learns slower, needing more time to learn how to extract the most relevant information from the original visual features.

It is abundantly clear that, based on results illustrated in Table 4.23, the variant with 6 blocks is able to outperform the one with 3 blocks in almost all benchmarks, demonstrating that increasing the number of blocks can help the model to learn more complex visual representations, which can be then effectively used by the LLM to perform downstream tasks.

It is important to note that this does not exclude the possibility to have an under-fitting

problem, because an architecture with more parameters - such as the one with 6 blocks - could reach better performance in less time, being able to exploit the training data more effectively [71].

Table 4.23: Comparison considering LLaVA LLaMA 3.2 1B with different perceiver resampler configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Perceiver (8 Q.) 3 Blocks	59.6	**48.1**	**73.1**	1298.0	**65.7**	46.3	28.5	31.4	31.9	**4.6**	**46.1**	24.9	**11.1**	13.3	37.6	37.6	36.7	42.4	**16.0**	40.6	**47.6**	**43.2**	30.1	51.6	**40.8**	**48.6**	**44.8**
	-9.6%	-17.4%	-12.9%	-1.6%	+9.3%	-24.5%	-27.8%	-53.0%	-6.5%	-43.2%	-6.1%	-26.3%	-7.5%	-53.3%	-23.7%	-17.2%	0.0%	-11.7%	+24.0%	+2.8%	+2.3%	-17.2%	-10.4%	-8.5%	-20.8%	-18.2%	-25.7%
Perceiver (8 Q.) 6 Blocks	**60.1**	48.0	72.9	**1349.9**	65.6	**46.4**	**28.8**	**31.5**	**33.4**	4.1	**46.1**	**26.1**	9.5	**14.7**	**38.1**	**42.2**	**37.4**	**44.1**	14.7	**43.0**	**47.6**	42.1	**31.1**	**53.0**	38.4	48.3	39.9
	-8.8%	-17.5%	-13.1%	+2.4%	+9.2%	-24.3%	-27.1%	-52.8%	-2.1%	-49.4%	-6.1%	-22.8%	-20.8%	-48.4%	-22.7%	-7.0%	+1.9%	-8.1%	+14.0%	+8.9%	+2.4%	-19.3%	-7.4%	-6.0%	-25.4%	-18.7%	-33.8%

Ablation on Number of Queries

Similarly to pooling-based, information budget can be expanded by increasing the number of queries in the perceiver resampler aggregator. Theoretically, this allows the model to discard less information. However, by increasing the number of queries, the query collapse problem can arise, making pointless the increase of the number of queries - given that they could collapse to the same representation.

In particular, in Table 4.24 are reported results of perceiver resampler with 6 blocks considering 8, 16 and 64 queries against the standard LLaVA architecture with 576 visual features. All other hyperparameters are the same of the best variant described in previous section with only one MLP after the aggregator and following the standard LLaVA training pipeline.

Similarly to pooling-based aggregators, even in this case the configuration with 16 queries is able to obtain the best performance in several benchmarks, even if 64 queries variant is able to **outperform** both the 16 queries variant and the standard LLaVA model in some visual question answering tasks.

The performance gap using 64 queries compared to 16 variant can be explained by the fact that when the number of queries is increased, the total amount of samples needed to train the aggregator could be greater. In this case 64 queries variant could suffer from under-fitting due to the limited number of training samples. Additionally, the 64 queries variant could be more prone to query collapse problem, given that it has more queries to learn how to aggregate the original visual features, which can lead to a worse performance in downstream tasks.

Table 4.24: Comparison considering LLaVA LLaMA 3.2 1B with different perceiver resampler with 6 blocks configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Perceiver 8 Queries	60.1	48.0	72.9	1349.9	65.6	46.4	**28.8**	31.5	**33.4**	4.1	46.1	26.1	9.5	14.7	38.1	**42.2**	**37.4**	44.1	14.7	**43.0**	47.6	42.1	**31.1**	**53.0**	38.4	48.3	39.9
	-8.8%	-17.5%	-13.1%	+2.4%	+9.2%	-24.3%	-27.1%	-52.8%	-2.1%	-49.4%	-6.1%	-22.8%	-20.8%	-48.4%	-22.7%	-7.0%	+1.9%	-8.1%	+14.0%	+8.9%	+2.4%	-19.3%	-7.4%	-6.0%	-25.4%	-18.7%	-33.8%
Perceiver 16 Queries	**60.8**	**49.1**	**73.6**	**1361.2**	65.9	**47.4**	28.4	**31.9**	31.0	4.0	**46.7**	**26.8**	**11.0**	16.1	**38.6**	41.4	36.4	**44.2**	15.3	39.3	46.6	43.6	29.8	52.4	**42.8**	49.2	43.9
	-7.7%	-15.6%	-12.3%	+3.2%	+9.7%	-22.7%	-28.1%	-52.2%	-9.1%	-50.6%	-4.9%	-20.7%	-8.3%	-43.5%	-21.7%	-8.8%	-0.8%	-7.9%	+18.6%	-0.5%	+0.2%	-16.5%	-11.3%	-7.1%	-16.9%	-17.2%	-27.2%
Perceiver 64 Queries	60.5	48.3	73.3	1347.6	**66.0**	**47.4**	28.7	31.8	31.6	**4.6**	**46.7**	26.1	**11.0**	**16.5**	37.4	39.6	37.3	42.1	**17.3**	40.9	**48.7**	**44.2**	30.5	52.4	39.5	**50.0**	**48.7**
	-8.2%	-17.0%	-12.6%	+2.2%	+9.8%	-22.7%	-27.3%	-52.4%	-7.3%	-43.2%	-4.9%	-22.8%	-8.3%	-42.1%	-24.1%	-12.8%	+1.6%	-12.3%	+34.1%	+3.5%	+4.8%	-15.3%	-9.2%	-7.1%	-23.3%	-15.8%	-19.2%

In fact, in Figure 4.7 is reported the histogram of the cosine similarity between the 64 query pairs - excluding self-similarity, i.e., 4,032 pairs - after the training of second stage. It is evident that a lot of pairs have a cosine similarity higher than 0.8 - which is a clear sign of query collapse problem - and the distribution is skewed towards high similarity values, instead of being a reasonable normal distribution centered around 0.5, which is the expected value of the cosine similarity between two random vectors in a high-dimensional space.

This analysis demonstrates that query collapse problem arises when the number of queries is increased, despite the orthogonal initialization and small learning rates. Additional experiments are needed to understand how to mitigate this problem effectively, given that it is a crucial aspect to improve the performance of perceiver resampler aggregators with a higher number of queries, which can help to reduce the information loss and improve the performance in downstream tasks.

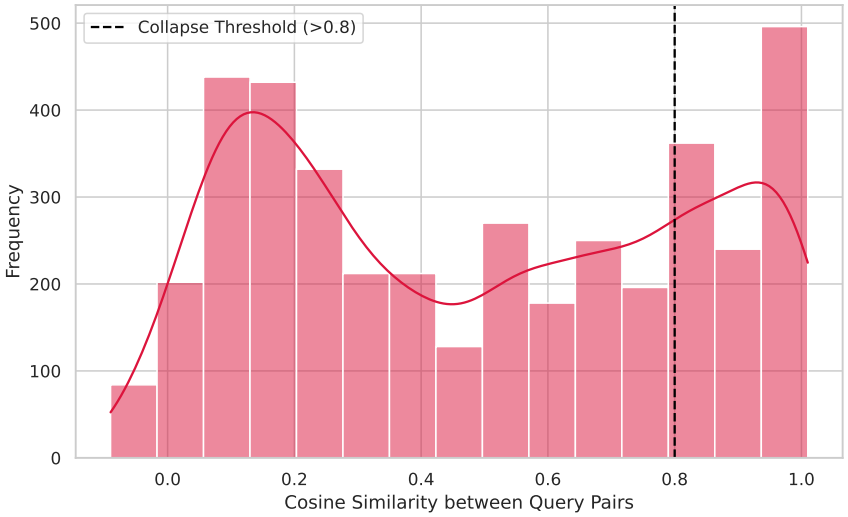


Figure 4.7: Histogram of the cosine similarity between the 64 query pairs excluding self-similarity.

Pooling-based Query Initialization

In Section 4.2.2 it is demonstrated that the query collapse problem arises when the number of queries is increased, despite the orthogonal initialization.

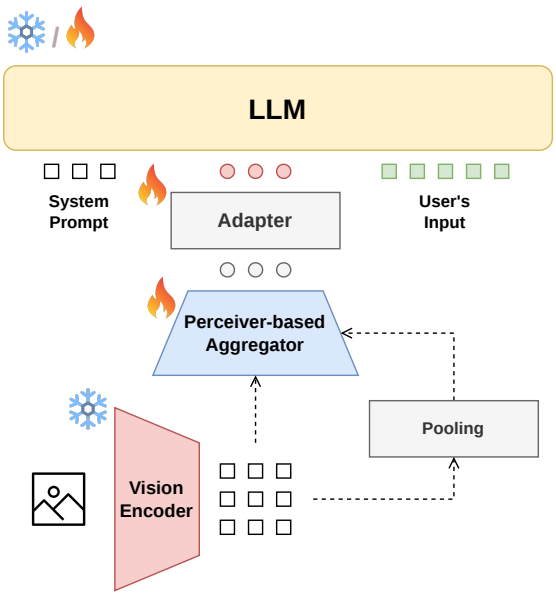


Figure 4.8: Pooling-based query initialization architecture.

In this section a more aggressive approach to discourage query collapse problem is considered and illustrated in Figure 4.8, which consists of initializing the queries with the sum of learnable

vectors and the pooled visual features - the pooling resolution must be equal to the number of queries. In this way, queries are intrinsically different from each other with a natural diversity which encourages spatial distinction in the aggregation process, which can help to mitigate the query collapse problem effectively, even when the number of queries is increased.

Results are reported in Table 4.25. In the configuration with 8 queries the pooling-based query initialization is able to improve the performance in some benchmarks, especially in visual question answering tasks, while in the configuration with 64 queries the pooling-based query initialization is able to improve the performance in almost all benchmarks.

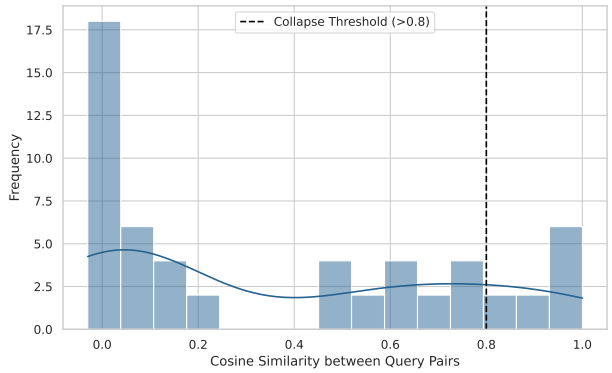
Table 4.25: Comparison considering LLaVA LLaMA 3.2 1B with different perceiver configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Perceiver 8 Q. Ortoghonal Init.	**60.1** -8.8%	48.0 -17.5%	72.9 -13.1%	**1349.9** +2.4%	65.6 +9.2%	**46.4** -24.3%	**28.8** -27.1%	**31.5** -52.8%	**33.4** -2.1%	4.1 -49.4%	**46.1** -6.1%	**26.1** -22.8%	9.5 -20.8%	**14.7** -48.4%	**38.1** -22.7%	**42.2** -7.0%	37.4 +1.9%	44.1 -8.1%	14.7 +14.0%	**43.0** +8.9%	47.6 +2.4%	42.1 -19.3%	**31.1** -7.4%	**53.0** -6.0%	38.4 -25.4%	48.3 -18.7%	39.9 -33.8%
Perceiver 8 Q. With Pooling Init.	60.0 -9.0%	**48.1** -17.4%	**73.7** -12.2%	1315.7 -0.2%	**66.2** +10.1%	46.3 -24.5%	28.2 -28.6%	**31.5** -52.8%	31.2 -8.5%	**4.3** -46.9%	45.8 -6.7%	25.6 -24.3%	**10.0** -16.7%	14.6 -48.8%	38.0 -22.9%	39.9 -12.1%	**38.3** +4.4%	**44.6** -7.1%	**20.0** +55.0%	40.1 +1.5%	**48.5** +4.2%	**43.5** -16.7%	29.6 -11.9%	52.4 -7.1%	**40.8** -20.8%	**49.6** -16.5%	**45.3** -24.9%
Perceiver 64 Q. Ortoghonal Init.	60.5 -8.2%	**48.3** -17.0%	**73.3** -12.6%	1347.6 +2.2%	66.0 +9.8%	47.4 -22.7%	28.7 -27.3%	**31.8** -52.4%	31.6 -7.3%	4.6 -43.2%	46.7 -4.9%	26.1 -22.8%	**11.0** -8.3%	16.5 -42.1%	37.4 -24.1%	**39.6** -12.8%	37.3 +1.6%	42.1 -12.3%	**17.3** +34.1%	40.9 +3.5%	48.7 +4.8%	**44.2** -15.3%	30.5 -9.2%	52.4 -7.1%	39.5 -23.3%	**50.0** -15.8%	**48.7** -19.2%
Perceiver 64 Q. With Pooling	**61.0** -7.4%	48.1 -17.4%	72.7 -13.3%	**1406.2** +6.7%	**66.2** +10.1%	**47.8** -22.0%	**29.1** -26.3%	31.4 -53.0%	**33.1** -2.9%	**4.9** -39.5%	**46.8** -4.7%	**26.3** -22.2%	10.4 -13.3%	**17.1** -40.0%	**39.4** -20.1%	38.1 -16.1%	**37.9** +3.3%	**43.4** -9.6%	**17.3** +34.1%	**41.7** +5.6%	**49.2** +5.8%	43.9 -15.9%	**31.2** -7.1%	**52.9** -6.2%	**41.4** -19.6%	**50.0** -15.8%	44.1 -26.9%

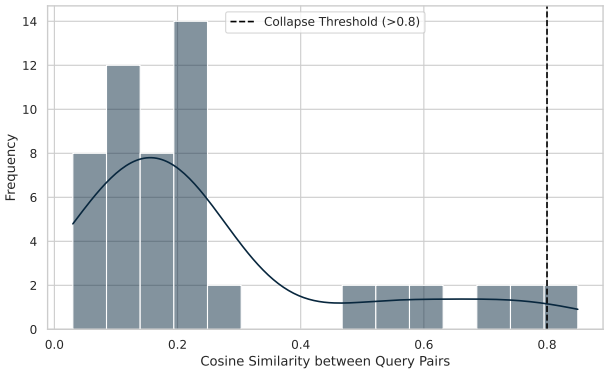
Additionally, in Figure 4.11b is reported the histogram of the cosine similarity between the 64 query pairs - excluding self-similarity - after the training of second stage when the queries are initialized with pooling. It is evident that a lot of pairs have a cosine similarity lower than 0.8 - that is a clear sign of query collapse problem mitigation - and the distribution is more balanced with respect to the one reported in Figure 4.11a, which is skewed towards high similarity values.

Similar evaluation is provided for the 8 (Figure 4.9) and 16 queries configurations (Figure 4.10) where the query collapse problem is less pronounced compared to 64 queries variant (Figure 4.11), but it is still present and the pooling-based query initialization is able to mitigate it effectively.

These results demonstrate that solving the query collapse problem is crucial to improve the performance of perceiver resampler aggregators with a higher number of queries, which can help to reduce the information loss and improve the performance in downstream tasks. Pooling-based query initialization is an interesting approach to mitigate the query collapse problem - given that

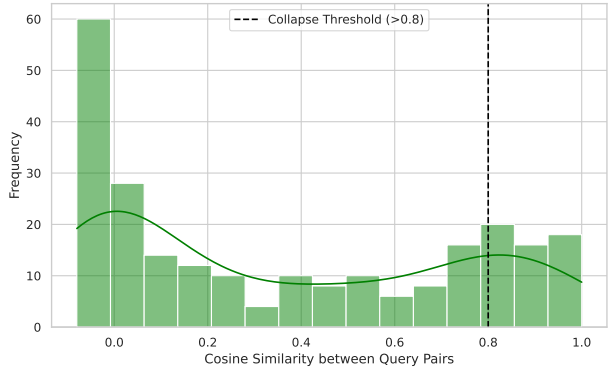


(a) Histogram of the cosine similarity between the 8 query pairs.

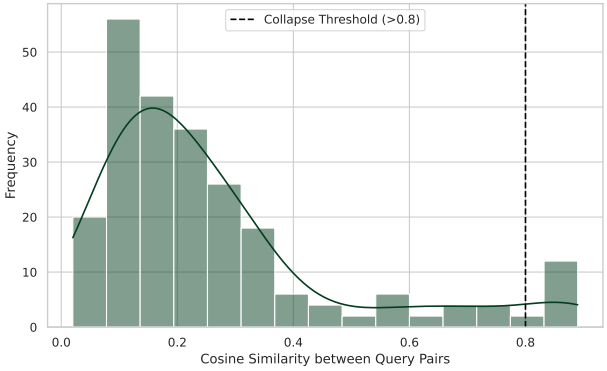


(b) Histogram of the cosine similarity between the 8 query pairs when initialized with pooling.

Figure 4.9: Comparison of the cosine similarity distribution between the 8 query pairs.

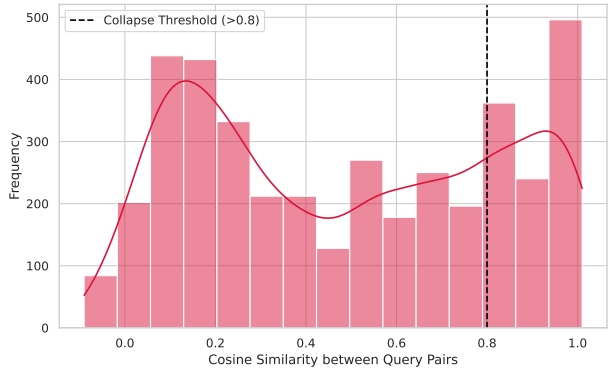


(a) Histogram of the cosine similarity between the 16 query pairs.

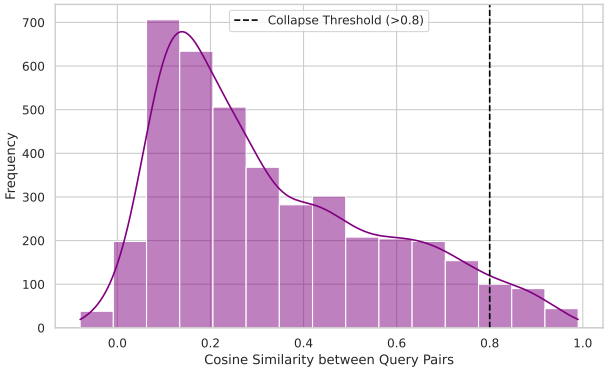


(b) Histogram of the cosine similarity between the 16 query pairs when initialized with pooling.

Figure 4.10: Comparison of the cosine similarity distribution between the 16 query pairs.



(a) Histogram of the cosine similarity between the 64 query pairs.



(b) Histogram of the cosine similarity between the 64 query pairs when initialized with pooling.

Figure 4.11: Comparison of the cosine similarity distribution between the 64 query pairs.

it is simple to implement, does not require additional parameters and it naturally provides spatial bias which can be helpful for the downstream tasks - however, it might not be the best solution to solve this problem. More experiments are needed to understand how to mitigate this problem effectively, given that it is a crucial aspect to improve the performance of perceiver resampler aggregators with a higher number of queries, which can help to reduce the information loss and improve the performance in downstream tasks.

2 Epochs in Stage 1

Similar to pooling-based aggregators, because there are more complex components in the LLaVA pipeline compared to the simple MLP adapter of the standard LLaVA architecture, it is reasonable to hypothesize that increasing the number of epochs in the first stage could help the model to better learn how to aggregate the visual features provided by the vision encoder, which can lead to better performance in downstream tasks in the second stage, when the LLM is unfrozen and can provide a more informative training signal to the aggregation components.

In order to evaluate this hypothesis, the perceiver resampler with 8 queries is considered in two configurations: 1) only learnable vectors and 2) with pooling initialization of queries. These models are trained with 2 epochs in the first stage, while all other hyperparameters are the same as those of the best variant described in the previous section with only one MLP after the aggregator. In second stage only one epoch is used.

Table 4.26: Comparison considering LLaVA LLaMA 3.2 1B with different perceiver configurations. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Perc. 8q 6b Identity	**60.1** -8.8%	48.0 -17.5%	72.9 -13.1%	**1349.9** +2.4%	65.6 +9.2%	46.4 -24.3%	**28.8** -27.1%	**31.5** -52.8%	**33.4** -2.1%	4.1 -49.4%	46.1 -6.1%	26.1 -22.8%	9.5 -20.8%	14.7 -48.4%	38.1 -22.7%	**42.2** -7.0%	**37.4** +1.9%	**44.1** -8.1%	14.7 +14.0%	**43.0** +8.9%	47.6 +2.4%	42.1 -19.3%	**31.1** -7.4%	**53.0** -6.0%	38.4 -25.4%	48.3 -18.7%	39.9 -33.8%
Perc. 8q 6b Identity (2ep)	60.0 -9.0%	**48.1** -17.4%	**73.0** -13.0%	1315.4 -0.2%	**66.0** +9.8%	**47.0** -23.3%	28.5 -27.8%	30.9 -53.7%	32.0 -6.2%	**4.5** -44.4%	**46.4** -5.5%	**27.5** -18.6%	**10.4** -13.3%	**18.2** -36.1%	**40.4** -18.1%	41.1 -9.5%	**37.4** +1.9%	43.5 -9.4%	**16.7** +29.5%	40.6 +2.8%	**48.9** +5.1%	**42.7** -18.2%	30.4 -9.5%	52.3 -7.3%	**39.2** -23.9%	**49.1** -17.3%	**42.5** -29.5%
Perc. 8q 6b Init Latents	60.0 -9.0%	**48.1** -17.4%	73.7 -12.2%	1315.7 -0.2%	66.2 +10.1%	**46.3** -24.5%	28.2 -28.6%	31.5 -52.8%	31.2 -8.5%	**4.3** -46.9%	45.8 -6.7%	25.6 -24.3%	**10.0** -16.7%	14.6 -48.8%	38.0 -22.9%	**39.9** -12.1%	**38.3** +4.4%	**44.6** -7.1%	20.0 +55.0%	40.1 +1.5%	48.5 +4.2%	**43.5** -16.7%	**29.6** -11.9%	52.4 -7.1%	**40.8** -20.8%	**49.6** -16.5%	45.3 -24.9%
Perc. 8q 6b Init Latents (2ep)	**60.1** -8.8%	47.5 -18.4%	**74.3** -11.4%	**1320.4** +0.1%	**66.7** +11.0%	46.1 -24.8%	**28.6** -27.6%	**31.8** -52.4%	**33.0** -3.2%	3.7 -54.3%	**45.9** -6.5%	**26.4** -21.9%	9.9 -17.5%	**17.5** -38.6%	**38.6** -21.7%	39.6 -12.8%	38.0 +3.5%	41.8 -12.9%	**20.7** +60.5%	**40.4** +2.3%	**49.2** +5.7%	43.4 -16.9%	29.1 -13.4%	**52.8** -6.4%	40.0 -22.3%	48.9 -17.7%	**46.3** -23.2%

The results are reported in Table 4.26 show that increasing the number of epochs in the first stage leads to a better performance in most benchmarks, demonstrating that a small extra training

time helps complex aggregators to learn better how to aggregate the visual features provided by the vision encoder - possibly mitigating the query collapse problem and under-fitting problem.

4.2.3 LLaVA Vicuna 7B with Aggregation

Previous experiments were based on a LLaMA 3.2 1B, which is a relatively small model compared to the state-of-the-art LLMs. In this section a larger model is considered, i.e. Vicuna with 7B parameters.

In Table 4.27 reports results of Vicuna 7B with average pooling-based aggregator with a final resolution of 2×4 and perceiver resampler with 8 queries (without pooling initialization) against the standard LLaVA architecture with 576 visual features. All other hyperparameters are the same as those of the best variant described in previous section with only one MLP after the aggregator for perceiver-based aggregation and two MLPs for the pooling-based aggregation.

Table 4.27: Comparison considering Vicuna 7B baseline with different pooling configurations. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB $_{(EN)}$	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Vicuna 7B (Baseline)	70.8	61.6	86.9	1714.5	74.4	60.1	39.7	54.5	35.1	12.8	56.5	34.3	17.6	29.3	56.9	33.2	49.4	53.1	24.7	43.8	76.1	53.2	32.6	58.7	50.4	59.4	64.8
2D Pooling 2×4	**64.3**	**54.1**	**81.7**	**1601.2**	70.4	50.8	35.6	50.6	33.4	4.1	54.4	28.0	12.6	18.0	48.8	32.7	45.1	46.0	**15.3**	42.9	**76.3**	46.8	33.7	55.3	42.8	51.8	50.4
	-9.2%	-12.2%	-6.0%	-6.6%	-5.4%	-15.5%	-10.3%	-7.2%	-4.8%	-68.0%	-3.7%	-18.2%	-28.4%	-38.6%	-14.2%	-1.5%	-8.7%	-13.4%	-38.1%	-2.1%	+0.3%	-12.0%	+3.4%	-5.8%	-15.1%	-12.8%	-22.2%
Perceiver 8Q 6B	**64.3**	53.2	81.6	1560.8	**70.9**	**51.6**	**36.8**	**52.3**	**34.8**	**4.8**	**55.3**	**29.4**	**13.2**	**21.5**	**49.8**	**33.0**	**45.3**	**47.1**	14.7	**43.3**	76.1	**48.7**	**34.5**	**57.0**	**45.3**	**52.6**	**54.3**
	-9.1%	-13.6%	-6.1%	-9.0%	-4.7%	-14.1%	-7.4%	-4.0%	-0.9%	-62.5%	-2.1%	-14.2%	-25.0%	-26.6%	-12.5%	-0.6%	-8.3%	-11.3%	-40.5%	-1.1%	0.0%	-8.3%	+5.8%	-2.9%	-10.1%	-11.4%	-16.2%

Results show that perceiver-based aggregation is able to outperform the pooling-based one in almost all benchmarks and the standard LLaVA architecture in some cases.

This could be explained by the fact that Vicuna 7B is a larger model compared to LLaMA 3.2 1B, which could be able to better exploit the complex visual representations of the perceiver-based aggregator and, by providing a stronger training signal to the aggregation components, it could mitigate the query collapse problem.

Overall, even if it is not able to outperform the standard LLaVA architecture, the perceiver-based aggregation is able to reach competitive performance despite the extremely small number of visual features - only 1.4% of the original - reducing the training time and computational resources significantly.

4.2.4 Experiments with Qwen3 0.6B and 1.7B

In previous sections, LLaMA 3.2 1B was primarily considered as LLM which was chosen for its good balance between performance and computational resources needed for training, acting as a proxy for other LLMs.

In this section the Qwen3 family is considered as LLMs: Qwen3 0.6B is a small model with good intelligence density which can be used also in low-resource scenarios such as mobile applications or edge devices, while Qwen3 1.7B is a larger model which can be used in more complex scenarios, still remaining efficient and deployable in resource-constrained environments.

The following experiments are conducted with both LLMs across several configurations: 1) average pooling-based aggregator with a final resolution of 2×4, 4×4 and 8×8; 2) max pooling-based aggregator with a final resolution of 2×4, 4×4 and 8×8; 3) perceiver resampler with 6 blocks in 8, 16 and 64 queries configurations; 4) perceiver resampler with 6 blocks in 8, 16 and 64 queries with pooling initialization configurations.

All these variants are trained with the same vision encoder (CLIP-ViT-Large-336) and same architectures proposed when LLaMA 3.2 1B was considered - i.e., two MLPs for pooling-based aggregators and one MLP after the perceiver resampler for perceiver-based aggregators. The learning rate used for MLPs in all these experiments is $1 \cdot 10^{-3}$ with a global batch size of 256 in the first stage and 128 in the second one.

Qwen3 0.6B with Pooling-based Aggregators

Table 4.28 reports results of Qwen3 0.6B with average and max pooling-based aggregator with a final resolution of 2×4, 4×4 and 8×8 against the standard LLaVA architecture with 576 visual features.

It is evident that max pooling-based aggregators are able to outperform average pooling-based ones in almost all benchmarks. It is interesting to note that this behavior is preserved across different configurations for general benchmarks, while in some vision question answering tasks such as MMVP the max pooling reaches better performance only when the number of pooled visual features is increased. Moreover, increasing the number of pooled visual features there is a slight performance degradation in vision tasks.

This could be explained by the fact that when the number of pooled visual features is increased, the amount of information provided to the LLM is also increased and average pooling

could be able to preserve more information, while max pooling could be more effective in selecting the most relevant information from the original visual features when the information budget is limited, which is the case when the number of pooled visual features is low.

Moreover, when the number of pooled visual features is increased, the model could suffer from under-fitting due to the limited number of training samples - especially due to the limited capability of the model to provide detailed responses - which could lead to a performance degradation in vision tasks, which require a deeper understanding of the image.

Despite its size and related training difficulty, Qwen3 0.6B is able to reach competitive performance compared to standard LLaVA architecture, **outperforming** it in some complex tasks such as MMVP - where in the best case is able to reach an improvement of almost 20 percentage points. This demonstrates the effectiveness of the proposed approach also when a different smaller LLM is considered, paving the way for the deployment of these configurations in low-resource scenarios where computation resources are limited, but vision capabilities are required.

Table 4.28: Comparison considering Qwen3 0.6B baseline with different pooling configurations. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Qwen3 0.6B (Baseline)	65.2	53.7	84.0	1477.2	69.0	54.1	33.2	27.1	32.0	25.8	48.0	28.2	12.3	22.6	42.9	35.1	40.9	55.3	20.7	44.1	43.6	46.2	34.1	43.9	43.8	51.9	57.5
2D Pool 2x4 VP+MP (MLP)	55.5 -14.9%	45.3 -15.6%	71.6 -14.8%	1321.0 -10.6%	62.6 -9.3%	42.5 -21.4%	27.2 -18.1%	**27.4** +1.1%	30.4 -5.0%	5.2 -79.8%	**46.0** -4.2%	16.5 -41.5%	9.2 -25.2%	2.2 -90.3%	29.1 -32.2%	25.6 -27.1%	37.7 -8.0%	**50.7** -8.3%	16.0 -22.7%	40.9 -7.3%	43.0 -1.3%	40.6 -12.1%	32.5 -4.7%	38.8 -11.6%	**40.1** -8.4%	**47.7** -8.1%	43.9 -23.7%
2D Pool 2x4 Max Pooling	**56.9** -12.7%	**46.5** -13.4%	**71.9** -14.4%	**1358.0** -8.1%	**64.4** -6.7%	**44.7** -17.4%	**28.0** -15.7%	26.4 -2.6%	**34.9** +9.1%	**5.7** -77.9%	44.9 -6.5%	**20.4** -27.7%	**9.7** -21.1%	**9.4** -58.4%	**33.5** -21.9%	**29.2** -16.8%	**40.0** -2.2%	50.2 -9.2%	**24.7** +19.3%	**41.5** -5.9%	**43.7** +0.3%	**41.7** -9.7%	**32.6** -4.4%	**43.3** -1.4%	38.9 -11.2%	47.6 -8.3%	**46.0** -20.0%
2D Pool 4x4 VP+MP	56.4 -13.5%	45.7 -14.9%	**72.9** -13.2%	**1336.4** -9.5%	63.2 -8.4%	43.8 -19.0%	27.3 -17.8%	27.5 +1.5%	31.2 -2.5%	5.1 -80.2%	**45.4** -5.4%	18.8 -33.3%	9.4 -23.6%	**7.8** -65.5%	32.0 -25.4%	26.1 -25.6%	**39.4** -3.7%	51.0 -7.8%	**21.3** +2.9%	**41.7** -5.4%	**43.6** -0.1%	**41.6** -10.0%	**31.2** -8.5%	**42.9** -2.3%	**40.6** -7.3%	47.4 -8.7%	46.1 -19.8%
2D Pool 4x4 Max Pooling	**57.6** -11.7%	**46.8** -12.8%	72.6 -13.6%	1287.4 -12.8%	**64.6** -6.4%	**46.3** -14.4%	**28.1** -15.4%	**27.6** +1.8%	**33.7** +5.3%	**5.8** -77.5%	45.2 -5.8%	**19.9** -29.4%	**10.0** -18.7%	6.8 -69.9%	**34.9** -18.6%	**27.8** -20.8%	37.3 -8.8%	**51.2** -7.4%	14.0 -32.4%	40.5 -8.2%	43.4 -0.4%	40.4 -12.6%	30.3 -11.1%	40.5 -7.7%	36.3 -17.1%	**48.3** -6.9%	**46.6** -19.0%
2D Pool 8x8 VP+MP	59.6 -8.6%	48.4 -9.9%	**75.6** -10.0%	**1433.6** -3.0%	66.3 -3.9%	47.9 -11.5%	28.6 -13.9%	**28.1** +3.7%	31.6 -1.2%	9.5 -63.2%	45.3 -5.6%	22.2 -21.3%	**10.4** -15.4%	**14.0** -38.1%	37.4 -12.8%	27.1 -22.8%	**38.4** -6.1%	**52.9** -4.3%	**18.7** -9.7%	**39.2** -11.1%	**42.9** -1.6%	42.2 -8.7%	**34.2** +0.3%	42.2 -3.9%	36.3 -17.1%	**48.5** -6.6%	49.7 -13.6%
2D Pool 8x8 Max Pooling	**60.0** -8.0%	**49.4** -7.9%	75.5 -10.1%	1355.9 -8.2%	**66.4** -3.8%	**48.5** -10.4%	**30.0** -9.6%	27.3 +0.7%	**32.2** +0.6%	**14.3** -44.6%	**46.0** -4.2%	**23.0** -18.4%	10.3 -16.3%	10.8 -52.2%	**37.8** -11.9%	**32.9** -6.3%	37.6 -8.1%	52.0 -6.0%	18.0 -13.0%	37.7 -14.5%	42.8 -1.8%	**42.7** -7.6%	31.1 -8.8%	**42.9** -2.3%	**38.4** -12.3%	48.4 -6.7%	**52.7** -8.3%

Qwen3 1.7B with Pooling-based Aggregators

Exactly the same analysis is conducted for Qwen3 1.7B in Table 4.29 with similar results, except that Qwen3 1.7B is able to reach better performance with max pooling-based aggregators also in vision question answering tasks.

This could be explained by the fact that Qwen3 1.7B is a larger model compared to Qwen3

0.6B, which could be able to better exploit complex visual representations, providing a stronger training signal to the aggregation components, mitigating the under-fitting problem and leading to a better performance in vision question answering tasks.

Even in this case, despite its size and related training difficulty, Qwen3 1.7B is able to reach competitive performance compared to standard LLaVA architecture, **outperforming** it in some tasks. This demonstrates again the effectiveness of the proposed approach with another different and larger LLM.

Table 4.29: Comparison considering Qwen3 1.7B baseline with different pooling configurations. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Qwen3 1.7B (Baseline)	68.3	55.6	84.6	1638.2	74.9	58.1	34.6	30.8	35.9	15.4	56.2	31.6	14.2	24.9	49.3	38.0	42.0	49.5	24.0	46.5	47.9	51.6	36.3	57.1	48.7	55.5	60.2
2D Pool 2x4 VP+MP (MLP)	59.4 -13.0%	**47.3** -14.9%	**77.1** -8.9%	1310.9 -20.0%	67.3 -10.1%	45.8 -21.2%	**31.5** -9.0%	**31.1** +1.0%	35.3 -1.7%	**5.0** -67.5%	**54.5** -3.0%	21.2 -32.9%	10.5 -26.1%	7.3 -70.7%	36.8 -25.4%	**30.3** -20.3%	36.3 -13.6%	**44.6** -9.9%	10.7 -55.4%	42.5 -8.6%	47.3 -1.3%	46.4 -10.1%	34.0 -6.3%	51.6 -9.6%	43.4 -10.9%	**52.5** -5.4%	50.6 -15.9%
2D Pool 2x4 Max Pooling	**59.6** -12.7%	**47.3** -14.9%	76.4 -9.7%	**1379.3** -15.8%	**68.0** -9.2%	**46.6** -19.8%	31.2 -9.8%	31.0 +0.6%	**36.0** +0.3%	4.7 -69.5%	53.0 -5.7%	**22.3** -29.4%	**11.0** -22.5%	**9.7** -61.0%	**38.7** -21.5%	29.7 -21.8%	**38.9** -7.4%	43.9 -11.3%	**20.7** -13.8%	**43.5** -6.5%	**47.6** -0.6%	**47.1** -8.7%	**35.0** -3.6%	**52.6** -7.9%	**43.8** -10.1%	52.4 -5.6%	**51.6** -14.3%
2D Pool 4x4 VP+MP	60.9 -10.8%	48.0 -13.7%	78.7 -7.0%	1363.4 -16.8%	69.2 -7.6%	47.7 -17.9%	**31.7** -8.4%	**31.8** +3.2%	**36.4** +1.4%	5.4 -64.9%	53.3 -5.2%	22.3 -29.4%	10.8 -23.9%	12.6 -49.4%	39.5 -19.9%	26.4 -30.5%	38.1 -9.3%	**42.2** -14.7%	20.7 -13.8%	42.0 -9.7%	47.5 -0.9%	46.5 -9.9%	33.3 -8.3%	51.4 -10.0%	**43.4** -10.9%	**52.0** -6.3%	52.4 -13.0%
2D Pool 4x4 Max Pooling	**61.5** -10.0%	**48.1** -13.5%	**79.1** -6.5%	**1459.0** -10.9%	**69.9** -6.7%	**49.0** -15.7%	31.4 -9.2%	31.0 +0.6%	35.3 -1.7%	**5.6** -63.6%	**53.7** -4.4%	**24.7** -21.8%	**11.0** -22.5%	**15.3** -38.6%	**40.4** -18.1%	**31.9** -16.1%	**38.3** -8.8%	40.8 -17.6%	**22.0** -8.3%	**42.9** -7.7%	**47.6** -0.7%	**46.9** -9.1%	**34.3** -5.5%	**53.3** -6.7%	41.5 -14.8%	51.9 -6.5%	**53.4** -11.3%
2D Pool 8x8 VP+MP	64.6 -5.4%	51.6 -7.2%	82.2 -2.8%	1495.6 -8.7%	**71.6** -4.4%	53.0 -8.8%	32.5 -6.1%	31.2 +1.3%	**38.4** +7.0%	5.6 -63.6%	**54.9** -2.3%	26.5 -16.1%	11.4 -19.7%	16.4 -34.1%	42.1 -14.6%	**36.2** -4.7%	38.1 -9.3%	42.5 -14.1%	19.3 -19.6%	**43.1** -7.3%	47.6 -0.6%	**49.4** -4.3%	**34.1** -6.1%	54.1 -5.3%	**46.8** -3.9%	**54.7** -1.4%	**57.1** -5.1%
2D Pool 8x8 Max Pooling	**64.9** -5.0%	**51.9** -6.7%	**82.3** -2.7%	**1556.8** -5.0%	**71.6** -4.4%	**53.7** -7.6%	**33.5** -3.2%	**31.5** +2.3%	37.7 +5.0%	**10.1** -34.4%	54.6 -2.8%	**27.6** -12.7%	**12.8** -9.9%	**21.2** -14.9%	**43.8** -11.2%	32.4 -14.7%	**39.0** -7.1%	**46.1** -6.9%	**20.0** -16.7%	42.2 -9.2%	**47.8** -0.2%	47.9 -7.2%	33.0 -9.1%	**55.1** -3.5%	43.1 -10.9%	53.4 -3.8%	54.7 -9.1%

Qwen3 0.6B with Perceiver Resampler Aggregators

The experimental outcomes for the evaluated perceiver-based configurations are detailed in the accompanying Table 4.30. The results compare the standard Qwen3 0.6B baseline against various perceiver-based visual feature extraction architectures, varying the number of latent queries - 8, 16, 64 queries - and transformer blocks - 3 and 6 blocks in 8 queries configuration.

Firstly, a clear scaling trend emerges when examining the capacity of the Perceiver module. Increasing the number of latent queries from 8 to 64 consistently yields performance improvements across nearly all evaluated domains, particularly within the general and vision categories. This suggests that expanding the information budget allocated to the LLM is crucial for mitigating information loss and preserving the fine-grained visual details required for complex multimodal reasoning.

Furthermore, the integration of pooled features in query initialization proves to be an effective

structural enhancement also in this case. While the standalone learnable query vector configuration occasionally shows marginal advantages in highly constrained setups, the pooling-based initialization approach generally stabilizes and boosts performance as the architecture scales to 16 and 64 queries.

In line with expectations, introducing severe aggregated representations - between only 1.4% and 11.1% of the original resolution - results in an overall performance penalty compared to the full-resolution Qwen3 0.6B baseline, primarily evident in fine-grained OCR and complex mathematical reasoning tasks. However, despite this architectural bottleneck, the proposed models exhibit strong capability retention, achieving highly competitive results, occasionally yielding improvements over the baseline. This demonstrates the viability of utilizing compact, query-based visual abstractions for scenarios where efficiency is paramount, without inducing a catastrophic degradation of the model's core multimodal understanding.

Table 4.30: Comparison considering Qwen3 0.6B baseline with different pooling configurations. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Qwen3 0.6B (Baseline)	65.2	53.7	84.0	1477.2	69.0	54.1	33.2	27.1	32.0	25.8	48.0	28.2	12.3	22.6	42.9	35.1	40.0	55.3	20.7	44.1	43.6	46.2	34.1	43.9	43.8	51.9	57.5
Perc. 8q 3b Identity	**55.4** -15.0%	**44.7** -16.8%	**69.5** -17.3%	**1301.5** -11.9%	63.6 -7.8%	**43.8** -19.0%	**28.4** -14.5%	27.7 +2.2%	30.7 -4.1%	**10.5** -59.3%	44.9 -6.5%	18.4 -34.8%	9.6 -22.0%	7.4 -67.3%	**33.6** -21.7%	**23.2** -33.9%	34.9 -12.8%	38.7 -30.0%	18.0 -13.0%	40.0 -9.3%	42.8 -1.8%	**42.8** -7.4%	**34.2** +0.3%	50.0 +13.9%	**38.1** -13.0%	46.1 -11.2%	**45.8** -20.3%
Perc. 8q 3b Identity + Pooled	55.2 -15.3%	44.5 -17.1%	68.5 -18.5%	1296.0 -12.3%	**64.5** -6.5%	43.5 -19.6%	27.3 -17.8%	**27.9** +3.0%	**31.2** -2.5%	4.6 -82.2%	**45.5** -5.2%	**19.0** -32.6%	**10.2** -17.1%	**10.0** -55.8%	33.0 -23.1%	22.8 -35.0%	**35.6** -11.0%	**39.4** -28.8%	**19.3** -6.8%	**40.6** -7.9%	**43.1** -1.1%	41.9 -9.3%	32.1 -5.9%	**52.6** +19.8%	34.1 -22.1%	**46.8** -9.8%	43.7 -24.0%
Perc. 8q 6b Identity	56.3 -13.7%	**45.2** -15.8%	70.8 -15.7%	**1402.3** -5.1%	64.6 -6.4%	44.5 -17.7%	**28.5** -14.2%	25.8 -4.8%	32.7 +2.2%	**10.4** -59.7%	45.1 -6.0%	19.3 -31.6%	9.8 -20.3%	10.1 -55.3%	**33.2** -22.6%	24.1 -31.3%	**36.4** -9.0%	41.4 -25.1%	**22.0** +6.3%	39.3 -10.9%	42.9 -1.6%	41.9 -9.3%	32.1 -5.9%	**50.4** +14.8%	36.0 -17.8%	46.1 -11.2%	45.0 -21.7%
Perc. 8q 6b Identity + Pooled	**57.1** -12.4%	**45.2** -15.8%	**71.5** -14.9%	1327.0 -10.2%	**66.5** -3.6%	**45.1** -16.6%	27.9 -16.0%	**27.8** +2.6%	**33.0** +3.1%	4.6 -82.2%	**46.4** -3.3%	**21.1** -25.2%	**10.1** -17.9%	**13.3** -41.2%	**33.2** -22.6%	**27.7** -21.1%	36.2 -9.5%	**41.7** -24.6%	18.7 -9.7%	**41.2** -6.6%	**43.2** -0.9%	**43.0** -6.9%	**33.4** -2.1%	49.6 +13.0%	**38.1** -13.0%	**47.9** -7.7%	**46.0** -20.0%
Perc. 16q 6b Identity	57.0 -12.6%	**45.7** -14.9%	72.2 -14.0%	**1338.9** -9.4%	64.8 -6.1%	45.5 -15.9%	27.8 -16.3%	27.6 +1.8%	33.1 +3.4%	**6.5** -74.8%	44.1 -8.1%	19.5 -30.9%	9.4 -23.6%	6.3 -72.1%	**34.5** -19.6%	27.6 -21.4%	35.7 -10.7%	**41.3** -25.3%	**19.3** -6.8%	39.1 -11.3%	43.1 -1.1%	43.4 -6.1%	**32.5** -4.7%	**51.2** +16.6%	39.8 -9.1%	46.4 -10.6%	47.1 -18.1%
Perc. 16q 6b Identity + Pooled	**57.4** -12.0%	45.3 -15.6%	**72.6** -13.6%	1293.4 -12.4%	**65.9** -4.5%	**45.8** -15.3%	**28.2** -15.1%	**27.8** +2.6%	**33.9** +5.9%	4.9 -81.0%	**46.2** -3.7%	**20.3** -28.0%	**9.9** -19.5%	**8.1** -64.2%	33.0 -23.1%	**30.2** -14.0%	**36.0** -10.0%	**41.3** -25.3%	**19.3** -6.8%	**40.2** -8.8%	**43.3** -0.7%	**44.6** -3.5%	31.7 -7.0%	51.0 +16.2%	**40.1** -8.4%	**47.9** -7.7%	**52.2** -9.2%
Perc. 64q 6b Identity	57.6 -11.7%	45.8 -14.7%	**73.6** -12.4%	1331.6 -9.9%	64.5 -6.5%	**46.6** -13.9%	**28.3** -14.8%	**27.7** +2.2%	30.1 -5.9%	**8.9** -65.5%	**46.4** -3.3%	**20.8** -26.2%	**10.0** -18.7%	**9.7** -57.1%	**33.9** -21.0%	**29.5** -16.0%	**37.4** -6.5%	**44.2** -20.1%	**20.0** -3.4%	**42.2** -4.3%	43.2 -0.9%	43.4 -6.1%	**33.4** -2.1%	51.2 +16.6%	39.2 -10.5%	46.8 -9.8%	46.2 -19.7%
Perc. 64q 6b Identity + Pooled	**58.0** -11.0%	**46.2** -14.0%	72.9 -13.2%	**1360.2** -7.9%	**66.1** -4.2%	**46.6** -13.9%	27.6 -16.9%	27.0 -0.4%	**31.1** -2.8%	6.6 -74.4%	45.9 -4.4%	19.8 -29.8%	**10.0** -18.7%	7.7 -65.9%	33.6 -21.7%	27.9 -20.5%	35.8 -10.5%	43.0 -22.2%	16.7 -19.3%	40.2 -8.8%	**43.5** -0.2%	**44.5** -3.7%	33.3 -2.3%	**52.2** +18.9%	**39.8** -9.1%	**47.3** -8.9%	**49.7** -13.6%

To further investigate the training dynamics and architectural choices of the Perceiver module, additional experiments were conducted by removing the orthogonality constraint on the initialization of the latent queries. The updated table includes two novel configurations under the 8-query, 6-block setting: one removes the orthogonality constraint imposed during perceiver initialization, and the other also employs a higher learning rate - i.e., $1 \cdot 10^{-3}$ - for the aggregator and a smaller learning rate - i.e., $5 \cdot 10^{-4}$ - for the MLP adapter during the first stage of LLaVA

training.

Results are reported in Table 4.31 and show that removing the orthogonality constraint leads to a performance degradation in several benchmarks, but improvements in others. Instead, the newer learning rate configuration is able to improve the performance, outperforming the baseline in some benchmarks.

This could be explained by the fact that the orthogonality constraint is a regularization technique which can help to prevent overfitting and improve generalization, but it could make difficult for the model to learn how to move starting vectors in the latent space toward the most relevant information from the original visual features. Increasing the learning rate for the perceiver resampler and decreasing it for the MLP adapter could help to learn better how to extract the most relevant information from the original visual features, while preventing overfitting in the MLP adapter, which is a small component with a limited number of parameters that could be prone to early overfitting on not yet aligned representations provided by the perceiver.

Table 4.31: Comparison considering Qwen3 0.6B baseline with different configurations without orthogonal initialization. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Qwen3 0.6B (Baseline)	65.2	53.7	84.0	1477.2	69.0	54.1	33.2	27.1	32.0	25.8	48.0	28.2	12.3	22.6	42.9	35.1	40.0	55.3	20.7	44.1	43.6	46.2	34.1	43.9	43.8	51.9	57.5
Perc. 8q 6b Identity	56.3	**45.2**	70.8	**1402.3**	64.6	44.5	28.5	25.8	32.7	**10.4**	45.1	19.3	9.8	10.1	33.2	24.1	36.4	41.4	22.0	39.3	42.9	41.9	32.1	50.4	36.0	46.1	45.0
	-13.7%	-15.8%	-15.7%	-5.1%	-6.4%	-17.7%	-14.2%	-4.8%	+2.2%	-59.7%	-6.0%	-31.6%	-20.3%	-55.3%	-22.6%	-31.3%	-9.0%	-25.1%	+6.3%	-10.9%	-1.6%	-9.3%	-5.9%	+14.8%	-17.8%	-11.2%	-21.7%
Perc. 8q 6b Identity + Pooled	**57.1**	**45.2**	**71.5**	1327.0	**66.5**	45.1	27.9	**27.8**	33.0	4.6	46.4	**21.1**	**10.1**	13.3	33.2	**27.7**	36.2	41.7	18.7	41.2	43.2	43.0	**33.4**	49.6	38.1	**47.9**	46.0
	-12.4%	-15.8%	-14.9%	-10.2%	-3.6%	-16.6%	-16.0%	+2.6%	+3.1%	-82.2%	-3.3%	-25.2%	-17.9%	-41.2%	-22.6%	-21.1%	-9.5%	-24.6%	-9.7%	-6.6%	-0.9%	-6.9%	-2.1%	+13.0%	-13.0%	-7.7%	-20.0%
Perc. 8q 6b Id. + High LR + No Orth.	55.3	44.6	68.3	1294.5	64.8	43.4	**29.0**	27.4	**33.0**	8.8	**47.0**	20.8	9.8	**14.8**	32.6	26.1	**37.4**	41.4	**23.3**	**41.4**	**43.5**	43.2	31.3	**51.8**	**39.5**	46.7	46.5
	-15.2%	-16.9%	-18.7%	-12.4%	-6.1%	-19.8%	-12.7%	+1.1%	+3.1%	-65.9%	-2.1%	-26.2%	-20.3%	-34.5%	-24.0%	-25.6%	-6.5%	-25.1%	+12.6%	-6.1%	-0.2%	-6.5%	-8.2%	+18.0%	-9.8%	-10.0%	-19.1%
Perc. 8q 6b MLP + No Orth.	56.7	44.8	**71.5**	1324.5	65.1	**45.3**	27.0	27.2	31.0	6.2	43.6	19.0	9.2	7.6	**34.0**	25.2	36.3	**43.5**	20.0	39.2	42.6	**43.3**	33.0	50.7	37.8	47.4	**47.7**
	-13.0%	-16.6%	-14.9%	-10.3%	-5.7%	-16.3%	-18.7%	+0.4%	-3.1%	-76.0%	-9.2%	-32.6%	-25.2%	-66.4%	-20.7%	-28.2%	-9.3%	-21.3%	-3.4%	-11.1%	-2.3%	-6.3%	-3.2%	+15.5%	-13.7%	-8.7%	-17.0%

Qwen3 1.7B with Perceiver Resampler Aggregators

Similar to the pooling-based approach, perceiver-based aggregators are also evaluated with the larger Qwen3 1.7B. The results presented in the subsequent Table 4.32 reveal how an increased language modeling capacity interacts with the proposed visual compression techniques. Most notably, the larger base model exhibits greater resilience in knowledge-intensive domains. Unlike the 0.6B variant, several configurations here effectively close the gap with the baseline - and occasionally surpass it - in specific reasoning tasks. This indicates that a more robust language model can better compensate for highly abstracted visual inputs by relying on its broader internal

knowledge base.

Furthermore, the architectural dynamics between the mapping strategies shift at this scale. While the pooling-based query initialization approach previously acted as a consistent stabilizer for the smaller network, the standalone learnable query vector configuration demonstrates superior performance in the general and knowledge categories when paired with the 1.7B model, particularly at the highest query capacity. This suggests that a more capable LLM is sufficiently adept at interpreting the raw latent queries directly, rendering the additional global context provided by pooling less critical outside of strictly vision-centric or OCR-based tasks.

Table 4.32: Comparison considering Qwen3 1.7B baseline with different pooling configurations. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Qwen3 1.7B (Baseline)	68.3	55.6	84.6	1638.2	74.9	58.1	34.6	30.8	35.9	15.4	56.2	31.6	14.2	24.9	49.3	38.0	42.0	49.5	24.0	46.5	47.9	51.6	36.3	57.1	48.7	55.5	60.2
Perc. 8q 3b	58.8	46.0	73.9	1380.7	68.1	**47.2**	**33.0**	**31.7**	**36.1**	**10.9**	53.1	22.6	**11.8**	11.5	**38.8**	28.5	36.7	42.2	16.0	**42.5**	46.1	45.3	34.2	**52.5**	40.6	50.7	48.6
Identity	-13.9%	-17.3%	-12.6%	-15.7%	-9.1%	-18.8%	-4.6%	+2.9%	+0.6%	-29.2%	-5.5%	-28.5%	-16.9%	-53.8%	-21.3%	-25.0%	-12.6%	-14.7%	-33.3%	-8.6%	-3.8%	-12.2%	-5.8%	-8.1%	-16.6%	-8.6%	-19.3%
Perc. 8q 3b	**59.4**	**46.2**	**75.6**	**1441.9**	**68.6**	**47.2**	31.7	31.2	35.8	6.0	**53.8**	**23.6**	10.2	**13.4**	**38.8**	**32.0**	**37.4**	**43.0**	**21.3**	38.6	**46.5**	**46.7**	**34.7**	52.4	**42.3**	**52.3**	**51.8**
Identity + Pooled	-13.0%	-16.9%	-10.6%	-12.0%	-8.4%	-18.8%	-8.4%	+1.3%	-0.3%	-61.0%	-4.3%	-25.3%	-28.2%	-46.2%	-21.3%	-15.8%	-11.0%	-13.1%	-11.2%	-17.0%	-2.9%	-9.5%	-4.4%	-8.2%	-13.1%	-5.8%	-14.0%
Perc. 8q 6b	**60.1**	**47.1**	**76.8**	**1457.7**	**68.9**	**47.7**	31.7	31.3	**37.0**	5.6	52.9	23.8	11.0	**15.2**	**39.4**	29.4	**37.2**	**42.2**	**18.0**	**41.7**	46.8	45.0	**35.7**	51.1	**40.6**	49.9	47.8
Identity	-12.0%	-15.3%	-9.2%	-11.0%	-8.0%	-17.9%	-8.4%	+1.6%	+3.1%	-63.6%	-5.9%	-24.7%	-22.5%	-39.0%	-20.1%	-22.6%	-11.4%	-14.7%	-25.0%	-10.3%	-2.3%	-12.8%	-1.7%	-10.5%	-16.6%	-10.1%	-20.6%
Perc. 8q 6b	59.5	46.6	75.5	1391.8	68.6	47.2	**31.8**	**32.4**	35.7	**5.7**	**53.2**	**25.0**	**11.3**	15.1	39.0	**34.6**	35.5	41.3	15.3	38.3	**47.1**	**45.8**	34.6	**52.1**	**40.6**	**51.5**	**50.4**
Identity + Pooled	-12.9%	-16.2%	-10.8%	-15.0%	-8.4%	-18.8%	-8.1%	+5.2%	-0.6%	-63.0%	-5.3%	-20.9%	-20.4%	-39.4%	-20.9%	-8.9%	-15.5%	-16.6%	-36.2%	-17.6%	-1.7%	-11.2%	-4.7%	-8.8%	-16.6%	-7.2%	-16.3%
Perc. 16q 6b	**61.2**	**47.7**	**77.5**	1419.0	**70.7**	**48.9**	**31.6**	**32.1**	**36.4**	4.5	53.4	24.4	10.3	15.1	39.7	32.7	37.1	43.3	16.7	41.4	47.1	46.5	**34.9**	**54.7**	41.4	50.6	50.8
Identity	-10.4%	-14.2%	-8.4%	-13.4%	-5.6%	-15.8%	-8.7%	+4.2%	+1.4%	-70.8%	-5.0%	-22.8%	-27.5%	-39.4%	-19.5%	-13.9%	-11.7%	-12.5%	-30.4%	-11.0%	-1.7%	-9.9%	-3.9%	-4.2%	-15.0%	-8.8%	-15.6%
Perc. 16q 6b	60.4	46.9	76.7	**1494.8**	69.4	48.7	31.5	32.0	34.6	**5.6**	**53.8**	**25.5**	**11.2**	**16.7**	**40.0**	**34.1**	**38.2**	**43.9**	**19.3**	**42.4**	**47.3**	**47.0**	34.3	53.0	**43.1**	**51.4**	**53.2**
Identity + Pooled	-11.6%	-15.6%	-9.3%	-8.8%	-7.3%	-16.2%	-9.0%	+3.9%	-3.6%	-63.6%	-4.3%	-19.3%	-21.1%	-32.9%	-18.9%	-10.3%	-9.0%	-11.3%	-19.6%	-8.8%	-1.3%	-8.9%	-5.5%	-7.2%	-11.5%	-7.4%	-11.6%
Perc. 64q 6b	**62.2**	**49.4**	78.3	1424.0	**70.0**	**51.0**	**33.4**	31.4	33.6	**14.9**	**53.6**	**25.6**	**11.3**	**17.7**	**40.5**	32.8	37.2	41.6	16.0	**44.0**	47.4	**47.7**	**35.0**	53.4	**43.4**	**52.3**	**54.4**
Identity	-8.9%	-11.2%	-7.4%	-13.1%	-6.5%	-12.2%	-3.5%	+1.9%	-6.4%	-3.2%	-4.6%	-19.0%	-20.4%	-28.9%	-17.8%	-13.7%	-11.4%	-16.0%	-33.3%	-5.4%	-1.0%	-7.6%	-3.6%	-6.5%	-10.9%	-5.8%	-9.6%
Perc. 64q 6b	61.2	47.1	**78.5**	**1424.4**	69.9	49.1	31.2	**31.6**	**34.7**	5.9	52.7	25.3	10.9	17.2	39.8	**33.4**	**38.9**	**43.5**	**22.0**	42.5	**47.5**	45.5	32.7	**53.8**	39.6	49.6	52.0
Identity + Pooled	-10.4%	-15.3%	-7.2%	-13.1%	-6.7%	-15.5%	-9.8%	+2.6%	-3.3%	-61.7%	-6.2%	-19.9%	-23.2%	-30.9%	-19.3%	-12.1%	-7.4%	-12.1%	-8.3%	-8.6%	-0.8%	-11.8%	-9.9%	-5.8%	-18.7%	-10.6%	-13.6%

4.2.5 Comparison with OC-VTP

To provide a more comprehensive understanding of the proposed architectures' efficacy, a detailed comparative analysis against the established OC-VTP baseline is warranted. The data unequivocally demonstrates that relying on learned latent queries or deterministic spatial aggregation offers a fundamentally more robust representation than discrete token-selection mechanisms, especially under severe compression regimes.

The proposed approach, which relies on pooling-based and perceiver-based aggregators, outperforms OC-VTP in almost all benchmarks (Table 4.33). This demonstrates that extracting and aggregating information from the original visual features is superior to the selection-based

technique used by OC-VTP. By condensing these features into dense visual concepts rather than merely selecting a subset, the proposed method provides the LLM with richer representations that is more effectively utilized for downstream tasks.

Table 4.33: Comparison considering Qwen3 0.6B with different configurations and OC-VTP [11]. Best values among groups of models are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	$MMB_{(EN)}$	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V^*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.2	53.7	84.0	1477.2	69.0	54.1	33.2	27.1	32.0	25.8	48.0	28.2	12.3	22.6	42.9	35.1	40.0	55.3	20.7	44.1	43.6	46.2	34.1	43.9	43.8	51.9	57.5
OC-VTP	56.1	44.1	**72.9**	1279.6	62.9	44.6	27.3	27.5	30.3	**5.9**	45.4	**22.4**	9.9	12.4	32.8	**34.6**	36.2	43.1	18.7	40.4	42.7	**44.0**	32.7	**50.4**	**39.5**	**48.6**	**48.9**
8 Slots	-14.0%	-17.9%	-13.2%	-13.4%	-8.8%	-17.6%	-17.8%	+1.5%	-5.3%	-77.1%	-5.4%	-20.6%	-19.5%	-45.1%	-23.5%	-1.4%	-9.5%	-22.1%	-9.7%	-8.4%	-2.1%	-4.8%	-4.1%	+14.8%	-9.8%	-6.4%	-15.0%
Perceiver 8 Q.	**57.1**	45.2	71.5	1327.0	**66.5**	**45.1**	27.9	**27.8**	33.0	4.6	**46.4**	21.1	**10.1**	**13.3**	33.2	27.7	36.2	41.7	18.7	41.2	43.2	43.0	**33.4**	49.6	38.1	47.9	46.0
With Pooled Init.	-12.4%	-15.8%	-14.9%	-10.2%	-3.6%	-16.6%	-16.0%	+2.6%	+3.1%	-82.2%	-3.3%	-25.2%	-17.9%	-41.2%	-22.6%	-21.1%	-9.5%	-24.6%	-9.7%	-6.6%	-0.9%	-6.9%	-2.1%	+13.0%	-13.0%	-7.7%	-20.0%
Max Pooling	56.9	**46.5**	71.9	**1358.0**	64.4	44.7	**28.0**	26.4	**34.9**	5.7	44.9	20.4	9.7	9.4	**33.5**	29.2	**40.0**	**50.2**	**24.7**	**41.5**	**43.7**	41.7	32.6	43.3	38.9	47.6	46.0
2×4	-12.7%	-13.4%	-14.4%	-8.1%	-6.7%	-17.4%	-15.7%	-2.6%	+9.1%	-77.9%	-6.5%	-27.7%	-21.1%	-58.4%	-21.9%	-16.8%	0.0%	-9.2%	+19.3%	-5.9%	+0.2%	-9.7%	-4.4%	-1.4%	-11.2%	-8.3%	-20.0%
OC-VTP	43.4	36.5	57.2	1122.1	45.1	34.7	27.6	26.2	32.2	5.3	**46.7**	18.9	9.2	2.3	28.4	**35.9**	35.0	40.4	16.0	**40.6**	42.9	38.9	28.3	47.0	36.3	44.6	38.1
16 Slots	-33.4%	-32.0%	-31.9%	-24.0%	-34.6%	-35.9%	-16.9%	-3.3%	+0.6%	-79.5%	-2.7%	-33.0%	-25.2%	-89.8%	-33.8%	+2.3%	-12.5%	-26.9%	-22.7%	-7.9%	-1.6%	-15.8%	-17.0%	+7.1%	-17.1%	-14.1%	-33.7%
Perceiver 16 Q.	57.4	45.3	**72.6**	**1293.4**	**65.9**	45.8	**28.2**	**27.8**	**33.9**	4.9	46.2	**20.3**	9.9	**8.1**	33.0	30.2	36.0	41.3	**19.3**	40.2	43.3	**44.6**	**31.7**	**51.0**	**40.1**	47.9	**52.2**
With Pooled Init.	-12.0%	-15.6%	-13.6%	-12.4%	-4.5%	-15.3%	-15.1%	+2.6%	+5.9%	-81.0%	-3.7%	-28.0%	-19.5%	-64.2%	-23.1%	-14.0%	-10.0%	-25.3%	-6.8%	-8.8%	-0.7%	-3.5%	-7.0%	+16.2%	-8.4%	-7.7%	-9.2%
Max Pooling	**57.6**	**46.8**	**72.6**	1287.4	64.6	**46.3**	28.1	27.6	33.7	**5.8**	45.2	19.9	**10.0**	6.8	**34.9**	27.8	**37.3**	**51.2**	14.0	40.5	**43.4**	40.4	30.3	40.5	36.3	**48.3**	46.6
4×4	-11.7%	-12.8%	-13.6%	-12.8%	-6.4%	-14.4%	-15.4%	+1.8%	+5.3%	-77.5%	-5.8%	-29.4%	-18.7%	-69.9%	-18.6%	-20.8%	-6.8%	-7.4%	-32.4%	-8.2%	-0.5%	-12.6%	-11.1%	-7.7%	-17.1%	-6.9%	-19.0%
OC-VTP	57.5	44.6	74.5	1296.0	64.5	46.4	27.0	26.9	30.8	6.4	43.7	**23.9**	**10.4**	**15.6**	34.0	**35.8**	35.7	41.8	**18.0**	40.0	43.1	43.3	**33.7**	51.0	35.7	**48.4**	47.9
64 Slots	-11.8%	-16.9%	-11.3%	-12.3%	-6.5%	-14.2%	-18.7%	-0.7%	-3.7%	-75.2%	-9.0%	-15.2%	-15.4%	-31.0%	-20.7%	+2.0%	-10.7%	-24.4%	-13.0%	-9.3%	-1.1%	-6.3%	-1.2%	+16.2%	-18.5%	-6.7%	-16.7%
Perceiver 64 Q.	57.9	46.2	72.9	**1360.2**	66.1	46.6	27.6	27.0	31.1	6.6	45.9	19.8	10.0	7.7	33.6	27.9	35.9	43.0	16.7	**40.2**	**43.5**	44.5	33.3	**52.2**	**39.8**	47.3	49.7
With Pooled Init.	-11.2%	-14.0%	-13.2%	-7.9%	-4.2%	-13.9%	-16.9%	-0.4%	-2.8%	-74.4%	-4.4%	-29.8%	-18.7%	-65.9%	-21.7%	-20.5%	-10.3%	-22.2%	-19.3%	-8.8%	-0.2%	-3.7%	-2.3%	+18.9%	-9.1%	-8.9%	-13.6%
Max Pooling	**60.0**	**49.4**	**75.5**	1355.9	**66.4**	**48.5**	**29.9**	**27.3**	**32.2**	**14.3**	**46.0**	22.9	10.3	10.8	**37.8**	32.9	**37.6**	**52.0**	**18.0**	37.7	42.8	42.7	31.1	42.9	38.4	**48.4**	**52.7**
8×8	-8.0%	-8.0%	-10.1%	-8.2%	-3.8%	-10.4%	-9.9%	+0.7%	+0.6%	-44.6%	-4.2%	-18.8%	-16.3%	-52.2%	-11.9%	-6.3%	-6.0%	-6.0%	-13.0%	-14.5%	-1.8%	-7.6%	-8.8%	-2.3%	-12.3%	-6.7%	-8.3%

Despite the fact that OC-VTP is pre-trained on the MS COCO dataset, it is able to reach slightly better performance than proposed approach in related benchmarks only in the 8 queries configuration, while in the 16 and 64 queries configurations the proposed approach is able to outperform OC-VTP also in the most favorable task for this model.

These results demonstrate the effectiveness of training the aggregation components in an end-to-end manner with the LLM - which is able to provide a stronger training signal to the aggregation components compared to a proxy objective in a pre-training phase - allowing them to learn how to extract the most relevant information from the original visual features and how to aggregate it into dense visual concepts, which can be then effectively used by the LLM to perform downstream tasks.

4.2.6 Training and Inference Time Comparison

In previous sections, several experiments are conducted to evaluate the effectiveness of the proposed approach with different LLMs and different aggregation components. In this section,

a comparison of training and inference time is provided, in order to evaluate the computational efficiency of the proposed approach compared to the other architectures.

In particular, the max pooling-based aggregator with a final resolution of 2×4 and the perceiver resampler with 8 queries (without pooling initialization) are considered against the standard LLaVA architecture with 576 visual features and OC-VTP [11] with the same number of slots. For all these architectures, the same vision encoder (CLIP-ViT-Large-336) and same LLM (Qwen3 0.6B) are considered, while all other hyperparameters are the same as those of the best variant described in previous section with only one MLP after the perceiver resampler for perceiver-based aggregation and two MLPs for the pooling-based aggregation.

Training time is evaluated considering the wall clock time needed to obtain the final model. For standard LLaVA and proposed approach the training time is composed of the time needed to train both stage 1 and stage 2, while for OC-VTP is also considered the time needed to pre-train the Slot Attention aggregator, given that it is not ensured that a standard LLaVA pre-trained checkpoint with desired configuration is always available and in any case it must be trained from scratch at least once.

Model	**Stage 1**	**Stage 2**
Qwen3 0.6B	40 minutes	20 hours
Qwen3 0.6B Perceiver 8 Queries	12 minutes -70.0%	11 hours and 10 minutes -44.6%
Qwen3 0.6B Max Pooling 2×4	11 minutes -72.5%	10 hours and 20 minutes -48.3%

Table 4.34: Training time comparison considering standard LLaVA Qwen3 0.6B baseline with different configurations training on 8 GPUs NVIDIA A100 (64Gb). The training time is evaluated considering the wall clock time in stage 1 and stage 2.

Results obtained training with 8 GPUs NVIDIA A100 (64Gb) are reported in Table 4.34 and show that the proposed approach is able to reduce the training time significantly compared to standard LLaVA architecture, given that the number of visual features is reduced from 576 to 8 (approximately 1.4% of the original), which reduces the training time of both stage 1 and stage 2. In particular, in the first stage, it is possible to train the model in less than a third of the time needed for standard LLaVA architecture, while in the second stage the training time is reduced almost by half - due to the fixed amount of time needed to train the LLM with *gradient*

checkpointing technique, which is used in the same way for both architectures.

Moreover, if OC-VTP is considered, the training time gain is even more significant, given that in addition to standard LLaVA first and second stages, the pre-training of the Slot Attention aggregator is needed, which requires a small but non-negligible amount of time: approximately a quarter of an hour with 8 GPUs NVIDIA A100 (64Gb).

It is important to note that the times reported above were obtained considering a very small LLM with only 0.6B parameters. When larger models are considered - such as Qwen3 1.7B or Vicuna 7B - the training time savings are expected to be even more significant. This is because the computational overhead will be increasingly dominated by the LLM itself, while the training time of our aggregation modules - which are small and lightweight - will remain virtually unchanged. Similarly, variants with additional operations such as pooling-based initialization of queries will not significantly increase the training time, given that they are small and lightweight operations compared to the LLM and vision encoder, making them nearly negligible.

Inference time could be considered more important than training time in many scenarios, given that it is the time needed to obtain the final output of the model, which is crucial in real-time applications or when a large number of samples must be processed.

In this case, inference time is evaluated 10 times with different batch sizes between 1 and 1,028 on a single NVIDIA A100 (64Gb). Results are reported in Figure 4.12 and Table 4.35 considering 1) the standard LLaVA architecture with 576 visual features; 2) the OC-VTP with 8 slots; 3) the max pooling-based aggregator with a final resolution of 2×4; and 4) the perceiver resampler with 8 queries (without pooling initialization). All these architectures are trained with the same vision encoder (CLIP-ViT-Large-336) and same LLM (Qwen3 0.6B), while all other hyperparameters are the same as those of the best variant described in previous sections.

Standard LLaVA and OC-VTP do not have values for all batch sizes due to the fact that they require a large amount of GPU memory, which is not available in a single NVIDIA A100 (64Gb). In particular, standard LLaVA architecture is able to process a maximum batch size of 128, while OC-VTP is able to process a maximum batch size of 512. On the other hand, max pooling-based aggregator and perceiver resampler are able to process all batch sizes up to 1,028.

The metrics considered for the comparison are: 1) samples per second, which is the number of samples processed in one second; 2) average time to first token in milliseconds, which is the time needed to obtain the first output token; 3) average prefill time in teraflops, which is the amount of elaboration needed to process the input (before actual generation phase); 4) average

latency per batch in milliseconds, which is the time needed to process a batch of samples and obtain the final output.

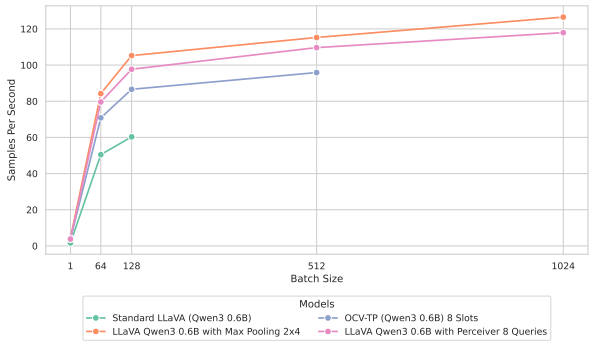
Model	Batch Size	Samples Per Second	Avg TTFT (ms)	Avg Prefill TFLOPS	Avg Latency Per Batch (ms)
Qwen3 0.6B	1	1.65	89.66	25.78	604.39
	64	50.47	874.83	169.67	1254.14
	128	60.31	1543.55	192.68	2099.04
Qwen3 0.6B Perceiver 8 Queries	1	3.80 +129.7%	85.16 -5.0%	2.05 -92.1%	263.11 -56.5%
	64	79.59 +57.7%	424.13 -51.5%	31.45 -81.5%	795.30 -36.6%
	128	97.69 +62.0%	710.36 -54.0%	38.36 -80.1%	1295.92 -38.3%
	512	109.65	1664.70	40.92	2886.42
	1024	117.91	3057.13	44.57	5368.35
Qwen3 0.6B Max Pooling 2×4	1	4.37 +164.0%	71.74 -20.0%	2.27 -91.2%	228.98 -62.1%
	64	84.23 +66.9%	404.12 -53.8%	30.86 -81.8%	751.51 -40.1%
	128	105.23 +74.5%	665.07 -56.9%	38.31 -80.1%	1203.05 -42.7%
	512	115.25	1583.11	40.24	2746.12
	1024	126.52	2877.22	44.28	5002.99
Qwen3 0.6B OCV-TP 8 Slots	1	3.81 +130.1%	88.04 -1.8%	26.25 +1.8%	262.66 -56.5%
	64	70.82 +40.3%	502.33 -42.6%	295.50 +74.2%	893.84 -28.7%
	128	86.60 +43.6%	835.43 -45.9%	355.99 +84.8%	1461.88 -30.4%
	512	95.87	1960.81	379.19	3301.49

Table 4.35: Inference time comparison considering standard LLaVA Qwen3 0.6B baseline with different configurations across various batch sizes. Relative percentage gaps are computed with respect to the baseline performance for the corresponding batch size.

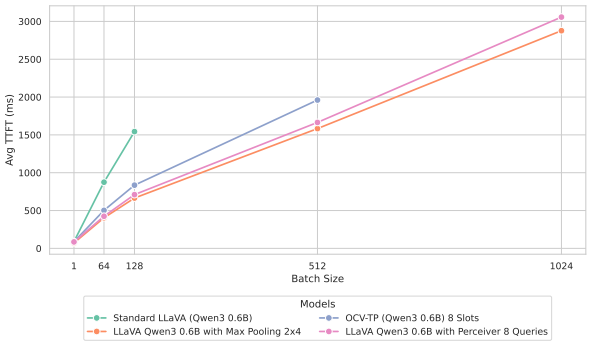
Results show that the proposed approach is able to reduce the inference time and resources significantly compared to standard LLaVA architecture and OC-VTP, given that the number of visual features is extremely reduced from 576 to 8 (approximately 1.4% of the original), which reduces the amount of elaboration needed to process the input and obtain the final output. This is particularly evident in the average prefill time in teraflops, which is less than a quarter of the one needed for standard LLaVA architecture when batch size is equal to 128 and less than seven times compared to OC-VTP when batch size is equal to 512 - due to the more complex and time consuming nature of Slot Attention compared to simpler aggregators.

These definitive results clearly illustrate the true potential and effectiveness of the proposed

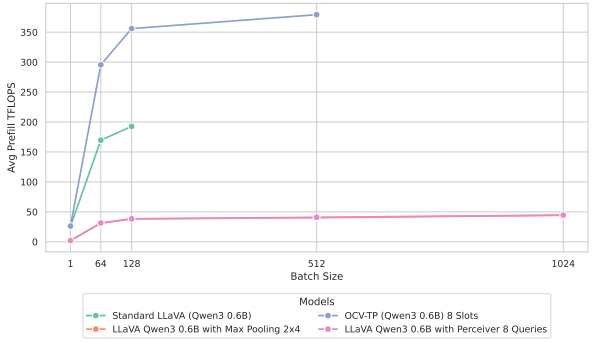
approach. In practice, this method drastically curtails the total inference execution time and optimizes the computational resources that are inherently required to ingest the input data and deliver the final output. Securing these performance enhancements is crucial for the deployment of such models, particularly in real-time operational environments where low latency is essential, or in situations characterized by big data demands where a vast number of samples must be processed systematically.



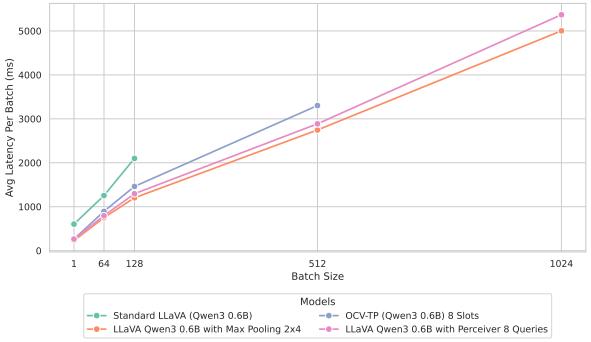
(a) Comparison of the inference time in terms of samples per second.



(b) Comparison of the average time to first token in milliseconds.



(c) Comparison of the average prefill time in teraflops.



(d) Comparison of the average latency per batch in milliseconds.

Figure 4.12: LLaVA Qwen3 0.6B-based inference time comparison of different approaches.

4.3 Experiments on Text-guided Vision Aggregation

Standard LLaVA stage 1 and stage 2 datasets are very effective to train MLLMs which have vision-only components such as a vision adapter or a vision aggregator. In fact, the first stage is used to train the vision part of the model through an image captioning task, in order to make it able to understand visual information and produce visual concepts that can be effectively used by the LLM to perform downstream tasks. The second stage is used to fine-tune the LLM - in

order to make it able to effectively leverage the visual concepts - and the vision part to work together in a coordinated manner.

As already mentioned, the first stage is based on a image-caption pairs dataset, so there are no specific or complex questions to answer, but the model is trained to generate a caption describing the image starting only from the image. Given that no input text is involved in this stage, it is reasonable to think that text-related components do not benefit from this training phase as it is.

Nevertheless, plugging text-related components randomly initialized into the pipeline directly in the second stage - when the LLM is trainable - can lead to a strong degradation of the model capability, due to the ingestion of random noise that the LLM must compensate by modifying its weights.

A possible way to mitigate this problem is to use synthetic generic questions - such as *"What is in this picture?"* or *"Describe this image"* - in order to have a signal also for the text-related components. However, given that all these questions have the same semantics, it is reasonable to think that they do not provide a strong signal for the text-related components, which are not trained to effectively leverage the input text to extract the most relevant information from the image, but they are just trained to produce a caption describing the image without considering the input question. This could lead to "turning off" - *weight dying problem* - the text-related components in the first stage - where the learning rate, so the learning capability, is higher - which difficulty can be recovered in the second stage, where the learning rate is noticeably lower.

Alternatively, a more text-oriented dataset in the first stage - to train the text-related components jointly - or more epochs in the second stage - eventually with variable learning rate - in order to give more time to the model to understand how to leverage the text-related components could be effective in solving this problem. However, in a similar way to LLaVA-Mini (Section 2.11), these solutions can be very computationally expensive, given that they require training the model for more epochs, which can be a problem especially when the model is large and the computational resources are limited.

Nevertheless, in order to train models that are comparable with previously presented results, the standard LLaVA stage 1 and stage 2 datasets and hyperparameters are used in most of the following experiments. Even in this case only the frozen CLIP-ViT-Large-336 vision encoder is considered for all experiments.

4.3.1 Weights Dying Problem

Initial experiments are conducted to evaluate the weights dying problem described above - training the architecture proposed in Section 3.4 with the standard LLaVA datasets and hyper-parameters - evaluating the weight variation of the text-related components during the training process.

In particular, the variant with the set of softmaxed learnable parameters to select the best LLM layer is considered, which is trained with LLaMA 3.2 1B with learning rates illustrated in Table 4.36 - even if the perceivers have different numbers of blocks, the learning rates are the same due to the fact that text aggregator is more in depth in the architecture, needing more stable weights updates. Visual and text aggregators are both perceiver resamplers with 8 learnable queries: the text aggregator has only one block, instead the visual aggregator has 3 blocks which start from the output of the text aggregator.

To better evaluate the weights dying problem, parameters used to select the best LLM layer are initialized in several different ways: 1) all parameters are initialized with the same value; 2) parameters are initialized with a linear increasing values from 0 to 1; 3) parameters are initialized linearly decreasing values from 1 to 0; 4) parameters are initialized with a linear increasing values from 0 to 10; 5) parameters are initialized linearly decreasing values from 10 to 0. In other words, in the first case no initial bias is given to the model to select a specific layer, while in the other cases the model is biased to select a specific layer at the beginning - the first or the last ones.

In all these cases, at the end of the first training stage, focusing on those parameters the variation in magnitude is very low, with a variation of approximately between 0.1 and 0.01 regardless of the initialization strategy, which is a clear indication that the learning process is not effective for these parameters, which are basically frozen during the training process. Same experiments were also conducted with a higher learning rate and applying the softmax operation which would take to the extreme the weights, but the results are very similar, confirming that the learning process is not effective for these parameters in the first stage.

On the other hand, at the end of the second stage there is a higher variation, which is a clear indication that the learning process in this phase is more effective, being able to rely on question-answering datasets to learn how to leverage the text-related components.

These results confirm the presence of the weights dying problem described in previous section when standard LLaVA datasets are used, which prevents the model from effectively

leveraging the text-related components - which are basically frozen during the first stage.

Table 4.36: Learning rates for Stage 1 and Stage 2 when standard LLaVA datasets are used with LLaMA 3.2 1B, CLIP-ViT-Large-336, 1-block text perceiver and 3-block visual perceiver (both with 8 queries).

Component	**Stage 1**	**Stage 2**
LLM	-	2e-5
Vision Projector	1e-3	2e-5
Vision Aggregator	5e-4	2e-5
Multimodal Projector	1e-3	2e-5
Text Encoder's Modulation Weights	1e-3	2e-5
Pre-aggregation Text Projection	1e-3	2e-5
Text Aggregator	5e-4	2e-5

4.3.2 LLM Layer Selection

Given the weights dying problem described above, it is not effective to exploit a set of learnable parameters to select the best LLM layer, due to the fact that these parameters remain basically frozen during the first stage, changing very little also in the second stage.

For this reason a manual comparison between different LLM layers is proposed. LLaMA 3.2 1B is used and it has 17 layers where the first is the simple lookup embedding layer. Following the same intuition of CLIP visual features, the last layer is ignored - it is reasonably very specialized in next token prediction task - instead the penultimate layer and the middle layer are considered, which are the 16^{th} and the 8^{th} layers respectively. In both cases the rest of the architecture and hyperparameters remain the same described in Section 4.3.1, where text aggregator produces a set of 8 queries which will be the starting point for the visual aggregation process - i.e., perceiver resampler [4].

Results illustrated in Table 4.37 show that in some benchmarks the two layers have a very similar performance, while in other benchmarks there are significant differences. In particular, the 16^{th} layer is better in visual question answering, while the 8^{th} layer is better in other tasks - such as OCR and document understanding - with very polarized benchmarks that modify a lot the average performance.

This behavior can be explained by the fact that the 16^{th} layer is more specialized in next

token prediction task, producing more abstract and high-level representations, which can be more effective in visual question answering tasks, where a deeper understanding of the image is required to answer the question correctly. On the other hand, the 8th layer is less specialized in next token prediction task, producing more noun-related information, which can be more effective in OCR and document understanding tasks, where a more detailed and fine-grained understanding of the image is required to extract the relevant information.

It is interesting to notice that the 16th layer allows the model to **outperform** the standard LLaVA architecture over all vision tasks, demonstrating the importance of a guided vision aggregation approach, which provides more relevant visual information to the LLM, even if the number of visual features is extremely limited.

Moreover, results confirm the importance of a combination of different layers of the LLM, in order to give the model the possibility to understand how to leverage the different layers of the LLM to extract the most relevant information from the image, which can be effectively used by the LLM to perform downstream tasks.

Table 4.37: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. Text aggregator has 1 block with 8 queries, while visual aggregator has 3 blocks. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
LLM's 16th Layer	54.6	48.5	74.1	1108.9	**48.4**	**46.4**	37.3	61.4	**30.8**	9.8	**47.2**	28.9	**9.8**	18.0	40.1	47.7	**37.1**	**45.6**	**14.0**	**41.2**	**47.6**	46.2	28.2	52.1	48.0	**53.5**	49.4
	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.6%	+21.2%	-3.8%	-14.5%	-18.5%	-36.9%	-18.7%	+5.1%	+1.0%	-5.1%	+8.7%	+4.3%	+2.4%	-11.5%	-16.0%	-7.6%	-6.8%	-9.9%	-18.1%
LLM's 8th Layer	**55.2**	**49.2**	**75.3**	**1140.5**	48.2	46.1	**38.5**	**62.3**	29.1	**17.0**	45.7	**31.0**	9.7	**19.6**	**40.3**	**54.5**	34.3	45.4	7.3	38.9	45.5	**46.9**	**29.0**	**54.3**	**48.2**	53.2	**49.7**
	-16.3%	-15.5%	-10.2%	-13.5%	-19.8%	-24.8%	-2.6%	-6.8%	-14.7%	+109.6%	-6.9%	-8.2%	-19.4%	-31.1%	-18.2%	+20.0%	-6.7%	-5.4%	-43.5%	-1.6%	-2.1%	-10.3%	-13.8%	-3.7%	-6.4%	-10.4%	-17.6%

4.3.3 Aggregation Size

Similarly to the experiments conducted for the text-influenced vision-centric architecture, a study on the number of blocks of the perceiver-based aggregators is presented, in order to understand if a higher number of blocks can lead to a better performance.

For this experiment LLaMA 3.2 1B is used and its 16th layer is selected as input for the text aggregator - having demonstrated better performance in visual question answering benchmarks (Section 4.3.2). The aggregator used in Section 4.3.1 - with 3 blocks for the visual aggregator and 1 block for the text aggregator - is compared with a variant with double the number of blocks

- 6 blocks for the visual aggregator and 2 blocks for the text aggregator.

The most significant hyperparameter variation is the learning rate of the aggregators. Given the higher number of blocks, the learning rate must be reduced to avoid instability during training due to the higher number of parameters and the higher depth of the architecture, which can lead to a stronger vanishing gradient problem. In particular, the learning rate of the visual aggregator is reduced from 5×10^{-4} to 2×10^{-4} during the first stage for both perceivers, while they remain the same in the second stage and equal to 2×10^{-5}.

Results are reported in Table 4.38. The larger variant with 6 blocks for the visual aggregator and 2 blocks for the text aggregator is able to outperform the smaller variant with 3 blocks for the visual aggregator and 1 block for the text aggregator in most of benchmarks, **outperforming** the standard LLaVA architecture (576 visual features) in complex tasks such as VizWiz [72] and MathVista [73] - where the performance is **doubled** - only exploiting 8 visual features.

On the other hand, smaller variant is able to achieve better performance in some vision question answering benchmarks such as MMVP [74]. This could be due to the deeper text perceiver of larger variant which could degrade the text signal - that presumably was the game changer to obtain better performance in this class of benchmarks. Instead, in other tasks where pure vision information is more important, the deeper vision perceiver of larger variant is able to extract more relevant information from the image, which can be effectively used by the LLM to perform downstream tasks.

Table 4.38: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. LLM's 16$^{\text{th}}$ layer is considered to extract text features. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Text: 1 Block	54.6	48.5	**74.1**	1108.9	48.4	46.4	37.3	61.4	**30.8**	9.8	**47.2**	28.9	**9.8**	18.0	**40.1**	47.7	**37.1**	45.6	**14.0**	**41.2**	**47.6**	46.2	28.2	52.1	**48.0**	53.5	49.4
Vision: 3 Blocks	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.6%	+21.2%	-3.8%	-14.5%	-18.5%	-36.9%	-18.7%	+5.1%	+1.0%	-5.1%	+8.7%	+4.3%	+2.4%	-11.5%	-16.0%	-7.6%	-6.8%	-9.9%	-18.1%
Text: 2 Blocks	**55.4**	**49.5**	73.7	**1128.6**	**50.6**	**47.0**	**38.2**	**61.5**	27.4	**17.0**	46.9	**30.1**	9.7	**18.1**	39.4	**53.3**	34.1	**45.7**	8.4	38.3	43.9	**47.1**	**28.8**	**52.9**	46.8	**56.1**	**51.1**
Vision: 6 Blocks	-15.9%	-15.0%	-12.2%	-14.4%	-15.8%	-23.3%	-3.3%	-7.9%	-19.6%	+109.6%	-4.5%	-10.9%	-19.4%	-36.5%	-20.0%	+17.4%	-7.2%	-4.8%	-34.8%	-3.1%	-5.6%	-9.8%	-14.4%	-6.2%	-9.1%	-5.6%	-15.3%

4.3.4 Residual Connection Between Text and Vision Aggregators

The architecture variants considered in previous described experiments rely on the hypothesis that the vision aggregator is able to use as initial queries the output of the text aggregator - which

is a set of 8 learnable queries - to extract the most relevant information from the image based on the input question.

An alternative approach could rely on a residual connection after the visual aggregation process. In this way, the vision aggregation process must produce a variation which will be applied to the initial queries provided by the text aggregator. The benefit of this residual connections is the mitigation of the vanishing gradient problem during training - which could affect this pipeline due to its depth.

However, even if the residual connections are widely used in deep learning architectures, this mechanism is not always effective. In particular, results in Table 4.39 show that in the majority of benchmarks the variant without residual connection is better - having a remarkable negative gap only in MMVP and MMMU.

This can be explained by the fact that residual connection mechanism provides a sort of "refinement" of the initial values, but in this scenario the initial queries come from the text aggregator, which does not have any visual information, so the vision aggregator must produce a strong variation to extract the most relevant information from the image, which can be difficult to achieve with a residual connection that tends to preserve the initial queries. This can lead to a sort of "inertia" in the learning process, where the model struggles to learn how to effectively leverage the visual information to refine the initial queries, which can lead to a degradation of the performance in downstream tasks.

Table 4.39: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. 6 blocks for the vision aggregator and 2 blocks for the text aggregator, considering LLM's 16th layer. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
No Residual Connection	**55.4**	**49.5**	73.7	**1128.6**	**50.6**	**47.0**	**38.2**	**61.5**	27.4	**17.0**	**46.9**	**30.1**	**9.7**	**18.1**	**39.4**	**53.3**	34.1	**45.7**	8.4	38.3	43.9	**47.1**	**28.8**	**52.9**	**46.8**	**56.1**	**51.1**
	-15.9%	-15.0%	-12.2%	-14.4%	-15.8%	-23.3%	-3.3%	-7.9%	-19.6%	+109.6%	-4.5%	-10.9%	-19.4%	-36.5%	-20.0%	+17.4%	-7.2%	-4.8%	-34.8%	-3.1%	-5.6%	-9.8%	-14.4%	-6.2%	-9.1%	-5.6%	-15.3%
With Residual Connection	54.7	48.7	**75.2**	1095.7	48.6	46.3	37.7	61.2	**30.6**	12.8	46.4	29.1	9.3	17.3	38.9	51.0	**36.1**	43.3	**15.1**	**38.6**	**47.2**	45.6	27.7	50.8	44.7	53.9	**51.1**
	-16.9%	-16.3%	-10.4%	-16.9%	-19.1%	-24.4%	-4.6%	-8.4%	-10.3%	+57.7%	-5.6%	-13.8%	-22.5%	-39.2%	-21.0%	+12.3%	-1.7%	-9.7%	+17.4%	-2.2%	+1.6%	-12.7%	-17.5%	-10.0%	-13.3%	-9.2%	-15.3%

4.3.5 Pooling-based Initialization

Similarly to text-influenced vision-centric architecture (Section 3.3) a variant with a pooling-based text aggregator's queries initialization is considered to understand if providing visual information also in the text-related components can help the model to learn how to effectively leverage input question to extract the most relevant information from the image.

The considered model is based on LLaMA 3.2 1B with 6 blocks for the visual aggregator and 2 blocks for the text aggregator considered in previous sections.

The idea is to provide a visual bias to text aggregation in order to ground the text information extraction to visual concepts. For example, given a question *"What time is it?"* and an image with a *clock* on the top right, the query initialized with the top right pooled region should attend the text information about the time more than other queries.

Results in Table 4.40 show that this initialization approach does not improve the performance of the model in visual benchmarks, but it is able to improve the performance of other benchmarks.

This can be explained by the fact that pooling-based initialization provides a strong visual bias to text aggregation, achieving better performance in benchmarks where a high-level understanding of the image is required. Instead, in visual question answering benchmarks a pure text guidance is more effective, because there are complex questions - such as in MMVP or MathVista - that require a deeper understanding of textual information, which can be hindered by a strong visual bias that can represent a sort of noise for the text aggregation process, leading to a degradation of the performance in downstream tasks.

Table 4.40: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. 3-blocks vision aggregator and 1-block text aggregator, considering LLM's 16$^{\text{th}}$ layer. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	$\text{OCR}_{\text{Bench}}$	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Learnable Queries	54.6	48.5	74.1	1108.9	48.4	46.4	**37.3**	61.4	**30.8**	**9.8**	47.2	28.9	9.8	18.0	40.1	47.7	**37.1**	45.6	**14.0**	**41.2**	**47.6**	46.2	28.2	52.1	**48.0**	53.5	**49.4**
No Initialization	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.6%	+21.2%	-3.8%	-14.5%	-18.5%	-36.9%	-18.7%	+5.1%	+1.0%	-5.1%	+8.7%	+4.3%	+2.4%	-11.5%	-16.0%	-7.6%	-6.8%	-9.9%	-18.1%
Pooling-based	**56.0**	**49.3**	**76.5**	**1143.1**	**49.9**	**47.3**	36.9	**61.9**	30.7	7.3	**47.5**	**32.1**	**10.1**	**22.3**	**40.2**	**55.6**	36.2	**46.7**	11.2	40.1	46.6	**47.0**	**32.0**	**53.7**	46.2	**53.6**	49.3
Initialization	-15.0%	-15.3%	-8.8%	-13.3%	-16.9%	-22.9%	-6.7%	-7.3%	-9.9%	-9.6%	-3.2%	-5.1%	-15.7%	-21.6%	-18.4%	+22.5%	-1.5%	-2.7%	-13.0%	+1.6%	+0.3%	-10.1%	-4.9%	-4.8%	-10.2%	-9.7%	-18.3%

4.3.6 Vision Encoder's Layer Routing

Described in Section 3.4.3, the vision routing is a mechanism that combines the output of different layers of the vision encoder, in order to provide a richer visual information to the vision aggregator, based on the input question. This mechanism aims to bridge the gap of current state-of-the-art MLLMs that can not exploit both the low-level visual features of first layers and the more high-level visual concepts provided by last layers of the vision encoder.

In this section four different routing approaches are evaluated: 1) MLP with a softmax activation function to constrain the output weights to be between 0 and 1 and to sum up to 1; 2) MLP with a softmax activation function followed by a normalization of the output weights; 3) MLP with a sigmoid activation function followed by a normalization; 4) MLP with a sigmoid activation function followed by a normalization with an additional MLP to process visual information before weighted combination.

All these variants are based on a common architecture with LLaMA 3.2 1B as LLM, 3 blocks for the visual aggregator and 1 block for the text aggregator, with learning rates described in Table 4.36. Given that 8 learnable queries are used, in the following experiments the input of the routing MLP is the mean of them.

Results are reported in Table 4.41. None of the proposed variants emerge as the best one in all benchmarks, even if variants with softmax activation reach some better performance - beating the others and the variant without routing - thanks to its very good performance in tasks such as MathVista where it **outperforms** also the standard LLaVA architecture.

Variants with normalization step - as a possible way to mitigate the problem of combining features of different layers of the vision encoder - are evaluated, in particular the sigmoid-based which could benefit to operate on normalized values - due to the fact that the output weights are not constrained to sum up to 1, making it more difficult to learn to doze off the less relevant layers.

When normalization layer is applied, the softmax-based variant outperforms the sigmoid-based one, which can be explained by the fact that softmax provides a fixed budget of 1 which is distributed among the different layers, which can help to have a more stable training and simpler learning dynamics. These results could be reasonable given that normalization is a very simple operation that might not be effective to mitigate the problem of combining features of different layers of the vision encoder, so if the model is not able to learn to select only few layers - and as already mentioned the softmax is more effective than sigmoid in this task - the performance

can be significantly degraded by the presence of too much noisy information coming from less relevant layers.

Additionally, it is interesting to notice that the variant with an additional MLP to process visual information before weighted combination in the sigmoid-based variant is able to reach better performance than simpler one, demonstrating that when sigmoid-based combination of different layers is used, it is important to ensure that their intrinsically different layer distributions are processed in a more effective way before the weighted combination - in order to mitigate the problem of combining features of different layers of the vision encoder.

Table 4.41: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. 3 blocks for the vision aggregator and 1 block for the text aggregator, considering LLM's 16th layer. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
No Routing	54.6	48.5	74.1	1108.9	48.4	46.4	37.3	61.4	30.8	9.8	**47.2**	28.9	9.8	18.0	40.1	47.7	37.1	45.6	14.0	**41.2**	47.6	46.2	28.2	52.1	48.0	53.5	49.4
	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.6%	+21.2%	-3.8%	-14.5%	-18.5%	-36.9%	-18.7%	+5.1%	+1.0%	-5.1%	+8.7%	+4.3%	+2.4%	-11.5%	-16.0%	-7.6%	-6.8%	-9.9%	-18.1%
MLP Routing Softmax	54.7	49.0	73.4	1137.9	48.3	46.1	**38.4**	61.9	**30.9**	**14.6**	46.2	29.4	**10.1**	18.1	39.5	49.8	35.4	44.7	11.2	40.8	44.8	45.3	32.0	52.2	44.9	53.3	44.0
	-16.9%	-15.8%	-12.5%	-13.7%	-19.6%	-24.8%	-2.8%	-7.3%	-9.3%	+80.8%	-6.0%	-13.1%	-16.0%	-36.5%	-19.9%	+9.8%	-3.7%	-6.9%	-13.0%	+3.4%	-3.7%	-13.4%	-4.9%	-7.4%	-12.9%	-10.3%	-27.1%
MLP Routing Softmax + Norm.	56.0	**49.5**	**75.9**	1132.6	49.8	**47.9**	36.1	61.7	27.3	9.5	46.1	30.8	9.6	20.4	39.6	53.4	**37.2**	45.4	**14.6**	40.7	**48.1**	45.8	31.5	**52.7**	43.7	51.5	49.6
	-15.1%	-15.0%	-9.5%	-14.1%	-17.1%	-21.8%	-8.5%	-7.6%	-19.9%	+17.3%	-6.2%	-9.0%	-20.0%	-28.4%	-19.7%	+17.7%	+1.3%	-5.4%	+13.0%	+3.1%	+3.4%	-12.3%	-6.2%	-6.5%	-15.2%	-13.3%	-17.8%
MLP Routing Sigmoid + Norm.	**56.3**	49.4	74.9	**1181.4**	50.1	47.9	36.4	**62.0**	28.8	9.2	45.5	**30.9**	9.4	19.3	40.0	**55.1**	34.1	**45.8**	9.5	36.7	44.2	47.3	31.0	52.5	47.0	54.1	**51.8**
	-14.6%	-15.2%	-10.7%	-10.4%	-16.6%	-21.9%	-7.9%	-7.2%	-15.4%	+13.5%	-7.3%	-8.4%	-21.5%	-32.4%	-18.8%	+21.3%	-7.3%	-4.5%	-26.1%	-7.1%	-5.0%	-9.5%	-7.8%	-7.0%	-8.7%	-8.9%	-14.1%
Pre-MLP +MLP Routing Sigmoid + Norm.	55.8	48.7	75.0	1145.8	**50.5**	47.8	36.1	**62.0**	29.5	7.3	45.6	30.9	9.0	**20.7**	**40.7**	53.3	36.3	45.6	**14.6**	40.4	44.5	**47.4**	**32.7**	51.2	**49.3**	**54.6**	49.0
	-15.2%	-16.3%	-10.6%	-13.1%	-16.0%	-22.1%	-8.7%	-7.2%	-13.5%	-9.6%	-7.2%	-8.6%	-24.6%	-27.5%	-17.5%	+17.3%	-1.3%	-5.1%	+13.0%	+2.2%	-4.2%	-9.3%	-2.7%	-9.3%	-4.2%	-8.0%	-18.8%

Routing Behavior Analysis

The introduction of the routing mechanism permits the model to select the most relevant layers of the vision encoder based on the input question. Analyzing how the routing weights change during the training process can provide insights about the learning dynamics of the model and the importance of different layers of the vision encoder for different tasks.

Firstly, softmax-based routing variant is considered - which allows a clearer analysis of the layer selection thanks to its natural constraint to have output weights between 0 and 1 and to sum up to 1, considerable a sort of *"budget of relevance"*. In particular, mean and variance in linear scale of the routing weights are reported in Figure 4.13b - and in log-scale in Figure 4.13a to better visualize the distribution given that it is very unbalanced - for each layer of the vision encoder, at the end of the first stage and at the end of the second stage.

At the end of the first stage there is a clear selection for the penultimate layer of the vision encoder, with a mean which is approximately equal to 1 - unless for the floating point fluctuations - an a very low variance. This is very interesting because the model learns to select only the penultimate layer of the vision encoder - without any initial bias or explicit enticement in training objective - that is exactly the one which is commonly used in literature as input for MLLMs.

This empirically demonstrates that the penultimate layer of the vision encoder is the most informative one to solve high-level vision understanding tasks, such as image captioning, which is the task used in the first stage to train the vision part of the model.

On the other hand, at the end of the second stage the distribution of the routing weights changes visibly both in linear (Figure 4.14b) and log scale (Figure 4.14a), with a higher variance and a lower mean for the penultimate layer, which is still the most selected one, but with a significant contribution also from other layers of the vision encoder.

This can be explained by the fact that in the second stage the model is trained to solve more complex tasks - i.e., visual question answering - which require leveraging different types of visual information, both low-level features provided by more inner layers and high-level concepts provided by last layers of the vision encoder, so the model learns to select a combination of different layers to extract the most relevant information from the image based on the input question. In particular in log-scale (Figure 4.14a), there is a significant increase of the contribution of more than one layer - even if thanks to the softmax constraint of summing to 1 - the model is able to learn to select only few layers, which can be very effective to mitigate the problem of combining features of different layers that could lead to noisy information.

The same analysis was conducted for the sigmoid-based routing variant, which does not have the natural constraint of softmax to have output weights between 0 and 1 and to sum up to 1, making more difficult to learn to doze off the less relevant layers, but at the same time it can provide a more flexible way to combine features of different layers of the vision encoder.

At the end of the first stage (Figure 4.15b), even in this case there is a clear preference for the penultimate layer of the vision encoder, but differently from the softmax-based variant, the mean of the routing weights is not 1 - approximately to 0.9 - with a higher variance. Moreover, there is a notable contribution from some other layers that is not present in previous case - where thanks to the softmax nature the other layers are completely turned off.

This can be explained by the fact that sigmoid-based routing does not have any constraint to doze off the less relevant layers, letting the model keep a certain contribution from them, even

if they are less relevant, which can be a sort of noise for the learning process, leading to a worse performance in downstream tasks with respect to softmax-based routing.

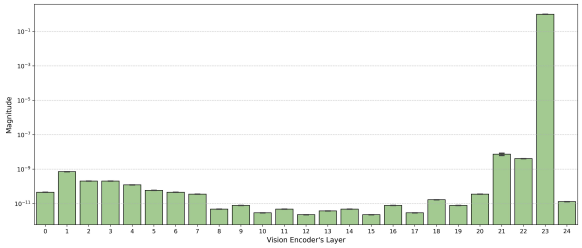
Instead, at the end of the second stage (Figure 4.16b) there is a different distribution of the routing weights, still preferring the penultimate layer, but with a contribution from other layers that are different from the previous case. Even in this case, the lack of sigmoid constraint may be the main reason for the higher contribution of other layers and - given that using sigmoid the model can use more layers - it could have found a different local minimum that leverages on more and different layers compared to softmax-based routing. The lower performance of sigmoid-based could be explained by the fact that the model combines visual features from more layers, which, as already mentioned, can be considered a sort of additional noise that can lead to a degradation of final performance in downstream tasks.

Textual Information Detachment in Routing

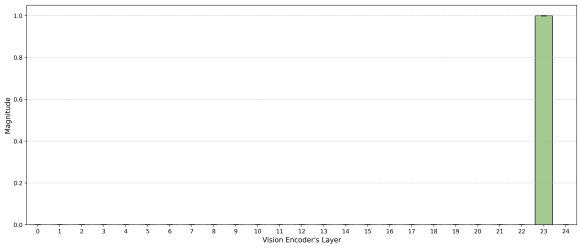
In Section 4.3.6, the proposed routing mechanism relies on the mean of output queries from the text aggregator. In this way, the gradient coming from the routing MLP can flow back to the text aggregator, which can either be beneficial thanks to the fact that the text aggregator can learn to produce queries that are more effective for the routing process, or it can be detrimental due to the fact that the routing MLP can eclipse the needs of the text aggregator, which can lead to a degradation of the performance in downstream tasks.

In order to understand which of these two cases is more likely to happen, an ablation study with a detached mean of the output queries from the text aggregator is proposed, which prevents the gradient coming from the routing MLP to flowing back to the text aggregator and modify its weights accordingly.

The considered model is the softmax-based without normalization described in Section 4.3.6 and results are reported in Table 4.42. It is clear that detaching the mean of the output queries from the text aggregator leads to a degradation of the performance - especially in visual question answering benchmarks - obtaining a tiny improvement in general tasks.

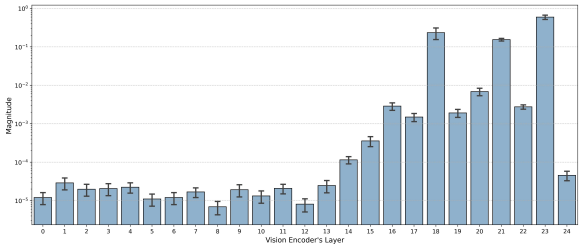


(a) Log-scale routing weights of softmax-based.

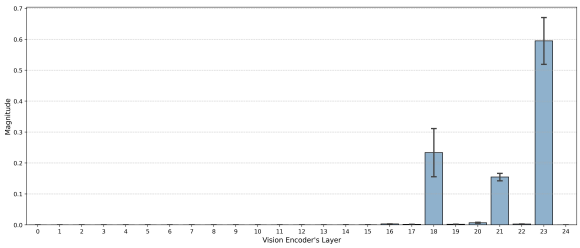


(b) Routing weights of softmax-based.

Figure 4.13: Routing weights of softmax-based at the end of the first stage.

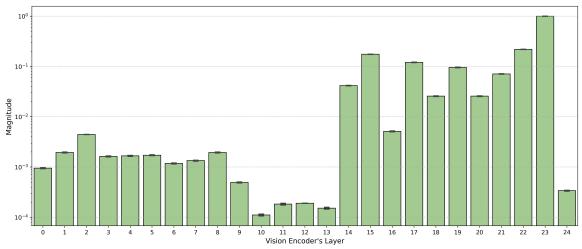


(a) Log-scale routing weights of softmax-based.

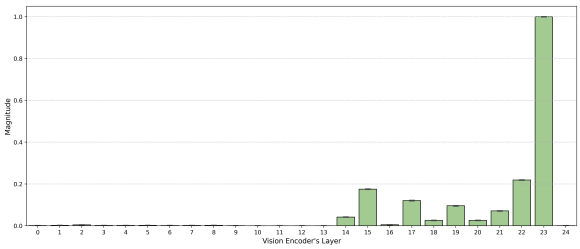


(b) Routing weights of softmax-based.

Figure 4.14: Routing weights of softmax-based at the end of the second stage.

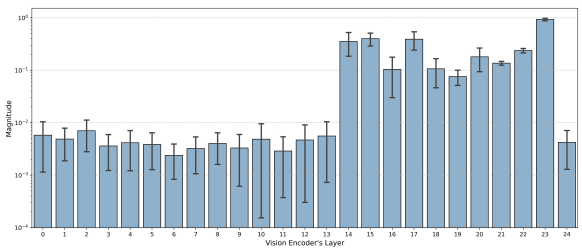


(a) Log-scale routing weights of sigmoid-based.

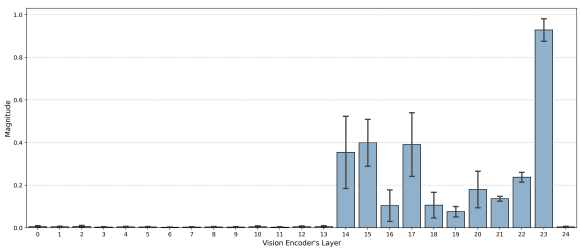


(b) Routing weights of sigmoid-based.

Figure 4.15: Routing weights of sigmoid-based at the end of the first stage.



(a) Log-scale routing weights of sigmoid-based.



(b) Routing weights of sigmoid-based.

Figure 4.16: Routing weights of sigmoid-based at the end of the second stage.

Table 4.42: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. 3 blocks for the vision aggregator and 1 block for the text aggregator, considering LLM's 16th layer. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Softmax MLP Routing	54.6	48.5	74.1	1108.9	48.4	46.4	**37.3**	**61.4**	**30.8**	9.8	**47.2**	28.9	**9.8**	**18.0**	**40.1**	47.7	**37.1**	**45.6**	**14.0**	**41.2**	**47.6**	**46.2**	28.2	52.1	**48.0**	**53.5**	**49.4**
	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.6%	+21.2%	-3.8%	-14.5%	-18.5%	-36.9%	-18.7%	+5.1%	+1.0%	-5.1%	+8.7%	+4.3%	+2.4%	-11.5%	-16.0%	-7.6%	-6.8%	-9.9%	-18.1%
Softmax MLP Routing Detached Text Aggr.	**55.1**	**49.1**	**74.5**	**1114.1**	**49.2**	**46.8**	36.8	61.3	29.5	**10.1**	46.1	**28.9**	9.5	17.3	39.0	**49.8**	34.5	44.2	10.7	39.0	44.1	45.4	**30.8**	**52.6**	41.9	52.5	49.3
	-16.4%	-15.6%	-11.2%	-15.5%	-18.2%	-23.6%	-7.0%	-8.2%	-13.5%	+25.0%	-6.1%	-14.5%	-20.6%	-39.2%	-20.9%	+9.7%	-6.0%	-7.9%	-17.4%	-1.2%	-5.1%	-13.0%	-8.2%	-6.7%	-18.6%	-11.6%	-18.3%

This empirically demonstrates that the gradient coming from the routing MLP does not disturb the learning process of the text aggregator, but, on the contrary, it is beneficial to have this gradient flow back to the text aggregator, which can learn to produce queries that are more effective for the routing process or to better understand which text features are most relevant, leading to a better performance in downstream tasks.

More Sophisticated Vision Encoder Layer Routing Approaches

In previous experiments the routing MLP takes as input the mean of the output queries from the text aggregator, which is a very simple way to combine the information from all queries, but it is not guaranteed that the mean of possibly very different queries is the best way to provide information to the routing part. For this reason, more sophisticated approaches are evaluated.

The first alternative relies on the introduction of an extra learnable query in the text aggregator which is only used as input for the routing MLP, while the other queries are used as input for the visual aggregator. In this way, the routing part can leverage a dedicated query to learn how to select the most relevant vision encoder layers, without any interference with the main objective of text aggregator's output.

The considered model is based on the softmax-based without normalization described in Section 4.3.6 and results are reported in Table 4.43. The variant without extra learnable query still achieves better results in most of visual question answering benchmarks, while the variant with extra learnable query performs better in other general tasks.

This could be explained by the fact that relying on 8 queries - instead of only one - there is a greater extraction capability thanks to the higher informative budget, which is exploited to

better understand the questions and decide which vision encoder layer must be selected, while in other benchmarks where the question is less important, the mean of the queries can be more effective to provide a more stable and less noisy signal to the routing MLP, leading to a better performance in downstream tasks.

An alternative approach to evaluate if averaging text aggregation output is the most effective way to produce a set of routing weights is to dynamically select vision encoder layers leveraging all output values at the same time. The proposed way is to treat output values independently, generating a set of routing weights for each of them and averaging them only at the end.

In this way each learnable query can learn how to aggregate its related text features and to produce its own routing weights based on its text-guided needs. Then, the final mean is used to mediate these different needs. In other words, in this approach the informative bottleneck for the routing is removed, preferring to have a more independently weights generation.

The only disadvantage of this approach is the increment in computational needs due to the computation of more routing weights. However, given the very small size of the routing MLP - in particular with respect to the size of the LLM - this additional computational need can be reasonably ignored.

Results are illustrated in Table 4.43 and show that this approach outperforms the previous one in almost all benchmarks - except for some visual question answering tasks - demonstrating that the pre-routing informative bottleneck degraded the model performance on downstream tasks. For this reason - ignoring the very little overhead of this method - this approach could be considered the best one among the proposed to select vision encoder layers.

Table 4.43: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. 3 blocks for the vision aggregator and 1 block for the text aggregator, considering LLM's 16th layer. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
Softmax MLP Routing	54.6	48.5	74.1	1108.9	48.4	46.4	37.3	61.4	**30.8**	9.8	**47.2**	28.9	9.8	**18.0**	40.1	47.7	**37.1**	**45.6**	14.0	**41.2**	**47.6**	46.2	28.2	52.1	48.0	53.5	**49.4**
	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.7%	+21.0%	-3.9%	-14.5%	-18.3%	-36.8%	-18.7%	+5.1%	+1.0%	-5.0%	+8.5%	+4.3%	+2.4%	-11.5%	-16.1%	-7.6%	-6.8%	-9.9%	-18.1%
Softmax MLP Routing With Extra Query	**55.1**	48.6	73.5	**1156.3**	48.6	**47.0**	**37.6**	**63.0**	29.0	**12.3**	46.0	28.4	**10.0**	14.1	39.5	50.0	34.8	44.5	14.0	36.4	44.4	46.1	31.2	52.2	45.1	53.1	48.8
	-16.4%	-16.5%	-12.4%	-12.3%	-19.1%	-23.3%	-4.9%	-5.7%	-15.0%	+51.9%	-6.3%	-16.0%	-16.7%	-50.5%	-19.9%	+10.1%	-5.2%	-7.3%	+8.5%	-7.8%	-4.5%	-11.8%	-7.1%	-7.4%	-12.4%	-10.6%	-19.1%
Softmax MLP Routing With Post Mean	55.0	**49.1**	**74.8**	1106.2	**48.9**	46.8	36.9	61.5	27.6	12.1	46.4	**30.0**	9.4	17.7	**40.2**	**52.7**	36.2	44.5	**15.1**	37.8	47.3	**47.6**	**31.6**	**52.7**	**48.4**	**56.0**	49.3
	-16.5%	-15.6%	-10.8%	-16.1%	-18.6%	-23.7%	-6.6%	-7.9%	-19.1%	+49.4%	-5.5%	-11.2%	-21.7%	-37.9%	-18.5%	+16.1%	-1.5%	-7.3%	+17.1%	-4.3%	+1.7%	-8.9%	-6.0%	-6.6%	-6.0%	-5.7%	-18.2%

4.3.7 2 Epochs in the First Stage

Similarly to the experiments conducted for the text-influenced vision-centric architecture, a variant with 2 epochs in the first stage is considered to understand if providing more time to the model to learn how to leverage the text-related components can lead to a better performance.

The evaluation is based on LLaMA 3.2 1B with 3 blocks for the visual aggregator and 1 block for the text aggregator, using the same learning rates used in previous experiments.

Results are reported in Table 4.44 and show that this approach is able to improve the performance of the model in several benchmarks, especially in general tasks.

However, in some visual question answering benchmarks the performance is degraded, which can be explained by the fact that the model is able to learn better how to leverage the text-related components, but it may have overfitted on the first stage datasets - unrecoverable in second stage - leading to a degradation of the performance in downstream tasks.

Even in this case, the second epoch in the first stage represents a valid training strategy to improve the performance of the model in several benchmarks, but requires additional studies to understand how to avoid performance degradation in some tasks.

Table 4.44: Comparison considering LLaVA LLaMA 3.2 1B with CLIP-ViT-Large-336. 3 blocks for the vision aggregator and 1 block for the text aggregator, considering LLM's 16th layer. Best values among models in each column are highlighted in **bold** and when also outperforming the baseline, they are further underlined. Relative percentage gaps are computed with respect to the baseline.

	General						Knowledge					OCR					Vision					Other					
Model	**Avg**	GQA	POPE	MME	MMB (EN)	SEED	**Avg**	SQA	MMMU	MathVista	AI2D	**Avg**	ChartQA	OCR_{Bench}	TextVQA	VizWiz	**Avg**	RWQA	MMVP	Blink	V*	**Avg**	MMStar	QBench	ADE	Omni3D	COCO
Standard LLaVA	65.9	58.2	83.9	1318.5	60.1	61.3	39.5	66.8	34.1	8.1	49.1	33.8	12.0	28.5	49.3	45.4	36.7	48.0	12.9	39.5	46.5	52.2	33.6	56.4	51.5	59.4	60.3
1 Epoch	54.6	48.5	74.1	1108.9	48.4	**46.4**	**37.3**	**61.4**	**30.8**	9.8	**47.2**	28.9	**9.8**	18.0	40.1	47.7	**37.1**	45.6	**14.0**	**41.2**	47.6	46.2	28.2	52.1	48.0	53.5	49.4
in Stage 1	-17.2%	-16.7%	-11.7%	-15.9%	-19.5%	-24.3%	-5.6%	-8.1%	-9.6%	+21.2%	-3.8%	-14.5%	-18.5%	-36.9%	-18.7%	+5.1%	+1.0%	-5.1%	+8.7%	+4.3%	+2.4%	-11.5%	-16.0%	-7.6%	-6.8%	-9.9%	-18.1%
2 Epochs	**55.2**	**48.7**	**75.1**	**1136.5**	**48.9**	**46.4**	36.4	61.1	28.6	**10.3**	45.4	**30.0**	9.6	**21.8**	**40.3**	**48.4**	36.2	**46.1**	13.5	37.4	**47.8**	**47.7**	**30.8**	**52.5**	**49.7**	**55.1**	**50.6**
in Stage 1	-16.2%	-16.3%	-10.5%	-13.8%	-18.6%	-24.3%	-8.0%	-8.5%	-16.0%	+26.9%	-7.6%	-11.1%	-19.7%	-23.4%	-18.2%	+6.6%	-1.4%	-3.9%	+4.3%	-5.3%	+2.7%	-8.5%	-8.4%	-7.0%	-3.4%	-7.2%	-16.1%

5. Conclusions

The rapid evolution of Multimodal Large Language Models has undeniably expanded the horizons of artificial intelligence, endowing generative models with the remarkable ability to perceive, interpret, and reason about the visual world. However, this multimodal integration has traditionally come at a steep computational cost, driven by the quadratic scaling of self-attention mechanisms when confronted with uncompressed, high-resolution visual grids.

This work embarked on a comprehensive exploration to address this critical inefficiency, challenging the prevailing assumption that generative models must ingest the entirety of extracted visual features to maintain perceptual accuracy. Through the rigorous design, implementation, and evaluation of dedicated aggregator modules, this research proposed a fundamental paradigm shift: moving away from brute-force token ingestion, toward sophisticated, semantics-driven visual token aggregation.

The central and most resounding conclusion drawn from this extensive experimental analysis is that MLLMs do not inherently require the full sequence of raw visual patches. Visual data, by its very nature, exhibits a high degree of spatial and semantic redundancy. The empirical findings systematically demonstrate that, for the vast majority of downstream tasks, it is entirely sufficient to provide the language model with a drastically reduced set of highly informative, dense visual concepts. By strategically distilling hundreds or thousands of patches into a compact bottleneck - sometimes retaining less than 2% of the original visual features - the proposed architectures successfully maintained robust semantic coherence, proving that a few but highly relevant tokens, rather than an exhaustive grid of redundant patches, are the key to unlocking computational efficiency without sacrificing broad multimodal reasoning capabilities.

This insight was robustly validated through a meticulous comparative analysis across a diverse spectrum of architectural configurations and aggregation paradigms. By evaluating text-agnostic vision-only aggregators, text-influenced pooling mechanisms, and complex text-guided architectures, this research provided a nuanced understanding of how visual information can be optimally compressed.

Furthermore, testing these aggregators across different LLMs - ranging from highly efficient models like Qwen 0.6B and LLaMA 1B to larger architectures like Vicuna 7B - illuminated the intricate interplay between visual compression and language modeling capacity. The results in-

dicated that while simpler pooling mechanisms offer remarkable training stability and efficiency, more sophisticated, text-guided aggregators excel in dynamically isolating task-critical features, especially when paired with LLMs capable of leveraging deep linguistic context to guide the visual synthesis.

While the empirical results strongly advocate for aggressive visual compression, this research also delineates the boundary conditions of the proposed methodologies. The evaluations consistently revealed that highly compressed visual representations encounter structural limitations in tasks demanding microscopic, fine-grained perception, such as Optical Character Recognition and dense document understanding. In these specific domains, the aggressive filtering of low-level visual artifacts inevitably leads to a loss of detail. Recognizing this trade-off is a crucial outcome of the architectural analysis: it confirms that while dense conceptual aggregation is highly optimal for general visual question answering, spatial reasoning, and conversational interactions, specialized tasks requiring pixel-perfect reading still dictate the physical limits of extreme token reduction.

In conclusion, this work contributes a foundational stepping stone to the ongoing quest for efficient and scalable multimodal artificial intelligence. By successfully demonstrating that intelligent visual distillation can match - and occasionally surpass - the performance of computationally heavy, uncompressed baselines, this work provides a clear architectural direction for future MLLM development. As the demand for multimodal intelligence continues to grow across various deployment environments, moving away from redundant token ingestion toward elegant, context-aware information aggregation will be paramount. Ultimately, the realization that Multimodal Large Language Models thrive on the quality and relevance of visual features, rather than their sheer quantity, paves the way for a new generation of sophisticated, highly efficient vision-language architectures.

6. Publications

The research activity carried out in the context of the realization of this work led to the realization of a paper currently under review for the ECCV Workshop 2026 “Assistive Computer Vision and Robotics”.

- Nicola Ricciardi, Davide Caffagni, Marcella Cornia and Lorenzo Baraldi. “*LLaVA-Lite: Lightweight Multimodal LLMs via Extreme Visual Token Compression*”. Submitted to the European Conference on Computer Vision (ECCV) Workshop “Assistive Computer Vision and Robotics”, 2026.

Bibliography

[1] An Yang et al. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388 [cs.CL]. URL: https://arxiv.org/abs/2505.09388.

[2] Andrea et al. Grattafiori. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[3] Haotian Liu et al. *Visual Instruction Tuning*. 2023. arXiv: 2304.08485 [cs.CV]. URL: https://arxiv.org/abs/2304.08485.

[4] Jean-Baptiste Alayrac et al. *Flamingo: a Visual Language Model for Few-Shot Learning*. 2022. arXiv: 2204.14198 [cs.CV]. URL: https://arxiv.org/abs/2204.14198.

[5] Junnan Li et al. *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*. 2022. arXiv: 2201.12086 [cs.CV]. URL: https://arxiv.org/abs/2201.12086.

[6] Junnan Li et al. *BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models*. 2023. arXiv: 2301.12597 [cs.CV]. URL: https://arxiv.org/abs/2301.12597.

[7] Fangming Cui et al. *Generalizable Prompt Learning of CLIP: A Brief Overview*. 2025. arXiv: 2503.01263 [cs.CV]. URL: https://arxiv.org/abs/2503.01263.

[8] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV]. URL: https://arxiv.org/abs/2104.14294.

[9] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[10] Francesco Locatello et al. *Object-Centric Learning with Slot Attention*. 2020. arXiv: 2006.15055 [cs.LG]. URL: https://arxiv.org/abs/2006.15055.

[11] Guangyuan Li et al. *Object-Centric Vision Token Pruning for Vision Language Models*. 2025. arXiv: 2511.20439 [cs.CV]. URL: https://arxiv.org/abs/2511.20439.

[12] Wei-Lin Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[13] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[14] Google Cloud. *What is Deep Learning?* URL: https://cloud.google.com/discover/what-is-deep-learning (visited on 03/08/2026).

[15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2020.

[16] James MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. 1967, pp. 281–297.

[17] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[18] Marius-Constantin Popescu et al. "Multilayer Perceptron and Neural Networks". In: (2009).

[19] Frank Rosenblatt. "The Perceptron: A Probabilistic Model for Information Processing in the Brain". In: *Psychological Review* 65.6 (1958).

[20] Warren S. McCulloch and Walter H. Pitts. "A Logical Calculus of the Ideas Immanent in Neural Activity". In: *Bulletin of Mathematical Biophysics* 52.1/2 (1943).

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL]. URL: https://arxiv.org/abs/1409.3215.

[22] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL]. URL: https://arxiv.org/abs/1406.1078.

[23] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[24] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[25] Yun Luo et al. *An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning*. 2025. arXiv: 2308.08747 [cs.CL]. URL: https://arxiv.org/abs/2308.08747.

[26] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[27] Brian Lester, Rami Al-Rfou, and Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. arXiv: 2104.08691 [cs.CL]. URL: https://arxiv.org/abs/2104.08691.

[28] Neil Houlsby et al. *Parameter-Efficient Transfer Learning for NLP*. 2019. arXiv: 1902.00751 [cs.LG]. URL: https://arxiv.org/abs/1902.00751.

[29] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[30] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[31] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[32] Bill Kromydas. *Convolutional Neural Network (CNN): A Complete Guide*. https://learnopencv.com/understanding-convolutional-neural-networks-cnn/. Accessed: 2026-05-06. 2023.

[33] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[34] Xiaohua Zhai et al. *Sigmoid Loss for Language Image Pre-Training*. 2023. arXiv: 2303.15343 [cs.CV]. URL: https://arxiv.org/abs/2303.15343.

[35] Nathan Lambert. *Reinforcement Learning from Human Feedback*. 2026. arXiv: 2504.12501 [cs.LG]. URL: https://arxiv.org/abs/2504.12501.

[36] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[37] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[38] Danyang Zhang et al. *Mixture of Experts in Large Language Models*. 2025. arXiv: 2507.11181 [cs.LG]. URL: https://arxiv.org/abs/2507.11181.

[39] Alexander Rubinstein et al. *Are We Done with Object-Centric Learning?* 2025. arXiv: 2504.07092 [cs.CV]. URL: https://arxiv.org/abs/2504.07092.

[40] Maximilian Seitzer et al. *Bridging the Gap to Real-World Object-Centric Learning*. 2023. arXiv: 2209.14860 [cs.CV]. URL: https://arxiv.org/abs/2209.14860.

[41] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312.

[42] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338. DOI: 10.1007/s11263-009-0275-4. URL: https://doi.org/10.1007/s11263-009-0275-4.

[43] Ke Fan et al. *Adaptive Slot Attention: Object Discovery with Dynamic Slot Number*. 2024. arXiv: 2406.09196 [cs.CV]. URL: https://arxiv.org/abs/2406.09196.

[44] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML]. URL: https://arxiv.org/abs/1611.01144.

[45] Justin Johnson et al. *CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning*. 2016. arXiv: 1612.06890 [cs.CV]. URL: https://arxiv.org/abs/1612.06890.

[46] Hongjia Liu et al. *MetaSlot: Break Through the Fixed Number of Slots in Object-Centric Learning*. 2025. arXiv: 2505.20772 [cs.CV]. URL: https://arxiv.org/abs/2505.20772.

[47] Gautam Singh, Fei Deng, and Sungjin Ahn. *Illiterate DALL-E Learns to Compose*. 2022. arXiv: 2110.11405 [cs.CV]. URL: https://arxiv.org/abs/2110.11405.

[48] Haotian Liu et al. *Improved Baselines with Visual Instruction Tuning*. 2024. arXiv: 2310.03744 [cs.CV]. URL: https://arxiv.org/abs/2310.03744.

[49] Haotian Liu et al. *LLaVA-NeXT: Stronger LLMs Supercharge Multimodal Capabilities*. LLaVA Blog. Accessed: 2024-05-22. 2024. URL: https://llava-vl.github.io/blog/2024-01-30-llava-next/.

[50] Chroma Research. *Context Rot*. Chroma Blog. Accessed: 2024-05-22. 2024. URL: https://www.trychroma.com/research/context-rot.

[51] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[52] Cheng-Ping Hsieh et al. *RULER: What's the Real Context Size of Your Long-Context Language Models?* 2024. arXiv: 2404.06654 [cs.CL]. URL: https://arxiv.org/abs/2404.06654.

[53] Maarten Grootendorst. *A Visual Guide to Gemma 4*. Substack newsletter. 2026. URL: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-gemma-4.

[54] Yuan Zhang et al. *SparseVLM: Visual Token Sparsification for Efficient Vision-Language Model Inference*. 2025. arXiv: 2410.04417 [cs.CV]. URL: https://arxiv.org/abs/2410.04417.

[55] Long Xing et al. *PyramidDrop: Accelerating Your Large Vision-Language Models via Pyramid Visual Redundancy Reduction*. 2025. arXiv: 2410.17247 [cs.CV]. URL: https://arxiv.org/abs/2410.17247.

[56] Senqiao Yang et al. *VisionZip: Longer is Better but Not Necessary in Vision Language Models*. 2026. arXiv: 2412.04467 [cs.CV]. URL: https://arxiv.org/abs/2412.04467.

[57] Mingqian Ji, Shanshan Zhang, and Jian Yang. *Revisiting Token Compression for Accelerating ViT-based Sparse Multi-View 3D Object Detectors*. 2026. arXiv: 2604.14563 [cs.CV]. URL: https://arxiv.org/abs/2604.14563.

[58] Jingyu Lei, Gaoang Wang, and Der-Horng Lee. *CORE: Compact Object-centric REpresentations as a New Paradigm for Token Merging in LVLMs*. 2025. arXiv: 2511.14072 [cs.CV]. URL: https://arxiv.org/abs/2511.14072.

[59] Donghwan Chi et al. *Slot-MLLM: Object-Centric Visual Tokenization for Multimodal LLM*. 2025. arXiv: 2505.17726 [cs.CV]. URL: https://arxiv.org/abs/2505.17726.

[60] Shaolei Zhang et al. *LLaVA-Mini: Efficient Image and Video Large Multimodal Models with One Vision Token*. 2025. arXiv: 2501.03895 [cs.CV]. URL: https://arxiv.org/abs/2501.03895.

[61] Edge Impulse. *Layers - Edge Impulse Documentation*. URL: https://docs.edgeimpulse.com/knowledge/concepts/machine-learning/neural-networks/layers (visited on 05/19/2026).

[62] Juho Lee et al. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. 2019. arXiv: 1810.00825 [cs.LG]. URL: https://arxiv.org/abs/1810.00825.

[63] Jiahui Yu et al. *CoCa: Contrastive Captioners are Image-Text Foundation Models*. 2022. arXiv: 2205.01917 [cs.CV]. URL: https://arxiv.org/abs/2205.01917.

[64] Ziyi Wu et al. *SlotFormer: Unsupervised Visual Dynamics Simulation with Object-Centric Models*. 2023. arXiv: 2210.05861 [cs.CV]. URL: https://arxiv.org/abs/2210.05861.

[65] Davide Caffagni et al. *Recurrence-Enhanced Vision-and-Language Transformers for Robust Multimodal Document Retrieval*. 2025. arXiv: 2503.01980 [cs.CV]. URL: https://arxiv.org/abs/2503.01980.

[66] Ethan Perez et al. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871 [cs.CV]. URL: https://arxiv.org/abs/1709.07871.

[67] Hanpei Fang et al. *Do Large Language Models Favor Recent Content? A Study on Recency Bias in LLM-Based Reranking*. 2025. arXiv: 2509.11353 [cs.IR]. URL: https://arxiv.org/abs/2509.11353.

[68] Haoran Chen et al. *Multimodal Language Models See Better When They Look Shallower*. 2025. arXiv: 2504.21447 [cs.CV]. URL: https://arxiv.org/abs/2504.21447.

[69] Shengbang Tong et al. *Cambrian-1: A Fully Open, Vision-Centric Exploration of Multimodal LLMs*. 2024. arXiv: 2406.16860 [cs.CV]. URL: https://arxiv.org/abs/2406.16860.

[70] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97.

[71] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[72] Danna Gurari et al. *VizWiz Grand Challenge: Answering Visual Questions from Blind People*. 2018. arXiv: 1802.08218 [cs.CV]. URL: https://arxiv.org/abs/1802.08218.

[73] Pan Lu et al. *MathVista: Evaluating Mathematical Reasoning of Foundation Models in Visual Contexts*. 2024. arXiv: 2310.02255 [cs.CV]. URL: https://arxiv.org/abs/2310.02255.

[74] Shengbang Tong et al. *Eyes Wide Shut? Exploring the Visual Shortcomings of Multimodal LLMs*. 2024. arXiv: 2401.06209 [cs.CV]. URL: https://arxiv.org/abs/2401.06209.

[75] Master IESC. *Artificial Intelligence, Machine Learning, and Deep Learning: Same context, Different concepts*. 2018. URL: https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/ (visited on 05/06/2026).

[76] Joy Dip Das et al. "Encoder-Decoder Based LSTM and GRU Architectures for Stocks and Cryptocurrency Prediction". In: *Journal of Risk and Financial Management* 17.5 (2024). URL: https://www.mdpi.com/1911-8074/17/5/200.

[77] Amanpreet Singh et al. *Towards VQA Models That Can Read*. 2019. arXiv: 1904.08920 [cs.CL]. URL: https://arxiv.org/abs/1904.08920.

[78] Xingyu Fu et al. *BLINK: Multimodal Large Language Models Can See but Not Perceive*. 2024. arXiv: 2404.12390 [cs.CV]. URL: https://arxiv.org/abs/2404.12390.