DEFORMABLE VISION QUERY CONNECTOR: LETTING LLMS QUERY VISION INFORMATION THROUGH POSITION-GROUNDED LOOKUP

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Vision-Language Models (LVLMs) represent images as long sequences of visual tokens. However, preserving fine-grained visual content under tight token budgets remains challenging: spatial aggregation often lacks adaptivity, while query-based resampling can collapse without extensive training and miss important regions. We propose the *Deformable Vision Query (DVQ) Connector*, a position-grounded and spatially adaptive interface embedded directly within LLM layers. DVQ associates each visual query with a reference point and uses deformable cross-attention to refine its spatial focus. Uniformly sampled reference points encourage image coverage, while learned refinements allow adaptive concentration on salient regions, overcoming limitations of cross-attention connectors through this position-grounding. DVQ also natively encodes positional information within its queries, enabling differentiable bounding box encodings. Experiments on small-scale per-task fine-tuning and general-purpose vision-language modeling demonstrate that DVQ is competitive with strong vision connectors like DeepStack, while DVQ intrinsically enables token compression, making it particularly effective under reduced vision token budgets and in high-resolution settings. Ablations further demonstrate DVQ's robustness and flexibility. We provide our source code at `https://anonymous.4open.science/r/dvq`.

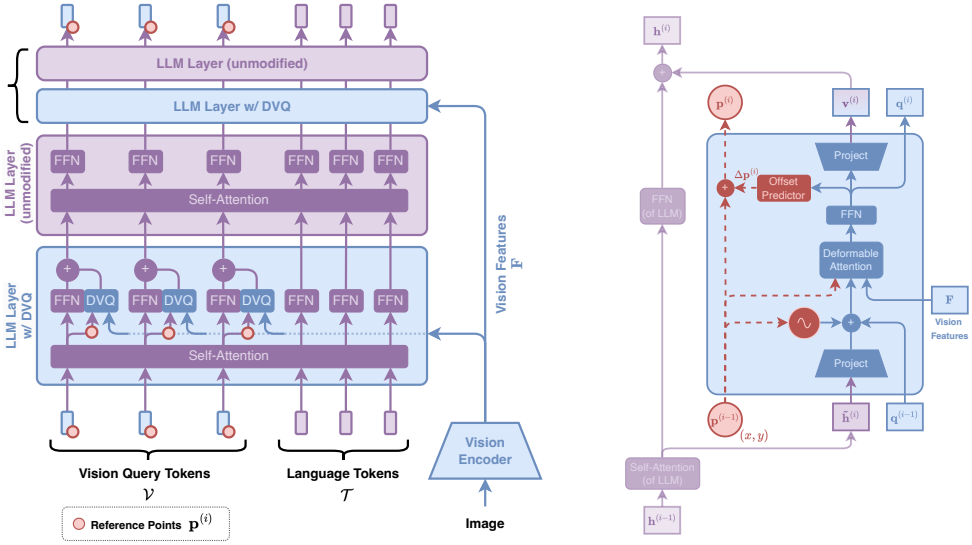
1 INTRODUCTION

Large Vision-Language Models (LVLMs) extend large language models (LLMs) with visual perception by projecting vision features into the LLM token space, achieving strong performance on visual question answering and reasoning (Liu et al., 2023). A major challenge is representing fine-grained visual information under a tight token budget, as naive dense patch projection yields many often redundant vision tokens. Existing vision connectors reduce this cost through spatial aggregation (Cha et al., 2024; Chen et al., 2024) or learned query-based resampling (Alayrac et al., 2022; Li et al., 2023b). However, spatial aggregation cannot adapt to image content, potentially wasting capacity on empty background, while query-based resamplers rely on unconstrained attention and provide no explicit guarantee of full-image coverage, potentially collapsing onto a dominant object.

We propose the *Deformable Vision Query (DVQ) Connector*, a position-grounded and spatially adaptive vision connector embedded directly within LLM layers. Rather than aggregating visual features externally, DVQ actively queries the vision encoder at each layer by using deformable cross-attention to sample around reference points, which are refined across layers to dynamically shift the spatial focus of each query token. Uniformly sampled initial reference points encourage full-image coverage, while refinement allows adaptive concentration on salient regions, mitigating the instability of unconstrained attention and the redundancy of uniform allocation. For instance, when a small object lies on a large background, DVQ guarantees that some queries are anchored near the object yet can still shift their focus adaptively, yielding robust representations without sacrificing global context. Furthermore, DVQ encodes positional information within its queries to guide visual sampling, enabling the model to process and predict spatial inputs or outputs like bounding boxes in a differentiable manner, without relying on discrete tokens.

Our contributions are as follows:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Injection of DVQ into an LLM.

(b) DVQ Component.

Figure 1: Overview of our DVQ connector. DVQ is inserted into selected LLM layers, operating in parallel with the LLM's FFN (a). At each layer i, it maintains a fixed-size set of $|\mathcal{V}|$ vision queries $\mathbf{q}^{(i)}$ (blue), each explicitly grounded via reference points $\mathbf{p}^{(i)}$ (red). Within DVQ (b), deformable attention gathers visual evidence around $\mathbf{p}^{(i)}$ from multi-level feature maps $\mathbf{F}$ to compute the injection signal $\mathbf{v}^{(i)}$. Uniform initialization of $\mathbf{p}^{(0)}$ ensures spatial coverage, while predicted offsets $\Delta\mathbf{p}^{(i)}$ refine the reference points for the next layer, allowing adaption of the spatial focus.

- We propose the *Deformable Vision Query (DVQ) Connector*, a position-grounded and spatially adaptive connector module to integrate visual information into LLMs.
- We show that DVQ is competitive with strong vision connectors such as DeepStack in both small-scale fine-tuning and general-purpose vision-language modeling scenarios.
- We demonstrate that DVQ intrinsically enables token compression, making it particularly effective under reduced vision token budgets and in high-resolution settings. Furthermore, DVQ natively integrates differentiable bounding-box inputs and outputs, a capability well-suited for end-to-end trainable systems.
- We demonstrate that DVQ's explicit position-grounding overcomes the limitations of dot-product cross-attention connectors. Additionally, we provide thorough ablations and analyses covering different token budgets, different LLMs and LLM scales, layer placement and parameter sharing, as well as different vision feature maps. Finally, qualitative analyses provide further insights into DVQ's behavior.

2 Preliminary: Vision Connectors for LLMs

LVLMs extend pretrained LLMs to jointly process text and images. An image is encoded into visual features $\mathbf{F}$, and a *vision connector* translates these features into signals for the LLM, either by constructing visual tokens appended to the input or by injecting signals into hidden states at selected layers. Let $\mathcal{V}$ denote the set of vision token positions and $\mathcal{T}$ the text token positions. We define a vision-derived signal $\mathbf{v}^{(i)}(\mathbf{F}) \in \mathbb{R}^{|\mathcal{V}|\times D}$, where D is the LLM hidden dimension. For token-based connectors, $\mathbf{v}^{(0)}$ provides the vision token embeddings and is concatenated with the text embeddings. For LLM layer injection, at layer $i > 0$, we update the LLM's hidden states $\mathbf{h}^{(i)}$:

$$\mathbf{h}_s^{(i)} \leftarrow \mathbf{h}_s^{(i)} + \mathbf{v}_s^{(i)}(\mathbf{F}) \qquad \forall s \in \mathcal{V}. \tag{1}$$

Patch Projection: Representing Vision as Tokens Patch projection (Liu et al., 2023) is the simplest and most widely used connector. The final vision feature map is spatially aggregated if needed (e.g., pooled), flattened, and projected via a learnable projection into a vision token sequence $\mathbf{v}^{(0)} \in \mathbb{R}^{|\mathcal{V}| \times D}$. This results in $|\mathcal{V}| = H \times W$ vision tokens for a (spatially aggregated) feature map of size $H \times W$. The resulting tokens are concatenated with the text embeddings and processed jointly by the LLM.

Perceiver: Learning What to Extract Patch projection ties the number of vision tokens to the spatial resolution of the feature map. To compress the visual information into a small, fixed set of tokens, a *Perceiver* can be used. It learns a set of queries $\mathbf{Q} \in \mathbb{R}^{|\mathcal{V}| \times D}$ (where $|\mathcal{V}|$ is the desired, freely chosen output size) that attend to the projected visual features (used as key/value sequences):

$$\mathbf{v}^{(0)} = \mathrm{CrossAttention}\left(\mathbf{Q}, \mathrm{Project}\left(\mathbf{F}\right)\right) \tag{2}$$

DeepStack: Injecting Vision Across Layers Token-based methods introduce visual information only at the input layer, limiting deeper layers' access to fine-grained features. DeepStack (Meng et al., 2024) injects multi-level vision feature maps $\mathbf{F} = \{\mathbf{F}^{(l)}\}_{l=1}^{L}$ across selected LLM layers, with L denoting the number of available levels. A mapping $l(i)$ assigns a subset of LLM layers i to vision levels $l(i)$, which are then projected to get $\mathbf{v}^{(i)}$, the vision information injected at this LLM layer. Initial vision tokens $\mathbf{v}^{(0)}$ are obtained via patch projection from the final feature map $\mathbf{F}^{(L)}$.

Cross-Attention: Hidden State Dependent Vision Querying Cross-attention connectors enable tokens to to directly attend to visual features at each layer. This allows the model to retrieve visual information dynamically, conditioned on the current hidden state $\mathbf{h}^{(i)}$:

$$\mathbf{v}^{(i)} = \mathrm{CrossAttention}\left(\mathbf{h}^{(i)}, \mathrm{Project}\left(\mathbf{F}\right)\right) \tag{3}$$

3 Deformable Vision Query (DVQ) Connector

We propose the *Deformable Vision Query (DVQ) Connector*, a position-grounded lookup mechanism for integrating visual context into LLMs (Fig. 1). At each LLM layer i, DVQ maintains a fixed-size set of queries $\mathbf{q}^{(i)}$. However, unlike a Perceiver, our queries are explicitly grounded via associated reference points $\mathbf{p}^{(i)}$ and refined throughout the LLM layers (Sec. 3.1). The resulting vision signal $\mathbf{v}^{(i)}$ is injected into the LLM's hidden state. Unlike DeepStack, $\mathbf{v}^{(i)}$ is computed via explicit lookup, using deformable attention over multi-level feature maps $\mathbf{F} = \{\mathbf{F}^{(l)}\}_{l=1}^{L}$ to gather visual evidence around each $\mathbf{p}^{(i)}$ (Sec. 3.2). To ensure coverage across the entire image space and prevent spatial query collapse, we uniformly sample the initial reference points $\mathbf{p}^{(0)}$ (Sec. 3.3). Implementation details are provided in Appendix C.1.

3.1 Query Propagation and Injection

DVQ is inserted at a chosen subset of LLM layers, typically every second (Fig. 1a). At each such layer i, the DVQ module operates in parallel with the LLM's FFN, with both modules receiving the hidden state after self-attention, $\tilde{\mathbf{h}}^{(i)}$. The vision-derived signal $\mathbf{v}^{(i)}$, used to update the LLM's hidden states $\mathbf{h}^{(i)}$ following Eq. (1), is computed as:

$$\mathbf{v}^{(i)} = \mathrm{DVQ}\left(\tilde{\mathbf{h}}^{(i)}, \mathbf{q}^{(i-1)}, \mathbf{p}^{(i-1)}, \mathrm{Project}\left(\mathbf{F}\right)\right), \tag{4}$$

with queries $\mathbf{q}^{(i-1)} \in \mathbb{R}^{|\mathcal{V}| \times D_{\mathrm{DVQ}}}$ from the previous layer and corresponding reference points $\mathbf{p}^{(i-1)} \in [0,1]^{|\mathcal{V}| \times 2}$, where the number of vision queries $|\mathcal{V}|$ can be freely chosen and D_{DVQ} refers to the hidden dimension of DVQ. Internally (Fig. 1b), to keep the computational overhead small, $\tilde{\mathbf{h}}^{(i)}$ is first projected down to the DVQ dimension $D_{\mathrm{DVQ}} < D$ and combined with the previous query features $\mathbf{q}^{(i-1)}$ via a residual connection from the previous DVQ layer. Sinusoidal positional encodings of the previous reference points $\mathbf{p}^{(i-1)}$ are then added to provide explicit spatial context. These features are then fed into the deformable attention module (Sec. 3.2) together with $\mathbf{p}^{(i-1)}$ and the multi-level feature maps $\mathbf{F} = \{\mathbf{F}^{(l)}\}_{l=1}^{L}$. The attention output is then passed through a feed-forward

network, which produces the refined queries $\mathbf{q}^{(i)}$. A dedicated offset predictor takes these refined queries as input and outputs an offset update $\Delta\mathbf{p}^{(i)}$ in logit space (we only apply a sigmoid when sampling, see Appendix C.1). This offset refines the reference points as $\mathbf{p}^{(i)} = \mathbf{p}^{(i-1)} + \Delta\mathbf{p}^{(i)}$, allowing queries to adapt their focus across layers. The refined queries $\mathbf{q}^{(i)}$ are projected back up to the LLM's full dimension via a linear layer to form $\mathbf{v}^{(i)}$. Zero-initialization of this layer ensures no visual signal is injected at the start of training. Note that we use pre-norm and residual connections at all modules, but omit them from the figure and equations for clarity.

3.2 Deformable Attention

We follow the deformable attention mechanism of Zhu et al. (2021). Unlike dot-product attention, deformable attention does not rely on feature similarity. Instead it explicitly samples a small set of locations around each query's reference point. This sparse, position-driven mechanism is computationally efficient and naturally suited to our spatially grounded queries. For each query token at index $s \in \mathcal{V}$ with reference point $\mathbf{p}_s \in [0,1]^2$, the module predicts M sampling offsets $\delta^{(l)}_{smh} \in [0,1]^2$ and attention weights $\alpha^{(l)}_{smh} \in [0,1]$ (normalized such that $\sum_l \sum_m \alpha^{(l)}_{smh} = 1$) across all heads h and feature levels l. The per-head output is computed by sampling at the predicted offsets:

$$\mathbf{z}_{s,h} = \sum_{l} \sum_{m=1}^{M} \alpha^{(l)}_{smh} \cdot \mathbf{W}\mathbf{F}^{(l)}\left[\mathbf{p}_s + \delta^{(l)}_{smh}\right], \tag{5}$$

where $\mathbf{W}$ is a learned value projection and $\mathbf{F}^{(l)}[\cdot]$ denotes bilinear interpolation of the level l feature map at continuous coordinates. The outputs of all heads are concatenated and projected via an output projection, as in standard multi-head attention.

3.3 Query and Reference Point Initialization

We initialize the query tokens $\mathbf{q}^{(0)} \in \mathbb{R}^{|\mathcal{V}| \times D_{\mathrm{DVQ}}}$ as learned, trainable embeddings, analogous to the query tokens in a Perceiver. The corresponding initial reference points $\mathbf{p}^{(0)}$ are non-learnable parameters, sampled uniformly from $[0,1]^2$ once at initialization. This assures good spatial coverage, even after only a few tokens, allowing DVQ to rapidly adapt the queries to the specific input image, despite the causal nature of the LLM. We then compute $\mathbf{v}^{(0)}$ by linearly projecting these queries into the D-dimensional LLM space[1]. Note that our connector also allows for other reference point initializations such as providing them as inputs (see for example Sec. 4.3).

4 Experiments

We compare our DVQ connector against existing vision connectors in two settings: small-scale per-task finetuning (Sec. 4.1) and general-purpose vision-language modeling (Sec. 4.2). For comparison, we consider the common types of vision connectors (see Sec. 2), treating *Patch Project* (Liu et al., 2023) and *DeepStack* (Meng et al., 2024) as our primary baselines, as they are commonly used in modern VLMs, for example in Qwen models (Bai et al., 2025). In Sec. 4.3, we additionally study how DVQ can utilize bounding boxes as differentiable inputs or predictions. Furthermore, we provide extensive ablation studies and model analyses in Appendix A. Details on our experimental setup and model implementation are provided in Appendix C.

4.1 Small-scale Fine-Tuning on Vision-Language Tasks

We study how DVQ performs in a small-scale fine-tuning scenario, where we first pretrain on COCO captions Chen et al. (2015) before fine-tuning the model individually on three tasks: The general VQA task GQA (Hudson & Manning, 2019), the OCR-heavy task TextVQA (Singh et al., 2019), and the localization task RefCOCOg (Mao et al., 2016). We compare DVQ with common types of vision connectors, including *Patch Project*, *DeepStack*, and *Perceiver*-based compression (an example of query-based resampling). For Patch Project and DeepStack, we reduce token counts (if required) via *spatial aggregation*. For DeepStack we also consider a version with more parameters

[1] In practice we can directly learn D-dimensional embeddings and downproject them in the first DVQ layer.

Table 1: Comparison of vision connectors on small-scale fine-tuning experiments with $|\mathcal{V}| = 1024$ and $|\mathcal{V}| = 256$ vision tokens. DVQ performs best or second best on all tasks, often closely matching or surpassing DeepStack and clearly outperforming Patch Project. Ablating position-grounding (cross-attention connector) or moving cross-attention outside the LLM (Perceiver) causes severe drops, showing that DVQ's explicit position-grounding overcomes the limitations of cross-attention.

Vision Connector	$\|\mathcal{V}\| = 1024$			$\|\mathcal{V}\| = 256$		
	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]
Patch Project	62.6	46.4	24.4	56.2	37.1	18.2
DeepStack (default injections)	63.0	**60.7**	55.3	62.2	**49.8**	54.2
→ 18 injections (as in DVQ)	62.4	47.3	67.3	62.2	47.5	**66.6**
Perceiver	56.6	13.1	11.0	48.6	10.2	9.1
DVQ (ours)	**64.6**	53.5	**67.9**	**64.2**	**49.8**	63.3
→ no grounding (cross-attention)	59.4	29.3	26.2	56.8	24.9,	28.7

and 18 injection layers, mimicking DVQ's injections. Additionally, we also train a *cross-attention* connector which closely follows DVQ's implementation but without explicit position grounding, i.e. with the deformable attention replaced by dot product attention. These two ablation studies verify that any measured benefits are not caused solely by additional parameters or injection points and validate the relevance of position-grounding.

DVQ shows strong performance in the fine-tuning scenario. As shown in Tab. 1, DVQ performs best or second best on all tasks, closely matching or outperforming DeepStack on most tasks, while clearly surpassing all other connectors including standard Patch Project. On GQA, DVQ is consistently best, exceeding DeepStack by 2.5% (at $|\mathcal{V}| = 1024$) and 3.2% (at $|\mathcal{V}| = 256$), respectively. On TextVQA, DVQ shows some limitations: at 1024 tokens it underperforms DeepStack by 11.9% but closes the gap at 256 tokens. On RefCOCOg, DVQ also performs well, outperforming DeepStack's default version by 22.8% (at $|\mathcal{V}| = 1024$) and 16.8% (at $|\mathcal{V}| = 256$), respectively. However, when DeepStack uses more injection points, the same 18 LLM layers as DVQ, it closes the gap on RefCOCOg and even slightly outperforms DVQ at $|\mathcal{V}| = 256$ tokens We assume this is because the default DeepStack underutilizes injection points in later LLM layers, which are important for localization. However, DeepStack's scores on other tasks do not improve with this variant, so DVQ's advantage is not simply due to more injection points or parameters (see also Appendix A.3).

Position-grounding overcomes the limitations of cross-attention. We observe that DVQ degrades severely when position grounding via deformable attention is removed (i.e., using the cross-attention connector), despite similar parameter counts, identical injection points, and higher compute from quadratic complexity. Moving cross-attention outside the LLM into a Perceiver performs even worse. This may explain why prior work favors patch projection or DeepStack, while cross-attention methods such as Q-Former (Li et al., 2023b), Flamingo (Alayrac et al., 2022), or Perceiver (Xiao et al., 2024) rely on extensive (pre)training for stability. DVQ's position-grounding avoids this instability, achieving competitive performance with no more training data or stages than DeepStack.

DVQ degrades gracefully under token budget reduction. When reducing the number of vision tokens (Fig. 2), DVQ degrades more slowly than other connectors on most tasks. The largest drops are observed with the Patch Project and DeepStack connectors, especially on TextVQA. Going from a token budget of 1024 to 256 and 64 the performance of DeepStack (with default injections) on TextVQA drops by 18.0% and 48.8%, respectively, while for our DVQ connector the performance only drops by 6.9% and 28.8%. Cross-attention is also affected by drops on most tasks while the Perceiver scores very low on all budgets.

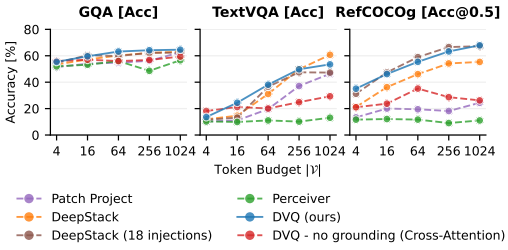


Figure 2: Performance of different vision connectors with varying token budgets.

DVQ utilizes high-resolution imaging more effectively than DeepStack, closing the gap on OCR performance. We also evaluate how well different vision connectors can utilize higher resolution images. In this scenario, we keep the same vision encoder but process the image in two parallel ways: once as a 512×512 low-resolution variant and at 1024×1024 resolution. The high-resolution variant is split into four 512×512 patches, encoded independently, and re-merged. For DeepStack, we use the low-resolution feature map for embeddings and the high-resolution map for injection, as is its standard configuration. For DVQ, we sample the positions from both the low- and high-resolution feature maps. As shown in Tab. 2, our DVQ connector utilizes the high-resolution images more effectively, leading to improvements in 2 of the 3 tasks. Most notably, the TextVQA score improves by 12.9%, closing the gap to DeepStack almost completely. Thus, DVQ provides competitive OCR performance when high resolution feature maps are provided.

Vision Connector	Vision Resolution	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]
DeepStack	512^2	63.0	**60.7**	55.3
DVQ (ours)	512^2	**64.6**	53.5	**67.9**
DeepStack	1024^2	63.4	59.4	49.2
DVQ (ours)	1024^2	**65.2**	**60.4**	**66.3**

Table 2: Performance in the high resolution scenario. DVQ's performance benefits more from high-resolution image encoding than DeepStack, outperforming it on all tasks in this scenario and closing the gap on OCR.

4.2 General Purpose Vision-Language Modeling

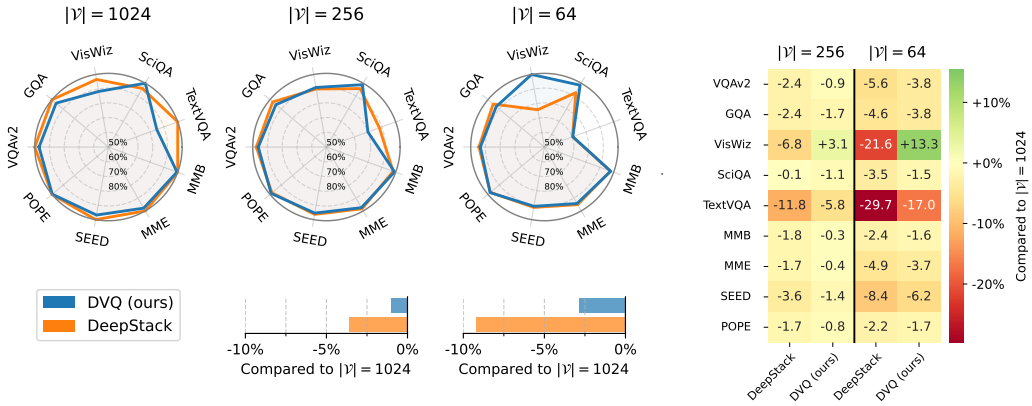
We study our vision connector in a general purpose vision-language modeling scenario following the LLaVA 1.5 (Liu et al., 2024) training protocol. We evaluate the resulting model on common VLM benchmarks, namely VQAv2 (Goyal et al., 2017), GQA (Hudson & Manning, 2019), VisWiz (Gurari et al., 2018), the image-context questions from ScienceQA (Lu et al., 2022), TextVQA (Singh et al., 2019) (without providing OCR tokens), the english part of MMBench (MMB) (Liu et al., 2025b), MME (Fu et al., 2025), SEED (Li et al., 2023a), and POPE (adversarial) (Li et al., 2023c). We compare our DVQ connector with DeepStack, the most competitive connector in Sec. 4.1 and a frequently used approach in modern VLMs (Bai et al., 2025). For fair comparison, we train DeepStack with the same protocol and base models. To study the performance under different vision token budgets, we train models for the full token budget $|\mathcal{V}| = 1024$ as well as reduced budgets of 256 and 64 tokens. Results are shown in Fig. 3, with additional baselines including Patch Project, Perceiver, and other published baselines provided in Appendix A.7.

DVQ demonstrates good visual concept alignment. DVQ performs exceptionally well on the SciQA task (4.3% improvement over DeepStack at 1024 tokens), while also showing strong performance on MMB and POPE (almost on par with DeepStack, reaching more than 99% of its score). This indicates that DVQ aligns high-level visual concepts well and enables complex visual reasoning. A limitation of DVQ is observed on OCR-heavy tasks, where it reaches only 84.9% and 91.4% of DeepStack's TextVQA and VisWiz scores at 1024 tokens. This likely stems from text requiring pixel-precise sampling, which sparse, point-based queries cannot always fully provide. Note that we observed similar behavior in the small-scale setting (Sec. 4.1), but found that increasing the token budget (Appendix A.2) or using high-resolution inputs (Tab. 2) closes most of this gap. On other tasks (VQAv2, GQA, MME, SEED) DVQ performs slightly, but not notably, worse than DeepStack, reaching around $96 - 98\%$ of DeepStacks performance on 1024 tokens. Overall DVQ closely matches DeepStack's performance on many tasks but rarely exceeds it.

DVQ handles a reduced token budget better than DeepStack. When reducing the vision token budget from 1024 to 256 or 64 tokens, DVQ retains its performance much better than DeepStack. While DeepStack degrades on average by 3.6% (at 256 tokens) and 9.2% (at 64 tokens) compared to the full budget, DVQ only degrades by 1.0% and 2.9%. At 256 and 64 tokens, DVQ now also outperforms DeepStack on VisWiz, MMB, and POPE, while narrowing or closing the gap on most other tasks, including the OCR-heavy task TextVQA.

4.3 Bounding Box Inputs and Outputs

DVQ integrates spatial location directly into the connector via reference points. This enables differentiable bounding-box inputs and outputs, which we evaluate on RefCOCOg in two scenarios: referring expression comprehension (box prediction) and region captioning (box input). In the de-



(a) Performance under three different vision token budgets $|\mathcal{V}|$: 1024, 256, and 64. **Top:** Performance per task (consistently normalized per task by best score over all models and budgets). **Bottom:** Mean performance drop compared to each model's $|\mathcal{V}| = 1024$ performance.

(b) Performance drops per task on reduced vision token budgets, compared to each model's full budget performance at $|\mathcal{V}| = 1024$.

Figure 3: General purpose vision-language modeling results following the LLaVA 1.5 training protocol and datasets, with detailed results and baselines provided in Appendix A.7. We compare our DVQ connector with DeepStack, a competitive and frequently used approach, which we trained in the same setup and with the same base models. DVQ closely matches the baseline performance on most tasks, and closes the remaining gaps when reducing the token budgets. Overall, DVQ is competitive at general-purpose language modeling while being less sensitive to smaller token budgets.

fault setting, bounding box outputs are encoded as text and box inputs are inpainted into images, as is common in VLMs. DVQ, however, associates every vision token with a reference point at each layer, providing a continuous and differentiable spatial representation. We investigate whether these reference points can serve as location encodings for both inputs and outputs. For bounding box prediction, we introduce additional vision tokens, called *Mention Region-of-Interest (MenROI)* tokens, into the output sequence. To this end, we extend the reference points into reference boxes, which additionally encode box size, and use the refined reference boxes from the last DVQ layer as predictions. For bounding box inputs, we similarly introduce additional vision tokens whose initial reference points/boxes encode either a single reference bounding box or a set of 5, 25, or 100 reference points describing the target region. To further improve localization, we also investigate pretraining DVQ with COCO bounding box annotations before fine-tuning. Implementation details are provided in Appendix C.2. Results are shown in Tab. 3.

DVQ is less reliant on bounding box pretraining. As shown in Tab. 3a, under the default setting with textual bounding box outputs, DVQ relies substantially less on bounding box pretraining than competing connectors. Without pretraining, DVQ's performance drops by only 13.4%, whereas DeepStack drops by 29.7%, more than twice as much. Consequently, DeepStack performs substantially worse than DVQ without pretraining but narrows the gap when pretraining is used.

DVQ's MenROI vision tokens enable differentiable bounding box outputs. Tab. 3a further shows that MenROI tokens provide a fully differentiable mechanism for bounding box prediction, reaching 93.9% of DVQ's text-token-based performance and matching the performance of Patch Project. However, they require bounding box pretraining to perform well. They therefore represent an attractive alternative when differentiable outputs are required, e.g. when predictions are used by downstream models, while text-based outputs remain the better default for DVQ.

DVQ's reference boxes enable differentiable bounding box inputs. In Tab. 3b, we show that DVQ vision tokens can also be used to encode differentiable bounding box inputs, performing roughly on par with inpainted boxes (reaching the same METEOR score and 98.7% of the CIDEr score). Providing the bounding box through multiple reference points instead performs slightly worse, especially with only a few points. Overall, DVQ achieves performance slightly below DeepStack (98.5% in METEOR, $97.\%$ in CIDEr), and outperforms Patch Project, showing that DVQ can handle both inpainted and differentiable bounding box inputs.

Table 3: Vision-language modeling with bounding box outputs and inputs. We study different mechanisms for bounding box prediction (a) and input (b) with the DVQ connector, comparing against Patch Project and DeepStack under the small-scale pipeline (Sec. 4.1). DVQ's design integrates spatial positions directly into the connector, making it specifically well-suited for representing location information. Our experiments show that this design supports differentiable bounding box prediction and input, and that DVQ remains competitive with the baselines when using standard box encodings.

(a) Bounding box prediction. DVQ enables the prediction of differentiable bounding boxes via MenROI tokens, which reaches 93.9% of the text-based performance but requires box pretraining to succeed. With text-based box prediction, however, DVQ is less reliant on box pretraining than other connectors.

Vision Connector	Box Output	RefCOCOg [Acc@0.5]
Without bounding box pretraining (see Sec. 4.1)		
Patch Project	Text	24.4
DeepStack	Text	**55.3**
DVQ (ours)	Text	**67.9**
DVQ (ours)	MenROI	43.6
With bounding box pretraining		
Patch Project	Text	73.6
DeepStack	Text	**78.7**
DVQ (ours)	Text	**78.4**
DVQ (ours)	MenROI	73.6

(b) Bounding box inputs. DVQ enables differentiable bounding box inputs via reference boxes or points, which closely matches DVQ's performance with inpainted bounding box inputs.

Vision Connector	Box Input	RefCOCOg	
		[METEOR]	[CIDEr]
Patch Project	Inpainted	19.3	96.9
DeepStack	Inpainted	**20.5**	**104.8**
DVQ (ours)	Inpainted	**20.2**	**102.6**
DVQ (ours)	Ref. Box	**20.2**	101.3
DVQ (ours)	Ref. Points (5)	19.7	97.4
DVQ (ours)	Ref. Points (25)	19.9	100.3
DVQ (ours)	Ref. Points (100)	20.1	100.0

4.4 Ablation Studies and Model Analysis

We further study the model behavior across different scenarios. Appendix A.1 shows that DVQ performs well across different LLMs, including a unified vision-language foundation LLM, and handles small LLMs better than DeepStack. In Appendix A.2, we also show that DVQ can utilize extended token budgets, closing the performance gap to DeepStack on OCR. In Appendix A.3, we study DVQ layer placement within the LLM as well as DVQ parameter sharing, showing that the number of injection points can be reduced and DVQ parameters shared without a large impact on performance. Additionally, Appendix A.4 studies the relevance of different vision feature maps, while Appendix A.5 compares the computational costs of DVQ and DeepStack.

Qualitative analysis demonstrates that vision queries steer towards relevant regions. In Appendix A.6, we analyze how reference points of vision queries are distributed across image space and how they evolve over the DVQ layers. For simple scenes, the reference points quickly focus on the primary foreground objects. With increasing complexity, the reference points start to explore the scene further in later layers, while in complex scenes the focus remains more distributed across the whole scene. Overall, this demonstrates that the vision queries can adaptively focus on relevant image regions while also keeping a more widespread coverage in complex scenes.

5 Discussion

Relevance and Impact of DVQ We demonstrate that DVQ is a strong vision connector that outperforms or closely matches the commonly used DeepStack- and Patch Project-based approaches. Notably, we also show that DVQ's explicit position-grounding overcomes the limitations of cross-attention-based connectors. DVQ performs especially well in the per-task fine-tuning scenario, on high-resolution images, with reduced vision token budgets, and with small LLMs. While also competitive in general-purpose vision-language modeling, this makes DVQ especially well-suited as a vision connector for specialized small-scale scenarios.

Future Work The token efficiency and high-resolution handling makes DVQ an optimal connector for high-resolution data, videos, or volumetric data, providing potential for fields such as medical imaging or remote sensing. Furthermore, our experiments show that DVQ handles differen-

tiable bounding box inputs and outputs with little or no performance reduction, making it well-suited for end-to-end trainable systems consisting of multiple models. Finally, DVQ enables the LLM to dynamically query image information conditioned on current model states. This may make it very useful in approaches like visual reasoning with adaptive vision tokens or dynamic budgets. Further discussion on future work is provided in Appendix B.

Limitations Despite its effectiveness, DVQ has a few limitations. Compared to Patch Project and DeepStack, it adds complexity to the LLM layers and token handling, making models slightly more computational expensive but also harder to serve at scale. Furthermore, DVQ does not consider previous textual tokens when refining its reference points. While this simplifies caching and avoids missing visual information, it also limits the compression ratio. OCR remains a limitation, as our sparse point-based sampling struggles with dense text, though this gap narrows with larger token budgets or higher-resolution inputs. Finally, our study focuses on natural images, and further research is needed to study DVQ in other domains or more complex modalities like videos or volumetric data.

6 Related Work

Vision connectors and token efficiency Most VLMs adopt a patch-projection connector, popularized by LLaVA (Liu et al., 2023). To reduce token counts, subsequent work uses spatial aggregation, such as C-Abstractor (Cha et al., 2024), PixelUnshuffle (Chen et al., 2024), and TokenPacker (Li et al., 2025b). Others employ learned query-based resampling, like the Perceiver Resampler in Flamingo (Alayrac et al., 2022) and the Q-Former (Li et al., 2023b). These directions have been extended through hierarchical/adaptive compression in MQT-LLaVA (Hu et al., 2024), LLaVA-1.5-M3 (Cai et al., 2025), METEOR (Liu et al., 2025c), and AVG-LLaVA (Lan et al., 2025), and through distillation-based compression in VoCo-LLaMA (Ye et al., 2025). Our DVQ connector combines the strengths of both families: uniformly sampled reference points promote full-image coverage while layer-wise refinement adaptively focuses on key regions. The compression ratio is simply the number of queries, avoiding the need for any extra compression modules. InstructBLIP (Dai et al., 2023) pioneered instruction-conditioned visual abstraction, later advanced by pre-fusion methods such as LLaMA-VID (Li et al., 2025c) and LLaVA-Mini (Zhang et al., 2025b), which achieve high compression but introduce quadratic costs during prefusion and prevent vision-token caching. Our DVQ connector instead queries vision information via position-based priors, entirely independent of text and extra prefusion layers. Alternative connectors inject vision features directly into LLM layers, e.g., through cross-attention in Flamingo (Alayrac et al., 2022) and PaLM2-VAdapter (Xiao et al., 2024), or by distributing patch features across layers as in DeepStack (Meng et al., 2024). DVQ builds on this idea while enforcing explicit positional priors and using deformable attention to encourage full-image coverage with adaptive focus. For completeness, token-efficiency methods such as PruMerge (Shang et al., 2025) and FastV (Chen et al., 2025) prune or merge visual tokens after the connector, making them orthogonal to our contribution and outside the scope of this work.

Deformable and spatially grounded attention Explicit spatial priors have been used to ground transformer queries, from anchor boxes in DAB-DETR (Liu et al., 2022) to learned reference points in deformable attention (Zhu et al., 2021). Inspired by deformable convolutions (Dai et al., 2017), deformable attention sparsely samples features around those reference points, driving advances in detection (Zhang et al., 2022; Liu et al., 2025a), vision transformers (Xia et al., 2022) and grounded VLMs (Rasheed et al., 2024). However, deformable attention has primarily been applied in vision-side modules (encoders, detection heads). DVQ extends this approach by bringing deformable cross-attention directly into LLM layers, enabling efficient, spatially grounded queries of visual features.

Spatial reasoning in VLMs VLMs commonly represent spatial locations in inputs or outputs as text coordinates: Kosmos-2(Peng et al., 2023) and Molmo (Deitke et al., 2025) use special location tokens for pointing and grounding, while Shikra (Chen et al., 2023) and Ferret (You et al., 2024), like most modern VLMs, treat coordinates as plain numerical text. Octopus (Zhao et al., 2024) employs an object detector inserted within the LLM, but still keeps spatial reasoning in this separate module. Segmentation-focused models like LISA (Lai et al., 2024) similarly rely on external decoders. DVQ, on the other hand, integrates spatial reasoning into the LLM layers by querying vision features via deformable cross-attention. This allows the LLM to directly reason spatially, while supporting differentiable location inputs and outputs when needed.

AI USE STATEMENT

In this work, we used generative AI tools to design or provide feedback on research methodology or experiments, and to clean and reformat datasets (specifically for data preprocessing/loading). We have not used generative AI tools to help develop theoretical models or conceptual frameworks, propose or refine hypotheses, implement methods, assist with translation, support qualitative and thematic data analysis, or interpret results; and generating synthetic data sets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, or assisting in the writing of proofs are not applicable to this work.

Additionally, we used generative AI tools for creating or modifying scientific figures or images (only via code generation for plotting), creating or editing software code (mainly for data preprocessing/loading, plotting, and utility scripts, as well as IDE-based autocomplete in all parts of the code), sourcing/searching for information, editing the research paper to improve readability, and identifying relevant literature.

We have reviewed all AI-assisted work. All AI-generated code was carefully read, verified, and tested by the authors. AI-based literature search was only used to complement non-AI-based literature search and review but did not replace it. Literature identified by AI was carefully read by the authors. AI-based feedback on experiments was only used complementary: experiments were not designed solely based on AI feedback without careful consideration and discussion among authors as well as comparison with existing literature. Core ideation and design decisions of the method did not rely on AI assistence. No fully AI-generated figures were used, instead, AI was solely used to generate plotting code, where we carefully verified the generated code and resulting figures. The final manuscript was carefully edited and reviewed by the authors.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our proposed method is described in Sec. 3, with implementation details provided in Appendix C.1. Details of the baselines are provided in Appendix C.3. The experimental setup, including training and evaluation protocols and dataset processing, is described in Appendix C.4. To support full reproducibility, we provide our source code, including configuration files for all models and experiments, in the supplementary materials. All datasets used for training and evaluation are publicly available. To support full reproducibility, we provide our source code, including configuration files for our models and experiments, at an anonymous repository: `https://anonymous.4open.science/r/dvq`.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikoł aj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a Visual Language Model for Few-Shot Learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 23716–23736. Curran Associates, Inc., 2022. doi: 10.52202/068431-1723. URL `https://proceedings.neurips.cc/paper_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html`.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen

Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL Technical Report, November 2025. URL http://arxiv.org/abs/2511.21631. arXiv:2511.21631 [cs.CV].

Mu Cai, Jianwei Yang, Jianfeng Gao, and Yong Jae Lee. Matryoshka Multimodal Models. *International Conference on Learning Representations*, 2025:46254–46272, May 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/722fcbc1a6667f2075d75ea79a1b3552-Abstract-Conference.html.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, pp. 213–229, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58452-8. doi: 10.1007/978-3-030-58452-8_13.

Junbum Cha, Wooyoung Kang, Jonghwan Mun, and Byungseok Roh. Honeybee: Locality-Enhanced Projector for Multimodal LLM. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13817–13827, June 2024. doi: 10.1109/CVPR52733.2024.01311. URL https://ieeexplore.ieee.org/document/10655549/. ISSN: 2575-7075.

Keqin Chen, Zhao Zhang, Weili Zeng, Richong Zhang, Feng Zhu, and Rui Zhao. Shikra: Unleashing Multimodal LLM's Referential Dialogue Magic, July 2023. URL http://arxiv.org/abs/2306.15195. arXiv:2306.15195 [cs.CV].

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An Image is Worth 1/2 Tokens After Layer 2: Plug-and-Play Inference Acceleration for Large Vision-Language Models. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 19–35, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73004-7. doi: 10.1007/978-3-031-73004-7_2.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. Microsoft COCO Captions: Data Collection and Evaluation Server, April 2015. URL http://arxiv.org/abs/1504.00325. arXiv:1504.00325 [cs.CV].

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, Ji Ma, Jiaqi Wang, Xiaoyi Dong, Hang Yan, Hewei Guo, Conghui He, Botian Shi, Zhenjiang Jin, Chao Xu, Bin Wang, Xingjian Wei, Wei Li, Wenjian Zhang, Bo Zhang, Pinlong Cai, Licheng Wen, Xiangchao Yan, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhai Wang. How far are we to GPT-4V? Closing the gap to commercial multimodal models with open-source suites. *Sci. China Inf. Sci.*, 67(12):220101, December 2024. ISSN 1869-1919. doi: 10.1007/s11432-024-4231-5. URL https://doi.org/10.1007/s11432-024-4231-5.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable Convolutional Networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 764–773, October 2017. doi: 10.1109/ICCV.2017.89. URL https://ieeexplore.ieee.org/abstract/document/8237351. ISSN: 2380-7504.

Wenliang Dai, Junnan Li, DONGXU LI, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. InstructBLIP: Towards General-purpose Vision-Language Models with Instruction Tuning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 49250–49267. Curran Associates, Inc., 2023. doi: 10.52202/075280-2142. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/9a6a435e75419a836fe47ab6793623e6-Abstract-Conference.html.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli VanderBilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron

Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittlif, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha Kembhavi. Molmo and PixMo: Open Weights and Open Data for State-of-the-Art Vision-Language Models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 91–104, June 2025. doi: 10.1109/CVPR52734.2025.00018. URL https://ieeexplore.ieee.org/document/11092508/. ISSN: 2575-7075.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, Rongrong Ji, Caifeng Shan, and Ran He. MME: A Comprehensive Evaluation Benchmark for Multimodal Large Language Models. *Advances in Neural Information Processing Systems*, 38, 2025. doi: 10.52202/085713-4899. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/d79a27cf2772fe00be7f341efc0eb517-Abstract-Datasets_and_Benchmarks_Track.html.

Gemma Team. Gemma 3 Technical Report, March 2025. URL http://arxiv.org/abs/2503.19786. arXiv:2503.19786 [cs.CL].

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6904–6913, 2017. doi: 10.1109/CVPR.2017.670. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Goyal_Making_the_v_CVPR_2017_paper.html.

Danna Gurari, Qing Li, Abigale J. Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P. Bigham. VizWiz Grand Challenge: Answering Visual Questions from Blind People. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3608–3617, June 2018. doi: 10.1109/CVPR.2018.00380. URL https://ieeexplore.ieee.org/document/8578478/. ISSN: 2575-7075.

Wenbo Hu, Zi-Yi Dou, Liunian Harold Li, Amita Kamath, Nanyun Peng, and Kai-Wei Chang. Matryoshka Query Transformer for Large Vision-Language Models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50168–50188. Curran Associates, Inc., 2024. doi: 10.52202/079017-1588. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/59c147c7d4fdb732daea3064eab949bf-Abstract-Conference.html.

Drew A. Hudson and Christopher D. Manning. GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6693–6702, June 2019. doi: 10.1109/CVPR.2019.00686. URL https://ieeexplore.ieee.org/document/8953451/. ISSN: 2575-7075.

Soumya Jahagirdar, Walid Bousselham, Anna Kukleva, and Hilde Kuehne. When LLaVA Meets Objects: Token Composition for Vision-Language-Models, February 2026. URL http://arxiv.org/abs/2602.04864. arXiv:2602.04864 [cs.CV].

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conferenceon Learning Representations*, 2014. doi: 10.48550/arXiv.1412.6980. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs.LG].

Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. LISA: Reasoning Segmentation via Large Language Model. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9579–9589, June 2024. doi: 10.1109/CVPR52733.2024.00915. URL https://ieeexplore.ieee.org/document/10658574/. ISSN: 2575-7075.

Zhibin Lan, Liqiang Niu, Fandong Meng, Wenbo Li, Jie Zhou, and Jinsong Su. AVG-LLaVA: An Efficient Large Multimodal Model with Adaptive Visual Granularity. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16852–16869, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.865. URL https://aclanthology.org/2025.findings-acl.865/.

Hugo Laurençon, Lucile Saulnier, Leo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander Rush, Douwe Kiela, Matthieu Cord, and Victor Sanh. OBELICS: An Open Web-Scale Filtered Dataset of Interleaved Image-Text Documents. *Advances in Neural Information Processing Systems*, 36:71683–71702, December 2023. doi: 10.52202/075280-3138. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/e2cfb719f58585f779d0a4f9f07bd618-Abstract-Datasets_and_Benchmarks.html.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. SEED-Bench: Benchmarking Multimodal LLMs with Generative Comprehension, August 2023a. URL http://arxiv.org/abs/2307.16125. arXiv:2307.16125 [cs.CL].

Hongyu Li, Jinyu Chen, Ziyu Wei, Shaofei Huang, Tianrui Hui, Jialin Gao, Xiaoming Wei, and Si Liu. LLaVA-ST: A Multimodal Large Language Model for Fine-Grained Spatial-Temporal Understanding. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8592–8603, Nashville, TN, USA, June 2025a. IEEE. ISBN 979-8-3315-4364-8. doi: 10.1109/CVPR52734.2025.00803. URL https://ieeexplore.ieee.org/document/11092641/.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 19730–19742. PMLR, July 2023b. URL https://proceedings.mlr.press/v202/li23q.html.

Wentong Li, Yuqian Yuan, Jian Liu, Dongqi Tang, Song Wang, Jie Qin, Jianke Zhu, and Lei Zhang. TokenPacker: Efficient Visual Projector for Multimodal LLM. *Int J Comput Vis*, 133(10):6794–6812, October 2025b. ISSN 1573-1405. doi: 10.1007/s11263-025-02491-7. URL https://doi.org/10.1007/s11263-025-02491-7.

Yanwei Li, Chengyao Wang, and Jiaya Jia. LLaMA-VID: An Image is Worth 2 Tokens in Large Language Models. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 323–340, Cham, 2025c. Springer Nature Switzerland. ISBN 978-3-031-72952-2. doi: 10.1007/978-3-031-72952-2_19.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Xin Zhao, and Ji-Rong Wen. Evaluating Object Hallucination in Large Vision-Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 292–305, Singapore, December 2023c. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.20. URL https://aclanthology.org/2023.emnlp-main.20/.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34892–34916. Curran Associates, Inc., 2023. doi: 10.52202/075280-1516. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved Baselines with Visual Instruction Tuning. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26286–26296, June 2024. doi: 10.1109/CVPR52733.2024.02484. URL https://ieeexplore.ieee.org/document/10655294/. ISSN: 2575-7075.

Shilong Liu, Feng Li, Hao Zhang, Xiao Yang, Xianbiao Qi, Hang Su, Jun Zhu, and Lei Zhang. DAB-DETR: Dynamic Anchor Boxes are Better Queries for DETR. In *ICLR*, March 2022. doi: 10.48550/arXiv.2201.12329. URL http://arxiv.org/abs/2201.12329. arXiv:2201.12329 [cs.CV].

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding DINO: Marrying DINO with Grounded Pre-training for Open-Set Object Detection. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV*

2024, pp. 38–55, Cham, 2025a. Springer Nature Switzerland. ISBN 978-3-031-72970-6. doi: 10.1007/978-3-031-72970-6_3.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. MMBench: Is Your Multi-modal Model an All-Around Player? In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 216–233, Cham, 2025b. Springer Nature Switzerland. ISBN 978-3-031-72658-3. doi: 10.1007/978-3-031-72658-3_13.

Yuchen Liu, Yaoming Wang, Bowen Shi, Xiaopeng Zhang, Wenrui Dai, Chenglin Li, Hongkai Xiong, and Qi Tian. METEOR: Multi-Encoder Collaborative Token Pruning for Efficient Vision Language Models. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 21492–21504, October 2025c. doi: 10.1109/ICCV51701.2025.01996. URL https://ieeexplore.ieee.org/document/11446273/. ISSN: 2380-7504.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. In *Advances in Neural Information Processing Systems*, volume 35, pp. 2507–2521. Curran Associates, Inc., 2022. doi: 10.52202/068431-0182. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/11332b6b6cf4485b84afadb1352d3a9a-Abstract-Conference.html.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L. Yuille, and Kevin Murphy. Generation and Comprehension of Unambiguous Object Descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11–20, 2016. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Mao_Generation_and_Comprehension_CVPR_2016_paper.html.

Lingchen Meng, Jianwei Yang, Rui Tian, Xiyang Dai, Zuxuan Wu, Jianfeng Gao, and Yu-Gang Jiang. DeepStack: Deeply Stacking Visual Tokens is Surprisingly Simple and Effective for LMMs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 23464–23487. Curran Associates, Inc., 2024. doi: 10.52202/079017-0739. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/29cd7f8331d13ede6dc6d6ef3dfacb70-Abstract-Conference.html.

Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. Kosmos-2: Grounding Multimodal Large Language Models to the World, July 2023. URL http://arxiv.org/abs/2306.14824. arXiv:2306.14824 [cs.CL].

Qwen Team. Qwen3.5-Omni Technical Report, April 2026. URL http://arxiv.org/abs/2604.15804. arXiv:2604.15804 [cs.CL].

Hanoona Rasheed, Muhammad Maaz, Sahal Shaji, Abdelrahman Shaker, Salman Khan, Hisham Cholakkal, Rao M. Anwer, Eric Xing, Ming-Hsuan Yang, and Fahad S. Khan. GLaMM: Pixel Grounding Large Multimodal Model. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13009–13018, June 2024. doi: 10.1109/CVPR52733.2024.01236. URL https://ieeexplore.ieee.org/document/10655326/. ISSN: 2575-7075.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, August 2020. Association for Computing Machinery. ISBN 978-1-4503-7998-4. doi: 10.1145/3394486.3406703. URL https://dl.acm.org/doi/10.1145/3394486.3406703.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. LLaVA-Prumerge: Adaptive Token Reduction for Efficient Large Multimodal Models. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22857–22867, October 2025. doi: 10.1109/ICCV51701.2025.02122. URL https://ieeexplore.ieee.org/document/11445607/. ISSN: 2380-7504.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA Models That Can Read. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8309–8318, June 2019. doi: 10.1109/CVPR.2019.00851. URL https://ieeexplore.ieee.org/document/8953586/. ISSN: 2575-7075.

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. SigLIP 2: Multilingual Vision-Language Encoders with Improved Semantic Understanding, Localization, and Dense Features, February 2025. URL http://arxiv.org/abs/2502.14786. arXiv:2502.14786 [cs.CV].

Zhuofan Xia, Xuran Pan, Shiji Song, Li Erran Li, and Gao Huang. Vision Transformer with Deformable Attention. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4784–4793, June 2022. doi: 10.1109/CVPR52688.2022.00475. URL https://ieeexplore.ieee.org/document/9878689/. ISSN: 2575-7075.

Junfei Xiao, Zheng Xu, Alan Yuille, Shen Yan, and Boyu Wang. PaLM2-VAdapter: Progressively Aligned Language Model Makes a Strong Vision-language Adapter, June 2024. URL http://arxiv.org/abs/2402.10896. arXiv:2402.10896 [cs.CV].

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, May 2025. URL http://arxiv.org/abs/2505.09388. arXiv:2505.09388 [cs.CL].

Xubing Ye, Yukang Gan, Xiaoke Huang, Yixiao Ge, and Yansong Tang. VoCo-LLaMA: Towards Vision Compression with Large Language Models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29836–29846, June 2025. doi: 10.1109/CVPR52734.2025.02777. URL https://ieeexplore.ieee.org/document/11093971/. ISSN: 2575-7075.

Haoxuan You, Haotian Zhang, Zhe Gan, Xianzhi Du, Bowen Zhang, Zirui Wang, Liangliang Cao, Shih-Fu Chang, and Yinfei Yang. Ferret: Refer and Ground Anything Anywhere at Any Granularity. *International Conference on Learning Representations*, 2024:57153–57180, May 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/fd6613131889a4b656206c50a8bd7790-Abstract-Conference.html.

Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection, July 2022. URL http://arxiv.org/abs/2203.03605. arXiv:2203.03605 [cs.CV].

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. LMMs-Eval: Reality Check on the Evaluation of Large Multimodal Models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 881–916, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.51. URL https://aclanthology.org/2025.findings-naacl.51/.

Shaolei Zhang, Qingkai Fang, Yang Yang, and Yang Feng. LLaVA-Mini: Efficient Image and Video Large Multimodal Models with One Vision Token. *International Conference on Learning Representations*, 2025:53285–53310, May 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/8400a2ec9ba85bfdb41eb2a584c0876d-Abstract-Conference.html.

Chuyang Zhao, Yuxin Song, Junru Chen, Kang Rong, Haocheng Feng, Gang Zhang, Shufan Ji, Jingdong Wang, Errui Ding, and Yifan Sun. Octopus: A Multi-modal LLM with Parallel Recognition and Sequential Understanding. In *Advances in Neural Information Processing Systems*, volume 37, pp. 90009–90029. Curran Associates, Inc., 2024. doi: 10.52202/079017-2858. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a3eadeebbc9eecd621086f6978865a85-Abstract-Conference.html.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable Transformers for End-to-End Object Detection. In *ICLR*, March 2021. doi: 10.48550/arXiv.2010.04159. URL http://arxiv.org/abs/2010.04159. arXiv:2010.04159 [cs.CV].

A Additional Results and Ablation Studies

A.1 LLMs and Scales

Table 4: Comparison of DVQ and DeepStack with different LLM backbones. DVQ performs especially well with small LLMs, outperforming DeepStack on them and having smaller performance drops when reducing LLM size.

LLM	Connector	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]
Qwen-3 4B (default)	DVQ	**64.6**	53.5	**67.9**
	DeepStack	63.0	**60.7**	55.3
Qwen-3.5 4B*	DVQ	72.6	68.8	85.9
	DeepStack	**73.0**	**71.8**	**86.0**
Qwen-3.5 0.8B*	DVQ	**64.4**	**58.9**	**78.8**
	DeepStack	62.2	56.0	67.5
Gemma 3 4B*	DVQ	**67.0**	63.1	**73.9**
	DeepStack	62.6	**68.8**	61.3

* LLM pretraining included vision data.

We study how our DVQ connector behaves across different LLM variants and scales. Therefore, we run small-scale experiments (see Sec. 4.1) with $|\mathcal{V}| = 1024$ vision tokens, using Qwen-3 4B (Yang et al., 2025) as our default, the Qwen-3.5 series (Qwen Team, 2026) at two different scales, and Gemma 3 4b (Gemma Team, 2025). Unlike Qwen-3, Qwen-3.5 and Gemma 3 were created as unified vision-language foundation models trained on multimodal image–text data. For comparability, we keep SigLIP-2 instead of using the original vision encoders of Qwen-3.5 or Gemma 3. For Qwen-3.5, we additionally consider the 0.8B variant to evaluate DVQ at a smaller scale.

We do not evaluate larger LLMs because our primary focus is on small-scale settings. Specifically, we choose 4B as the largest model size because it can still be trained on a single NVIDIA A100 for short sequences or when the LLM is frozen. We also note that the instruction-tuned Qwen-3 models are not available below 4B. Results are shown in Tab. 4.

DVQ performs well on small-scale LLMs. Scaling Qwen-3.5 down from 4B to 0.8B reduces DeepStack's average performance by 19.4%, whereas DVQ drops by only 11.3%. As a result, DVQ outperforms DeepStack across all tasks at 0.8B, by 8.5% on average, while both connectors perform comparably at 4B, with DVQ reaching 98.4% of DeepStack's average performance. This suggests that DVQ is particularly robust to reductions in LLM scale.

DVQ performs well across different LLMs. Comparing Qwen-3 with Qwen-3.5 and Gemma 3, we observe that both connectors generally benefit from using unified vision-language pretrained LLMs, with consistent gains on TextVQA and RefCOCOg. These models have been trained on multimodal data related to our downstream tasks, making them easy to adapt to a new vision encoder and connector. Notably, DVQ still benefits from a vision-language pretrained LLM, even though it is a substantially different vision connector than the one used during LLM pretraining. With Qwen-3.5 4B, DeepStack closes the performance gap to DVQ observed on Qwen-3, while with Gemma 3, DVQ further widens its advantage on GQA. Overall, these results highlight that DVQ performs consistently across LLM architectures and scales, suggesting that its benefits are not tied to the specific architecture or pretraining of an individual LLM.

A.2 Extended Token Budget

As DVQ's visual tokens are not tied to the number of patches, we can increase the token budget $|\mathcal{V}|$ further beyond the available number of image patches. We study, whether this improves the predictions on tasks that require more fine-grained access to vision information.

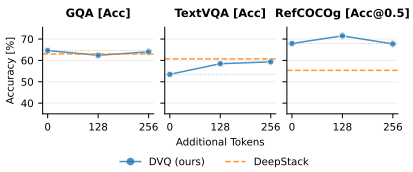


Figure 4: Increased token budgets, adding 128 or 256 vision tokens to the full $|\mathcal{V}| = 1024$ budget. Despite little impact on GQA and RefCOCOg, this notably improves the TextVQA scores.

DVQ's OCR performance improves with additional token budget. As shown in Fig. 4, adding 256 vision tokens (totaling at $|\mathcal{V}| = 1280$) almost completely closes the gap to DeepStack on TextVQA, reaching 97.7% of its score. This shows that for OCR tasks, additional tokens can provide the needed more fine-grained information. On other tasks, no notable changes in scores are observed, but DVQ remains on par or superior to DeepStack.

A.3 Vision Connector Placement

The DVQ vision connector can be inserted at arbitrary LLM layers. To understand the impact of this layer choice, we systematically explore several DVQ insertion variants across the 36-layer Qwen-3 4B LLM. Specifically, we test inserting DVQ at either 12 (33% density) or 18 (50% density) of the total layers. For each density, we vary the interleaving gap between consecutive DVQ layers (every layer, every second layer, or every third layer) which determines the resulting layer span (the distance from the first to the last inserted layer). We also shift the placement window while keeping density and interleaving fixed (e.g., for 12 layers with interleave 1, we evaluate the contiguous blocks layers 1–12, 13–24, and 25–36), allowing us to identify the relevance of specific layer placements within the LLM. Results are shown in Fig. 5.

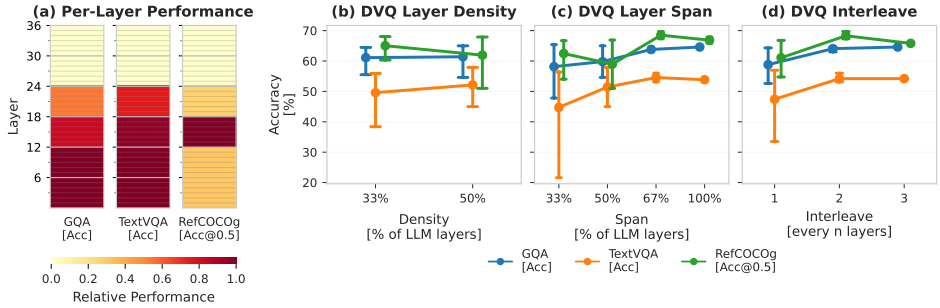


Figure 5: DVQ layer placement within the LLM. The figure compares the per-task performances for (a) each insertion window, (b) different DVQ layer densities, i.e. the fraction of LLM layers that use DVQ, (c) different DVQ layer spans, i.e. the distance from the first to the last DVQ layer, and (d) different interleaves, i.e. gaps between consecutive DVQ layers.

Optimal DVQ layer placement depends on the target task. While for GQA the first 12 layers are most important, for TextVQA the first 18 layers are important. RefCOCOg, on the other hand, profits the most from layers 12-18, while only weakly benefiting from the first 12. Overall, all tasks seem to benefit more from the first half of the LLM layers. We hypothesize that early layers benefit from low-level visual features, while mid-layers are better positioned for the semantic integration required by localization tasks.

The number of DVQ layers, and thus injection points, can be reduced as long as the relevant LLM layers are covered. For all tasks it is beneficial to cover a wide range of LLM layers (Fig. 5c). A larger layer span assures that the most important LLM layers are covered, i.e. layers 1-24 (as identified in Fig. 5a). One option to achieve this is using a higher DVQ layer density (i.e.

more DVQ layers). In Fig. 5b, comparing the use of 12 DVQ layers (33.3% DVQ layer density) and 18 DVQ layers (50% density), more DVQ layers (i.e. more DVQ layer density) slightly improve GQA and TextVQA scores while RefCOCOg scores decrease, considering the average over different configurations. However, a higher layer span can also be achieved with fewer DVQ layers but more interleave between those layers (Fig. 5d).

Table 5: Comparison of using 18 (default) and 12 DVQ layers, i.e. injection points, each with an interleave of 2 and spanning the most relevant layers identified in Fig. 5. When using 18 DVQ layers, we inject at 50% of all LLM layers, covering the full span of LLM layers, with 12 we only inject at 33.3% of the layers spanning the first 66.7% of the LLM.

DVQ Layers	Density	$\|\mathcal{V}\| = 1024$			$\|\mathcal{V}\| = 256$		
		GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]
18 (default)	50%	**64.6**	53.5	67.9	**64.2**	**49.8**	63.3
12	33%	**64.6**	**56.0**	**69.7**	60.8	46.8	**63.9**

Assuming the relevant layers are covered, in Tab. 5 we now specifically compare two configurations with an interleave of 2, namely 18 DVQ layers (injecting at 50% of all LLM layers, covering the full span of LLM layers) and 12 DVQ layers (injecting at 33.3% of the layers spanning 66.7% of the LLM). Reducing the number of DVQ layers from 18 to 12 does not degrade the performance notably on most tasks. With $|\mathcal{V}| = 1024$ vision tokens, the performance even slightly improves. This confirms that DVQ performs well even with fewer layers as long as the most important LLM layers are covered.

DVQ layer parameters can be shared. In Tab. 6, we study how DVQ's performance is impacted when DVQ's parameters are shared across all layers. Specifically, we share all parameters except the down- and up-projections, i.e. we share DVQ's deformable attention, FFN, and offset prediction parameters. We exclude the down- and up-projections from sharing because this allows filtering the features needed for individual LLM layers. We found that sharing DVQ's parameters only slightly impacts the performance, leading to only slight degradations on most tasks (and improvements on TextVQA at 1024 tokens). This highlights that DVQ does not solely benefit from more parameters but instead can learn a generalizable way (across LLM layers) to query vision information as needed.

Table 6: Impact of DVQ layer parameter sharing. We compare the default DVQ configuration with individual parameters per layer with a DVQ configuration where all DVQ parameters (except down- and up-projections) are shared accross all DVQ layers. In both scenarios, we inject the DVQ layers at 18 LLM layers (the default). DVQ's parameters can be shared with only minor performance impacts.

Connector	Shared	$\|\mathcal{V}\| = 1024$			$\|\mathcal{V}\| = 256$		
		GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]
DVQ	no	**64.6**	53.5	**67.9**	**64.2**	**49.8**	**63.3**
DVQ	yes	62.4	**56.8**	64.6	63.8	44.9	62.3

A.4 Vision Feature Maps and Levels

Table 7: Comparison of DVQ with different vision feature maps. We compare the default setup (using early vision feature maps and downsampled feature maps) with variants that either use no downsampeld feature map or no early vision feature maps.

Feature maps in DVQ	GQA [Acc]	TextVQA [Acc]	RefCOCOg [Acc@0.5]
Default (early and downsampled feature maps)	64.6	53.5	**67.9**
No downsampled vision feature maps	**65.2**	**55.7**	63.5
No early vision feature maps	63.4	54.7	66.6

DVQ can directly access hierarchical vision feature maps, using both early vision feature maps (vision layers 9 and 18, similar to DeepStack) and downsampled versions of the final feature maps (downsampled to 16×16, 8×8, and 4×4). To understand their relevance, we train DVQ without these sets of vision features. Results are shown in Tab. 7.

Early and downsampled vision feature maps are both relevant for localization, while less relevant for other tasks. We found that either removing early or downsampled vision feature maps, negatively impacts the performance on RefCOCOg, with downsampled feature maps showing the highest relevance. On other tasks, these feature maps have only minor impacts, without downsampled feature maps the performance is even slightly improved.

A.5 Computational Costs

Table 8: Computational costs of our DVQ connector compared to DeepStack. For DVQ we also consider the variant with shared parameters (see Tab. 6). We show the number of parameters and FLOPs at two different vision token budgets, namely $|\mathcal{V}| = 1024$ and $|\mathcal{V}| = 256$, assuming $|\mathcal{T}| = 1024$ textual tokens.

Connector	Parameters [B]					TFLOPs				
	LLM	Vision	Projector	DVQ	Total	LLM	Vision	Projector	DVQ	Total
$\lvert\mathcal{V}\rvert = 1024, \lvert\mathcal{T}\rvert = 1024$										
DVQ	4.02	0.046	0.157	0.655	4.878	16.505	0.849	0.085	1.604	19.070
DVQ (shared)	4.02	0.046	0.157	0.145	4.368	16.505	0.849	0.085	1.604	19.070
DeepStack	4.02	0.046	0.051	–	4.117	16.505	0.849	0.105	–	17.459
$\lvert\mathcal{V}\rvert = 256, \lvert\mathcal{T}\rvert = 1024$										
DVQ	4.02	0.046	0.157	0.653	4.876	10.315	0.849	0.085	0.497	11.746
DVQ (shared)	4.02	0.046	0.157	0.143	4.366	10.315	0.849	0.085	0.497	11.746
DeepStack	4.02	0.046	0.068	–	4.134	10.315	0.849	0.034	–	11.198

We study the computational costs of our DVQ connector and compare it to DeepStack. We estimate the FLOPs at two different vision token budgets, namely $|\mathcal{V}| = 1024$ and $|\mathcal{V}| = 256$, assuming $|\mathcal{T}| = 1024$ textual tokens. Results are shown in Tab. 8, where we additionally show the number of total (trainable or not) parameters within the LLM with the corresponding vision connector. Overall, compared to DeepStack, DVQ increases the number of parameters by about 18%, while increasing the FLOPs by only 9% at $|\mathcal{V}| = 1024$, and even only 5% at $|\mathcal{V}| = 256$. When sharing DVQ parameters, the total parameter count even only increases by 6% (the FLOPs are not affected by sharing). To set this into context: increasing the vision budget of DeepStack from $|\mathcal{V}| = 256$ to $|\mathcal{V}| = 1024$ increases the FLOPs by 56%, showing the extra cost of DVQ remains quite low.

A.6 Qualitative Model Inspection

We analyze how reference points of vision queries are distributed over the image space and how they evolve over the DVQ layers. For this we plot the reference point density at different layers, namely the initial reference points ($i = 0$), and at the end of each 12-layer block (at layers 11, 23, and 35). Additionally, we show the directional trends of the points to the next block. For this we compute the offsets for each reference point and, for better visibility, interpolate them on a regular grid. In Fig. 6, we show examples from the COCO test set using our general purpose model at $|\mathcal{V}| = 1024$.

Vision queries steer towards foreground objects in simple scenes. If a scene is relatively simple, containing one or a few foreground objects with relatively irrelevant background, then the reference points of the vision queries quickly steer towards these primary objects. This makes them focus on those foreground objects already in early layers, while fewer vision queries are used for the less relevant. This can be seen especially well on samples 1 and 2, but is also indicated in samples 3-5.

In complex scenes, the vision queries are more distributed over the whole scene. With increasing complexity, the focus on foreground objects is more relaxed. For example, in samples 3-6, the reference points first focus on the most dominant primary object(s), but then explore larger parts of the scene. This can be seen especially well in layer 23, where reference points start to loosen the



Figure 6: Qualitative reference point analysis, showing the reference point density at different layers, namely the initial reference points ($i = 0$), and at the end of each 12-LLM-layer block (at LLM layers 11, 23, and 35). Additionally, we show the directional trends of the points to the next block. The example images are ordered by scene complexity, starting at the top with simpler scenes (with few primary foreground objects) and going to complex scenes at the bottom (with many objects distributed over the scenes). For simple scenes, the reference points quickly focus on the primary foreground objects. With increasing complexity, the reference points start to explore the scene further in later layers (especially in layer 23), while in complex scenes the focus remains more distributed across the whole scene. Overall, this demonstrates that the vision queries can adaptively focus on relevant image regions while also keeping a more widespread coverage in complex scenes.

focus from the foreground and explore larger parts of the image. While a slight focus remains on the foreground objects, this focus is less extreme than in samples 1 and 2. In even more complex scene, like in samples 7 and 8, reference points stay more distributed and the scene is explored as a whole.

A.7 General Purpose Vision-Language Modeling: Benchmark Results and Baselines

Tab. 9 shows detailed results on the LLaVA-1.5 training pipeline. We show the results of our own experiments (Sec. 4.2) including our DVQ method and DeepStack, as well as Patch Project and Perceiver. Additionally, we also show reported scores of existing methods trained on LLaVA-1.5 or similar datasets. We include the reported scores of LLaVA-1.5, as an example of Patch Project, and the original DeepStack (DeepStack-L and DeepStack-V). Furthermore, we include common examples of vision token compression or distillation methods, including InstructBLIP (Dai et al., 2023), Mask-LLaVA (Jahagirdar et al., 2026), IDEFICS (Laurençon et al., 2023), TokenPacker (Li et al., 2025b), LLaVA-1.5-M3 (Cai et al., 2025), MQT-LLaVA (Hu et al., 2024), LLaVA-1.5-M3 (Cai et al., 2025), VoCo-LLaMA (Ye et al., 2025), and METEOR (Liu et al., 2025c). Finally, we also show image-text pre-fusion method, namely LLaMA-VID (Li et al., 2025c) and LLaVA-Mini (Zhang et al., 2025b).

Comparison with DeepStack, Patch Project, and Perceiver in our experiments We compare our DVQ method with DeepStack, Patch Project, and Perceiver in a controlled setting, using the same LLM, vision encoder, image resolution, and vision token budgets. At the full 1024-token budget, DVQ closely matches DeepStack and Patch Project on most tasks (except OCR-heavy tasks), while outperforming them on ScienceQA. At 256 tokens, DVQ further closes the gap with DeepStack, leading on VizWiz, ScienceQA, MMBench, and POPE. At 64 tokens, DVQ beats DeepStack by 14.1 points on VizWiz and 4.6 points on ScienceQA, while remaining competitive on MMBench and POPE. The Perceiver-based compression does not perform well in general. At 256 tokens, it only shows strong performance on ScienceQA (slightly better than DeepStack but lower than DVQ), while underperforming severly on other tasks. This is consistent with the low performance in the small-scale scenario (Sec. 4.1), indicating that learning a Perceiver from scratch during instruction tuning does not perform well and Perceiver-based compression would instead require additional large-scale pretraining, consistent with findings by Xiao et al. (2024). Overall, DVQ closely matches DeepStack's performance at high budgets, while DVQ is more robust under aggressive token compression.

Comparison with image-text prefusion Methods such as LLaMA-VID and LLaVA-Mini report very small LLM-side vision-token counts, but they rely on large pre-fusion budgets, typically 576 or even 2880 tokens. They often match or approach, and sometimes even exceed, LLaVA-1.5's performance, but the pre-fusion component still incurs long-sequence quadratic attention and extra computation. DVQ, on the other hand, avoids the extra pre-fusion stage. At 256 tokens, DVQ already exceeds these pre-fusion methods on ScienceQA, TextVQA, MMBench, and SEED, while pre-fusion methods are stronger at VizWiz, GQA, and MME. At 64 tokens, DVQ still outperforms them on most of these tasks but is now also competetive at VizWiz. Overall, while image-text prefusion enables high compression ratios, it introduces heavy extra computational costs, while DVQ avoids extra prefusion components.

Comparison with token compression or distillation Compared with token-compression or distillation methods, DVQ is competitive or better at matched budgets. At 64 tokens, DVQ clearly surpasses Mask-LLaVA (57 tokens), MQT-LLaVA (64 tokens), and VoCo-LLaMA (64 tokens) on almost all tasks, though its GQA score is slightly lower than some baselines. At 256 tokens, DVQ outperforms MQT-LLaVA (256 tokens), TokenPacker (144 tokens), LLaVA-1.5-M3 (144 tokens) and METEOR (242 tokens) on VQA, ScienceQA, and MMBench, although METEOR's high-resolution setting gives better GQA and TextVQA scores. A caveat is that DVQ uses a Qwen3-4B backbone rather than Vicuna-7B, so a direct comparison needs to be done with caution. But comparing DVQ with reduced budgets against Patch Projection at the full budget still shows that DVQ mostly retains or improves performance on several tasks, especially on VizWiz (where it even improves), ScienceQA, MMBench, MME, and POPE.

Table 9: Comparison of LLaVA-trained models on VLM benchmarks. The upper parts shows existing baselines (image-text pre-fusion, token compression or distillation, and common vision connectors), each trained on LLaVA-1.5 or similar datasets, within their respective training setups (as reported from their papers). In the lower part, we show results of our own experiments (Sec. 4.2), training with different vision budgets on LLaVA-1.5.

Model	LLM	Image Res.	Vision Tokens	VQAv2	GQA	VisWiz	SciQA IMG	TextVQA OCR	TextVQA –	MMB en	MME perc	SEED	POPE adv
LMMs with image-text pre-fusion													
† denotes the number of vision tokens during image-text pre-fusion.													
LLaMA-VID	Vicuna-7B	336^2	2 (576†)	79.3	64.3	54.2	68.3	–	–	65.1	1521.4	59.9	–
LLaVA-Mini	Vicuna-7B	336^2	1 (576†)	77.6	60.9	56.2	70.4	57.0	–	65.6	1466.0	58.5	–
LLaVA-Mini (64)	Vicuna-7B	336^2	64 (576†)	78.5	61.6	–	–	–	–	67.5	–	–	–
LLaVA-Mini-HD	Vicuna-7B	672^2	64 (2880†)	78.9	61.8	58.5	69.7	59.1	–	67.5	1476.8	60.2	–
LMMs with vision token compression or distillation													
InstructBLIP	Vicuna-7B	224^2	32	–	49.2	34.5	60.5	50.1	–	36.0	–	53.4	–
Mask-LLaVA	Vicuna-7B	336^2	57	74.8	60.6	51.8	68.8	–	–	63.1	1415.0	–	–
IDEFICS-9B	LLaMA-7B	224^2	64	50.9	38.4	35.5	44.2	25.9	–	48.2	–	–	–
TokenPacker	Vicuna-7B	336^2	64	77.2	61.1	50.7	–	–	–	64.1	–	–	–
TokenPacker	Vicuna-7B	336^2	144	77.9	61.9	52.0	–	–	–	65.1	–	–	–
LLaVA-1.5-M^3	Vicuna-7B	336^2	36	–	60.3	52.8	–	–	–	64.8	–	58.0	–
LLaVA-1.5-M^3	Vicuna-7B	336^2	144	–	61.3	53.1	–	–	–	66.4	–	59.7	–
LLaVA-1.5-M^3	Vicuna-7B	336^2	576	–	61.9	54.9	–	–	–	65.9	–	60.6	–
MQT-LLaVA (2)	Vicuna-7B	336^2	2	61.0	50.8	48.5	65.0	–	–	54.4	1144.0	–	–
MQT-LLaVA (64)	Vicuna-7B	336^2	64	75.3	60.0	51.5	67.0	–	–	63.5	1464.3	–	–
MQT-LLaVA (256)	Vicuna-7B	336^2	256	76.8	61.6	53.1	67.6	–	–	64.3	1434.5	–	–
VoCo-LLaMA	Vicuna-7B	336^2	1	72.3	57.0	–	65.4	–	–	58.8	1323.3	53.7	–
VoCo-LLaMA (64)	Vicuna-7B	336^2	64	75.4	60.4	–	–	–	–	60.5	–	56.3	–
METEOR	Vicuna-7B	1024^2	126	–	62.9	–	70.7	69.4	–	69.7	–	71.7	–
METEOR	Vicuna-7B	1024^2	242	–	63.5	–	71.4	71.1	–	69.8	–	72.8	–
LMMs with common vision connectors (trained on LLaVA-1.5)													
LLaVA-1.5	Vicuna-7B	336^2	576	78.5	62.0	50.0	66.8	58.2	–	64.3	1510.7	58.6	84.2
LLaVA-1.5	Vicuna-13B	336^2	576	80.0	63.3	53.6	71.6	61.3	–	67.7	1531.3	61.6	84.5
DeepStack-L	Vicuna-7B	672^2	576	79.5	63.1	–	–	62.4	–	–	–	60.6	–
DeepStack-V	Vicuna-7B	672^2	576	80.4	64.1	–	–	63.5	–	–	–	62.3	–
Our Experiments (trained on LLaVA-1.5)													
DeepStack	Qwen3-4B	512^2	1024	**82.4**	**62.9**	**56.1**	75.2	**70.8***	**68.8***	**76.5***	**1504.6**	**66.5**	**87.9**
DVQ	Qwen3-4B	512^2	1024	80.0	60.6	51.3	**78.4**	67.3*	58.4*	76.1*	1472.3	64.4	87.7
Patch Project	Qwen3-4B	512^2	1024	82.1	62.8	51.3	75.4	68.1*	66.3*	76.3*	1488.7	65.8	87.6
DeepStack	Qwen3-4B	512^2	256	**80.4**	**61.4**	52.3	75.1	**66.4***	**60.7***	75.1*	**1479.6**	**64.1**	86.4
vs. DeepStack (1024)				−2.4%	−2.4%	−6.8%	**−0.1%**	−6.2%	−11.8%	−1.8%	−1.7%	−3.6%	−1.7%
DVQ	Qwen3-4B	512^2	256	79.3	59.6	**52.9**	**77.5**	65.4*	55.0*	**75.9***	1466.7	63.5	**87.0**
vs. DVQ (1024)				**−0.9%**	**−1.7%**	**+3.1%**	−1.1%	**−2.8%**	**−5.8%**	**−0.3%**	**−0.4%**	**−1.4%**	**−0.8%**
Perceiver	Qwen3-4B	512^2	256	71.31	54.3	39.9	75.8	56.1*	33.3*	66.1*	1203.7	53.3	81.3
DeepStack	Qwen3-4B	512^2	64	**77.8**	**60.0**	44.0	72.6	**63.4***	48.4*	74.7*	**1431.0**	**60.9**	86.0
vs. DeepStack (1024)				**−5.6%**	−4.6%	−21.6	−3.5%	−10.5%	−29.7%	−2.4%	−4.9%	−8.4%	−2.2%
DVQ	Qwen3-4B	512^2	64	77.0	58.3	**58.1**	**77.2**	62.6*	**48.5***	**74.9**	1417.9	60.4	**86.2**
vs. DVQ (1024)				**−3.8%**	**−3.8%**	**+13.3%**	**−1.5%**	**−7.0%**	**−17.0%**	**−1.6%**	**−3.7%**	**−6.2%**	**−1.7%**

*Evaluated on dev splits as evaluation servers are no longer available.

B Future Work and Applications of DVQ

Our DVQ connector does not only support vanilla vision-language modeling but can be integrated into a wide range of more complex applications. Its token efficiency and differentiable localization inputs and outputs may provide major benefits for some of these applications. In the following, we provide simple design blueprints for applications where we think that DVQ may provide benefits.

Video-Language Models with Effective Sequence Compression Video-language models often rely on specifically designed compression modules (Li et al., 2025a), which require trade-offs between spatial and temporal resolution. DVQ, on the other hand, does not require explicit compression modules. To access video information, the deformable attention module in DVQ can be extended to support 2D+T (i.e., three-dimensional) data. Apart from this extension, no further modifications are needed and DVQ implicitly compresses the video information as needed, dynamically balancing spatial and temporal information through reference point refinement.

Multiple Modalities with Dynamic Access DVQ can be extended to integrate several modalities into the LLM. To achieve this, different sets of vision queries can be used. For each modality, we define one such set of queries, where the query/token count can be chosen individually. Each type of vision token does then trigger a modality-specific DVQ component. The corresponding DVQ component then conditions its deformable attention module on the specific modality features. Note that the deformable attention component can be adapted to also support 1D (signal/sequential), 3D (volumetric), or even non-regularly sampled modalities.

Visual Reasoning Chains The DVQ connector uses reference points to lookup visual information. Typically, we start with static (uniformly sampled) initial reference points, which are then refined throughout the DVQ layers. In Sec. 4.3, we studied an alternative approach, where for some vision tokens, the initial reference points are based on given bounding box inputs. However, the DVQ module is even more flexible: reference points can come from any source and vision tokens can be added at any place in the sequence. These properties could be used to dynamically revisit vision information. For example, a model could dynamically predict additional vision tokens as parts of its reasoning (thinking) chain. The initial reference points for such reasoning vision tokens could be based on previous vision tokens, which the model refers to. Alternatively, latent features of previous tokens could be used to predict initial reference points. Overall, this enables the model to decide on a dynamic compute budget and reiterate on vision information as needed, while building on the context of previous vision tokens. Future work may explore these architectural capabilities within post-training approaches.

End-to-End Trainable Systems through Differentiable Bounding Box Inputs and Outputs As shown in Sec. 4.3, DVQ supports differentiable bounding box inputs and outputs. While this property is not needed when encoding or predicting user inputs or outputs, it becomes useful when using a DVQ-based VLM inside a larger system. For example, a small-scale DVQ-based VLM can make initial bounding box predictions that are further refined by a larger-scale, but more expensive, DVQ-based VLM. Furthermore, bounding boxes predicted by DVQ can be used as inputs to an segmentation model, which uses it for guidance or to select relevant parts within the image. Differentiable bounding box inputs and outputs, make it possible to train, or fine-tune, such systems end-to-end.

C Experiment and Implementation Details

C.1 DVQ Implementation

Base Models and Layer Placement As our vision encoder, we use the fixed resolution variant of SigLIP-2 Tschannen et al. (2025), at 512×512 image resolution (with input images resized to this resolution without cropping), where we use the feature maps from the last layer as well as layer 9 and 18, each having $32 \times 32 = 1024$ patches. Each of these in projected to D_{DVQ} using a 2-layer MLP. Additionally, we create three downsampled feature maps with 16×16, 8×8, and 4×4 patches respectively, by applying convolutional layers with kernel size 3 and stride 2 consecutively to the last feature map. This results in a total of six feature levels, to which we add learned layer embeddings each. As our base LLM we use the instruction-tuned variant of Qwen3-4B Yang et al. (2025). We integrate our DVQ module in every second LLM layer throughout the full LLM, leaving the other layers unchanged.

Reference Point Refinement Reference points $\mathbf{p}^{(i)}$ are handled in logits space, i.e. in $\mathbb{R}^2$ without restrictions. All refinements $\Delta\mathbf{p}^{(i)}$ are also predicted and applied in logits space. Only when used within the deformable attention component and to compute the positional encodings, we convert them into relative image coordinates in $[0, 1]^2$ by applying the sigmoid activation.

Training For standard vision-language modeling, our method does not require any specialized training strategies or loss functions. We follow standard autoregressive training, using the chat template of the LLM, with the loss only applied to assistant message tokens. Before and after the vision tokens, we add special start and end tokens (new tokens in the text vocabulary), similar to how patch tokens are typically marked. The model is then trained in a standard 2-stage approach,

where we first pretrain on image captioning data, before fine-tuning on vision-language instruction data.

C.2 DVQ WITH BOUNDING BOXES

Extension of Reference Points to Reference Boxes We extend reference points to reference boxes by adding two dimensions for width and height, i.e. $\mathbf{p}_s^{(i)} \in [0,1]^4$, where the first two dimensions represent center coordinates of the reference box (corresponding to the reference position) and the last two the size (width and height in relative image coordinates). The deformable attention component uses width and height to normalize the predicted offsets δ. When adding positional encodings, we also compute sinusoidal encodings for width and height, such that the model is aware of the bounding box sizes. Box refinements $\Delta\mathbf{p}^{(i)}$ are also predicted with width and height. Width and height for initial reference boxes $\mathbf{p}^{(0)}$ are set to 0.5, if not otherwise provided as inputs, such that sampling in the first deformable attention layer is barely restricted.

Encoding Bounding Box Inputs When encoding bounding box inputs as vision tokens, we place them inside newly introduced special start and end tokens (new tokens in the text vocabulary) as part of the user message. When encoded as a single reference box, we add a single vision token s with its initial reference box $\mathbf{p}_s^{(0)}$ provided as input. For the corresponding query $\mathbf{q}_s^{(0)}$ we learn another embedding, similar to the initial embeddings of normal vision tokens. If we represent a bounding box as a set of reference points, we compute sets of reference points directly from the given bounding box and represent each of them as its own vision token with the computed initial reference position. We place them all inside the special text tokens to mark them as bounding box inputs. When using 5 reference points, we simply use the center and four corners of the bounding box. When using 25 or 100 points, we pick linearly spaced points alongside the x and y axes of the given box, using 5 or 10 points per axis, respectively. For all settings we still use the normal vision tokens with uniformly sampled reference points. Position refinement is applied to both the normal vision tokens and the vision tokens representing bounding box inputs. Both types of tokens will also have access to vision features, such that vision information can be explicitly sampled at or around the provided input bounding boxes.

Predicting Bounding Box Outputs To predict bounding boxes via vision tokens, we add MenROI tokens to the assistant message. Like other vision tokens, these use the DVQ module and come with associated queries $\mathbf{q}^{(i)}$ and reference boxes $\mathbf{p}^{(i)}$. Similar to bounding box input tokens, we introduce additional query embeddings $\mathbf{q}^{(0)}$ for those tokens. Their initial reference boxes $\mathbf{p}^{(0)}$ are however neither uniformly sampled nor provided as inputs. Instead, we predict them in the first DVQ module, using an MLP directly after downprojection of the LLM features. This allows them to be conditioned on the question text tokens (after the first self-attention layer). The reference boxes of those tokens are then refined using the normal DVQ updates. We then take the reference bounding boxes after the last DVQ layer as predictions. For training, we need to learn where to predict these MenROI tokens in the assistant message. Therefore, we extend the LLM language head to predict these special MenROI tokens. Additionally, we train the predicted bounding boxes using a loss inspired by object detection. We combine the DETR loss (Carion et al., 2020) on the bounding boxes (weighted by coefficient 0.05) with an L2 loss applied on the center coordinates (weighted by coefficient 0.5). Furthermore, we also apply the same loss on the initial reference points of the MenROI tokens, that were predicted in the first DVQ layer. The autoregressive language modeling loss is also kept (weighted by 1.0).

C.3 BASELINES

All baselines are trained with the exact same training protocols and using the same vision encoder and LLM as our DVQ connector. We only tune the learning rates individually, keeping all other hyperparameters the same.

Patch Project and Spatial Aggregation We use the final vision feature map, flatten it, and project it into the LLM's embedding space. For the projection, we use a 2-layer MLP, as commonly used in VLMs, e.g. in LLaVA-1.5 (Liu et al., 2024) or Qwen models (Bai et al., 2025). With the used

SigLIP-2 vision encoder at 512×512 image resolution, this leads to $|\mathcal{V}| = 1024$ vision tokens. For experiments with reduced vision token budgets, we apply spatial aggregation before flattening and projection. Specifically, we use pixel shuffle, i.e. merging neighboring patches along the channel dimension, which is commonly used in Qwen models (Bai et al., 2025). This enables use to reduce the token budgets to $|\mathcal{V}| = 256$ (via merging 2×2 patches), to $|\mathcal{V}| = 64$ (via merging 4×4 patches), to $|\mathcal{V}| = 16$ (via merging 8×8 patches), and to $|\mathcal{V}| = 4$ (via merging 16×16 patches).

DeepStack and Spatial Aggregation For DeepStack (Meng et al., 2024), we closely follow the implementation used in Qwen models (Bai et al., 2025). Specifically, we use the vision feature maps from layer 16 (medium abstraction) and 8 (low abstraction), injecting them into the LLM layers 8 and 16, respectively. The final (high abstraction) vision feature map is used as in Patch Project to form the vision tokens. Here we follow Patch Project as above. Injected DeepStack features are projected using independent 2-layer MLPs per layer. For spatial aggregation, we use pixel shuffle (as above) for the input sequence and average pooling for DeepStack features. For the modified DeepStack variant with 18 injections, we inject vision information at the same LLM layers as in DVQ, i.e. every second layer, while additionally providing it at the embedding level. We use the same feature maps as the original DeepStack connector (and DVQ) but repeat each several times such that up to LLM layer 7 high abstraction features are provided, up to layer 15 medium abstraction features, and then low abstraction features. To further increase the number of parameters, we replace the 2-layer MLP projections by 3-layer MLPs.

For our high-resolution experiments (Tab. 2), we use the low-resolution feature map (produced by encoding the 512×512 resolution image) to form the vision tokens while using the (pooled) high-resolution feature maps (produced by encoding the 1024×1204 resolution image) as DeepStack features.

Perceiver as Query-based Resampling Our Perceiver consists of a simple multi-head cross attention layer with learned query embeddings, applied in vision feature space before projection into the LLM's space. The vision token budget $|\mathcal{V}|$ can be controlled by varying the number of queries, and we train an individual model for each budget. In our small-scale experiments, we integrate the Perceiver during both pretraining and fine-tuning, since fine-tuning alone is extremely short. We also explored a more complex 4-layer transformer decoder variant, fine-tuning schedules with LoRA, or using the Perceiver only during fine-tuning, but observed no notable improvement.

Cross-Attention (DVQ without grounding) This baseline follows exactly the same architecture as our DVQ method, including the injection points, feature maps, and projections, but replaces the deformable attention component by standard multi-head dot product attention. This baseline can also be considered as a variant of the Perceiver that is integrated into the LLM layers. It also serves as an ablation study to demonstrate the relevance of the deformable attention module and thus of the explicit position-grounding.

C.4 Training and Evaluation Details

In all experiments, we use the Adam (Kingma & Ba, 2014) optimizer, without any weight decay, with a cosine annealing schedule and a linear warmup ratio of 3%. Furthermore, we use `bfloat16` precision, clip gradients at norm 1.0 and use gradient accumulation to increase the effective batch size. For evaluation, we rely on the LMMs-Eval (Zhang et al., 2025a) library. All training is done on NVIDIA A100 GPUs, while evaluation is done using A40 GPUs.

Small-Scale Fine-Tuning In our small-scale fine-tuning experiments, we first pretrain for 5 epochs on COCO Captions Chen et al. (2015), where the target caption is randomly sampled from the ground-truth captions. In this stage, only the connector (including any projections), and if needed any newly added special tokens, are trained. The LLM and the vision encoder remain frozen. We use an effective batch size of 64. For DVQ, we use a learning rate of 3e-5, for DeepStack we use 3e-4, and for all other baselines we use 1e-3, each with an effective batch size of 64.

In a second stage, we briefly train individually on each task. On GQA (Hudson & Manning, 2019) and RefCOCOg (Mao et al., 2016), we train for 1 epoch, while on TextVQA (Singh et al., 2019) we train for 2 epochs (as it is very small). Due to the small fine-tuning datasets, we keep the LLM also

frozen during fine-tuning. The projectors remain freely trainable. On all tasks, we use a learning rate of 3e-5 for DVQ, and 1e-4 for all baselines, each with an effective batch size of 64.

This fine-tuning produces task-specific models, which we evaluate individually on their corresponding test sets. We make sure to exclude all test samples (e.g. from RefCOCOg) from both the pretraining and fine-tuning data.

General Purpose Vision-Language Modeling For general purpose vision-language modeling, we exactly follow the LLaVA-1.5 protocol (Liu et al., 2024) and training data. First, we pretrain for 1 epoch on LLaVA Pretrain LCS-558K (Liu et al., 2023), while keeping the LLM (and vision encoder) frozen. For DVQ, we use a learning rate of 3e-5, while for the baselines we use LLaVA's default learning rate of 1e-3, each with an effective batch size of 256. Pretraining is done using a single NVIDIA A100 GPU.

After pretraining, we run visual instruction tuning for 1 epoch on LLaVA-1.5-665K-Instructions (Liu et al., 2024). During this stage, the LLM is unfrozen and freely trainable together with the vision connector, while the vision encoder remains frozen. For DVQ, we use a learning rate of 5e-6, while for the baselines we use LLaVA's default learning rate of 2e-5, each with an effective batch size of 128. We use DeepSpeed (Rasley et al., 2020) at level 2 and train on a node of 4 NVIDIA A100 GPUs.

This instruction tuning produces a single general-purpose model, which we evaluate without further fine-tuning in a zero-shot setting on common VLM benchmarks, namely VQAv2 Goyal et al. (2017), GQA Hudson & Manning (2019), VisWiz Gurari et al. (2018), the image-context questions from ScienceQA Lu et al. (2022), TextVQA Singh et al. (2019) (without providing OCR tokens), the english part of MMBench (MMB) Liu et al. (2025b), MME Fu et al. (2025), SEED Li et al. (2023a), and POPE (adversarial) Li et al. (2023c).