Focusing on What Matters: Object-Agent-centric Tokenization for Vision Language Action Models

Rokas Bendikas[*,†]
Centre for Artificial Intelligence, UCL
r.bendikas@ucl.ac.uk

Daniel Dijkman[*]
Qualcomm AI Research[‡]
ddijkman@qti.qualcomm.com

Markus Peschl
Qualcomm AI Research

Sanjay Haresh
Qualcomm AI Research

Pietro Mazzaglia
Qualcomm AI Research

Abstract:
Vision-Language-Action (VLA) models offer a pivotal approach to learning robotic manipulation at scale by repurposing large pre-trained Vision-Language-Models (VLM) to output robotic actions. However, adapting VLMs for robotic domains comes with a high computational cost, which we attribute to the tokenization scheme of visual inputs. In this work, we aim to enable more efficient VLA training by proposing Oat-VLA, an Object-Agent-centric Tokenization for VLAs. Building on the insights of object-centric representation learning, our method introduces an inductive bias towards scene objects and the agent's own visual information. As a result, we find that Oat-VLA can drastically reduce the number of visual tokens to just a few tokens without sacrificing performance. We reveal that Oat-VLA converges at least twice as fast as OpenVLA on the LIBERO suite, as well as outperforms OpenVLA in diverse real-world pick and place tasks.

Keywords: visual-language-action models, object-centric representations, robotic manipulation, imitation learning

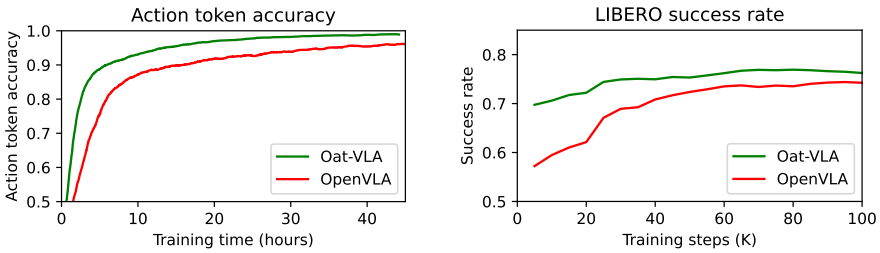


Figure 1: Comparison of our method Oat-VLA to OpenVLA on the LIBERO dataset, full fine-tuning. **Left**: Action token accuracy relative to training time. Oat-VLA converges more than 2× faster. **Right**: Average success rate across the four LIBERO tasks suites relative to training steps.

1 Introduction

Large language models (LLMs) [1, 2] have made it possible to tackle increasingly difficult tasks by leveraging self-supervised learning on large-scale datasets. Furthermore, the combination of the vast amount of knowledge present in modern pre-trained LLMs and the large amount of parameters have opened up many possibilities for adapting LLMs to a variety of applications, such as coding [3], reinforcement learning [4] and imitation learning [5]. A key ingredient thereof lies in the appropriate

[*] equal contribution
[†] work performed during internship at Qualcomm AI Research
[‡] Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

tokenization of the input data, which can be straightforward for natural language [6], but is generally significantly more powerful when leveraging the structure of the input [7]. While the latter has become a common strategy for language-valued inputs, the former choice of a simple tokenizer which is agnostic to input semantics has remained prevalent when training LLMs to additionally process visual information [8, 9].

Multimodal capabilities are the cornerstone of real-world applications including robotics [10]. Recently, Vision-Language-Action models (VLAs) have gained significant traction [11, 5, 10, 12, 13] due to their ability to adapt general knowledge to an imitation-learning setting. VLAs typically take images from cameras in the environment and text, e.g., language instructions, to predict robotic actions, allowing multimodal prompting of agents for robotic manipulation. Despite the success of VLAs, they rely on expensive pre-training on specialized robotics datasets such as Open X-Embodiment [12, 10, 13] to generalize to new tasks. Unfortunately, training VLAs at this scale requires large amounts of compute resources, unavailable to many research laboratories.

We identify the visual processing of current state-of-the-art VLAs [10] as one of the main bottlenecks that significantly increase the compute requirements of training VLAs. This process generally consists of dividing the image into (hundreds of) patches [14], processing them through a visual encoder model, and then feeding the outputs as visual tokens to the LLM. As we show in Figure 1, this visual tokenization scheme, which so far is the most popularly adopted, can be made fundamentally more efficient without compromising performance on robotics tasks. Our key insight is that task execution given a language instruction requires focus on specific parts of the input observation such as objects of interest, while background information is less relevant and can therefore be discounted.

In this work, we present an Object-Agent-centric Tokenization scheme for VLAs (Oat-VLA), which adaptively tokenizes important parts of the scene into a reduced set of visual tokens to feed to the VLA. By producing fewer, but more meaningful tokens, Oat-VLA significantly reduces memory and compute requirements. Furthermore, we designed Oat-VLA's architecture to be able to leverage the pre-trained knowledge already contained in OpenVLA [10] to maintain easy adaptation capabilities and high fine-tuning performance, which match or even outperform heavier VLA training schemes.

Our contributions can be summarized as follows:

- Inspired by object-centric representations, we design a method to condense visual information about objects into a reduced set of tokens;
- We design a simple approach to identify agent-related visual patches and concatenate them with object-centric tokens to ensure precision of the VLA when predicting actions;
- We show that the overall model, Oat-VLA, is modular and scalable, so that it can be efficiently adapted by reusing existing VLA checkpoints. We assess the performance of our method on the LIBERO benchmark and demonstrate a 2x speedup in terms of convergence according to the action token accuracy and success rate on tasks.
- Finally, we tested Oat-VLA on real-world pick and place tasks and find that it executes more robustly than OpenVLA, translating into a higher success rate.

2 Related work

Vision-language-action models. There has been an increasing trend in leveraging generalization capabilities of Vision-Language-Models (VLMs) for robotic control by fine-tuning these VLMs on robotics tasks [15, 10, 16]. RT-2 [15] utilizes frontier closed-sourced VLMs by co-fine-tuning them on visual and robotics tasks to achieve a general purpose robotics policy. At the same time, OpenVLA [10] and $\pi 0$ [16] embody similar open-source efforts. Preceding these, there have been multiple efforts at using the transformer architecture trained on large robotics dataset to achieve general purpose robotics policies. Gato [17] presented the first transformer trained on a multitude of language, vision and robotics tasks whereas RT-1 [11] presented a first concentrated effort at using large transformer architectures trained on a large robotics dataset. Octo [13] and JAT [18] present open-source transformer policies designed for efficient fine-tuning on down-stream tasks. Consequently, there have been multiple efforts at incorporating spatial awareness [19], 3D infor-

mation [20], generalization to new objects [21], recurrence and streaming ability [22, 23, 24, 25] and embodied chain-of-thought [26, 27] to VLAs to improve generalization. In our work, we focus on improving the visual representation of VLAs for compute efficiency reasons, while orthogonal works [28, 29] have shown significant progress by improving the action prediction component.

Efficient VLMs, VLAs. Concurrent to improving generalization capability of VLAs, a number of works have explored improving efficiency of these models and the VLMs on which they build. Related work TokenLearner [30] introduces a generic learned method to reduce the number of tokens, while other works [31, 32] increase the efficiency of the attention layers in transformers. TinyVLA [33] presents a VLA with a smaller backbone and diffusion-based action heads for faster training and inference speeds. Similarly, MiniVLA [34] trains a performant VLA using a 1B parameters models. VLA-Cache [35] proposes a caching mechanism that adaptively identifies visual tokens with minimal changes and re-uses computation for "unchanged" visual tokens for efficient processing. Deer-VLA [36] on the other hand, proposes dynamic inference that allows automatically adjusting the size of the model leading to performance improvements in inference. Previously, improving VLA tokenization has been studied by [37], which we consider orthogonal to our work as we aim to improve the visual backbone as opposed to the action tokenization.

Object-centric representations for robotics. Object-centric representation learning focuses on modeling individual objects within a scene to improve understanding of complex environments. Aside from reinforcement learning [38, 39, 40, 41], it has gained traction in robotics and imitation learning: A3VLM [42] leverages an object-centric approach to train robot agnostic VLAs, while DexGraspVLA [43] utilizes SAM [44] to additionally feed object-mask features into a VLA. Similarly, Sam2Act [45] uses SAM as an image encoder to learn multi-view policies, FOCUS [38] obtains mask supervision from SAM and POCR [46] proposes a general object-centric framework for pre-trained robotic embeddings. Concurrent with our work, ControlManip [47] conditions a pre-trained VLA on an object-centric representation. For our work, we employ FT-Dinosaur [48], which employs an unsupervised object-centric approach to adapt vision encoders for object discovery.

3 Method

VLAs learn a distribution $p(\mathbf{a}|\mathbf{o}, \ell)$ over actions $\mathbf{a}$, given an observation $\mathbf{o}$ and a task description ℓ. In our case, we assume that $\mathbf{o} \in \mathbb{R}^{C \times W \times H}$ consists of a single camera image. Observations are encoded using a visual encoder VisEnc, such as DinoV2+SigLIP [49, 50], resulting in a set of visual tokens $\mathbf{v}_{1 \ldots K} = \text{VisEnc}(\mathbf{o})$, whereas language instructions are encoded using the tokenizer of the underlying LLM, which we denote as $\mathbf{l}_{1 \ldots J}$. Overall, we have $p(\mathbf{a}|\mathbf{o}, \ell) = \text{LLM}(\mathbf{a}|\mathbf{l}_1, \ldots, \mathbf{l}_J, \mathbf{v}_1, \ldots, \mathbf{v}_K)$, where the distribution over actions is chosen to be discrete by using a binning scheme [10].

The most common visual tokenization strategy consists of dividing the observations into patches, processing them with a visual encoder, and feeding them to the LLM. For example, a 224×224 pixel image can be divided into 256 patches of size 14×14 pixels. This would result in the LLM processing $K = 256$ visual tokens, an amount of tokens that is generally one order of magnitude larger than the amount of tokens processed for a language instruction. As a consequence, the amount of visual tokens is one of the main bottlenecks when training VLAs. The aim of our work is to greatly reduce the number K of visual tokens with no loss of performance.

3.1 Object-centric tokens

In order to understand the scene, the agent needs to be able to look at all parts of the observed images. While feeding embedded image patches to the VLA's LLM is a safe design choice, it is inefficient from an information perspective. Several patches in an image often contain information that is irrelevant for the agent, e.g., background information. Furthermore, many patches in an image can represent the same entities, e.g., an object spanning across multiple patches. Such information could be efficiently compressed into a smaller amounts of tokens, which we call *object-centric tokens*.

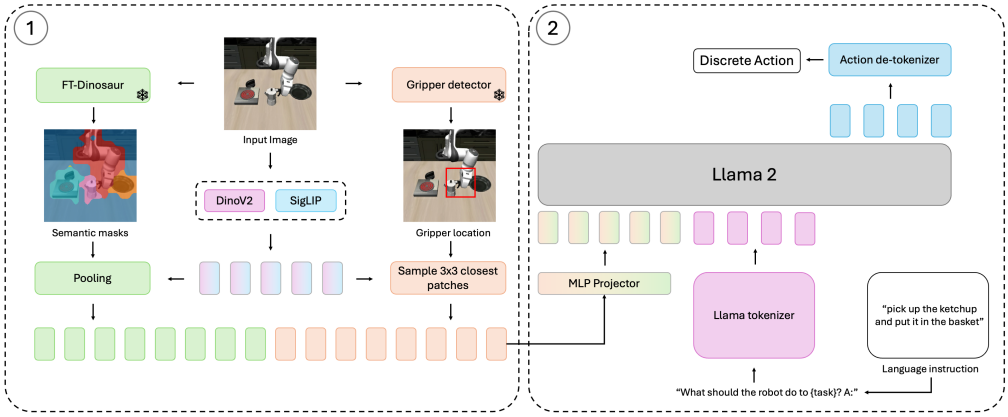


Figure 2: Oat-VLA introduces a visual tokenization process, which extracts object-centric and agent-centric tokens. These tokens are then fed to the LLM for action prediction.

Object-centric models that perform semantic segmentation [44, 48] allow to group pixels in an image, based on their common semantical meaning. Each group of pixels is generally assigned an index and can be referred to as an "object mask". Given the observation $\mathbf{o}$, we process the image with an object extractor, to obtain N segmentation masks, i.e. $\mathbf{m}_{1\ldots N} = \text{ObjEnc}(\mathbf{o})$, where N can be a fixed or a variable number.

In order to obtain *object-centric tokens* for our VLA, we perform the following operations:

1. we extract *visual tokens* for all patches in an image, using the visual encoder;
2. we obtain the *object masks* using an object-centric model at the same output resolution as the patches, and gather the visual tokens based on their corresponding masks;
3. we perform a *pooling* operation to compress the information of the tokens that belong to the same object mask.

Formally, we compute object-centric tokens as

$$\mathbf{t}_j = \text{pool}(\{\mathbf{t}_n^k \mid k \in \{1\ldots K\}, n \in \{1\ldots N\}\}),$$
$$\mathbf{t}_n^k = \mathbf{m}_n^k \odot \mathbf{v}_k,$$

where $\mathbf{m}_n^k$ is the mask corresponding to object n indexed by the patch coordinates of $\mathbf{v}_k$. This procedure allows us to reduce the number of visual tokens processed by 1 or 2 orders of magnitude, depending on the masking strategy adopted. In our experiments, we adopt FT-Dinosaur [48], an unsupervised object-centric model. FT-Dinosaur has two features that make it a good fit for our approach: (i) it can be flexibly fine-tuned on new datasets in an unsupervised fashion, and (ii) it allows to define a fixed number of masks to be extracted from an image. For the pooling part, we adopt a simple average pooling and ablate this choice in Section 4.3. For our main experiments, we use only 7 object masks, consequently obtaining 7 object-centric tokens.

3.2 Agent-centric tokens

While object-centric tokens summarize *what* is present and *where*, they inherently discard high-frequency details. For fine-grained manipulation, it is crucial that the agent possesses accurate information about the end effectors and their interactions with objects. During interaction with objects, the objects and the end effectors' information may be aggregated into different object-centric tokens. One of the consequences is that information about the interaction, e.g., touching an object, may be incidentally discarded. To avoid such possible scenarios, and always provide the agent with high-resolution information about the end-effectors, we introduce *agent-centric tokens*.

In a single-arm robotic manipulation system, the end effector is generally a gripper attached to the arm. Identifying the gripper in the camera observations enables to locate where the agent's actions

are being executed. In systems with camera calibration, the gripper location can be obtained from the robot pose. However, such calibration is often unavailable, e.g., it is missing for several datasets in the Open X-Embodiment dataset [12]. To obtain a generic solution, we train a gripper detector model. At test time, it returns a single 2-D keypoint, locating the gripper in the visual observation.

In order to obtain *agent-centric tokens* for our VLA, we perform the following operations:

1. (shared step with object-centric tokens) we extract *visual tokens* for all patches in an image, using the visual encoder;
2. we detect the *gripper location* in the camera observation, using the gripper detector;
3. we obtain a grid of *patches around the agent*, by selecting the patches around the gripper.

This procedure allows us to select an arbitrary number of visual tokens around the agent's end effector location, which ensures the agent sees the effects of its actions at all times. In our experiments, we adopt a gripper detector based on a lightweight ResNet-based [51] Faster R-CNN [52] architecture. The gripper training is supervised with a few thousand manually annotated frames from Open X-Embodiment plus automatically extracted gripper locations for the LIBERO dataset. For the agent-centric tokens, we adopt a 3×3 patches grid, which results in a set of 9 agent-centric tokens. In the very rare cases where no gripper is detected, the agent-centric tokens take the value of all patch features over the entire image.

3.3 Oat-VLA: Object-Action-centric Tokenization for VLAs

Our method, Oat-VLA, aims to provide a more efficient VLA training strategy, by employing a reduced number of object-centric and agent-centric tokens. In Oat-VLA, object-centric tokens enable semantically-organized perception of the scene, while agent-centric tokens enable precise manipulation. Oat-VLA's visual tokenization process is illustrated in Figure 2.

The visual tokenization is the main innovation of our approach. It consists of the object-centric tokens extraction and the agent-centric tokens extraction procedures, as explained in this section. The resulting tokens are passed through an MLP projector and fed to the (Llama 2) LLM backbone, following the OpenVLA's architecture [10]. Crucially, Oat-VLA does not disruptively change the information contained in the tokens fed to the LLM, as it only performs operations of aggregation and selection of the tokens from the pre-trained visual encoder. Thus, as we also show in Section 4, the agent is still able to leverage the pre-trained knowledge coming from the base VLM/VLA.

For 224x224 images, the number of visual tokens used by Oat-VLA is 16 in total: 7 object-centric tokens, and 9 agent-centric tokens. For the same image resolution, the original OpenVLA's visual backbones extracts 256 visual tokens. Thus, Oat-VLA uses **93.75% less visual tokens**. This reduction allows to increase the batch size significantly during training and, as we observe in Figure 1, it allows to train our VLAs to achieve slightly higher performance than OpenVLA, and **more than** $2\times$ **faster** (both in terms of training steps and actual time).

4 Experimental results

In Figure 1, we show that Oat-VLA can be trained significantly faster than its "original" counterpart, the OpenVLA model. The goal of this section is to demonstrate in detail that Oat-VLA's performance is consistently in line or superior to OpenVLA, across different training settings.

In terms of environments, we adopt two settings: the LIBERO environment, with its four task suites, and a real-world environment, using a UFACTORY xArm 6 to rearrange objects on a tabletop.

For OpenVLA experiments below, we start training our models from the the OpenVLA checkpoint [53] pre-trained on the Open X-Embodiment dataset (which excludes the LIBERO dataset). For Oat-VLA experiments, we start from the OpenVLA checkpoint that has been fined-tuned using the Oat-VLA architecture on a subset Open X-Embodiment for 200K steps, see Appendix 7.2 for details. We leave training Oat-VLA from a stock VLM checkpoint such as Prismatic VLMs [54] or PaliGemma [55] for future work.

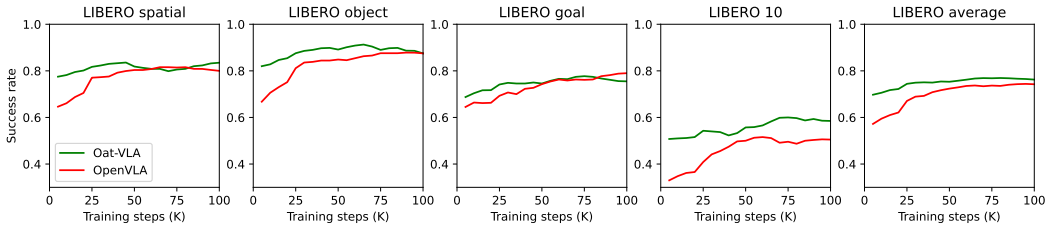


Figure 3: Evaluations on LIBERO, every 5K training (full fine-tuning) steps. The plots are mean filtered using one evaluation before and after.

4.1 Full Fine-tuning

We want to show that Oat-VLA enables a more efficient fine-tuning of the all the weights of the VLA model on new datasets. For this purpose, starting from the corresponding base VLA checkpoints, we fine-tune both Oat-VLA and OpenVLA on the full LIBERO dataset (SPATIAL, OBJECT, GOAL, 10 and 90). We use a single action space normalization for all five subsets.

Full fine-tuning of Oat-VLA is performed using a batch size of $8 \times 64 = 512$ which we measured to process 320 examples/second on average on an 8xH100 node. Full fine-tuning of OpenVLA is performed using a batch size of $8 \times 32 = 256$, processing 157 examples/second on average on an 8xH100 node. The larger batch-size of Oat-VLA is in inherent advantage of the method, because fewer visual tokens result in lower GPU memory requirements per sample. Any other training settings are chosen to remain the same as in OpenVLA.

The models are evaluated on the four LIBERO task suites: SPATIAL tests the agent's understanding of spatial relationships, OBJECT tests the agent's understanding of object types, GOAL tests the agent's knowledge of different task-oriented behaviors and LONG, also referred to as 10 consists of a variety of long-horizon tasks with diverse object interactions and versatile motor skills. For each task suite, we run 100 evaluations.

We used the full fine-tuning to study how the action token accuracy evolves over time. This is represented in Figure 1. While training the models, we also evaluate checkpoints every 5000 training steps, to benchmark the agent's performance in terms of training efficiency. We report the performance of the two models over training steps in Figure 3. Overall, Oat-VLA performs slightly better than OpenVLA, with a more significant performance advantage in the challenging LIBERO 10 suite. Importantly, Oat-VLA obtains higher success rates more than $2\times$ faster, thanks to the higher batch size and improved visual encoding, which simplifies training by discarding or compressing less relevant information.

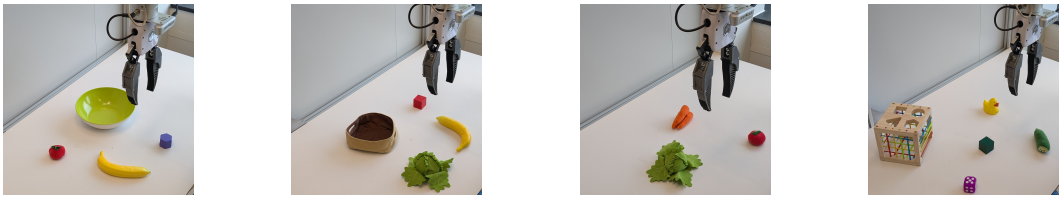
4.2 LoRA Fine-tuning

It is desirable that a VLA can be fine-tuned using parameter-efficient techniques, such as LoRA [56]. Thus, we compare Oat-VLA with OpenVLA, when it comes to LoRA fine-tuning of the weights. These results allow for a direct comparison with the LoRA results in [10].

Table 1: LoRA fine-tuning success rate comparison on LIBERO. The numbers for OpenVLA, Octo and Diffusion Policy are taken from OpenVLA [10].

	Spatial	Object	Goal	10	Average
Oat-VLA	**87.3** $\pm 0.7\%$	89.1 $\pm 0.2\%$	82.1 $\pm 0.8\%$	**55.9** $\pm 1.8\%$	**78.6** $\pm 0.5\%$
OpenVLA	84.7 $\pm 0.9\%$	88.4 $\pm 0.8\%$	79.2 $\pm 1.0\%$	53.7 $\pm 1.3\%$	76.5 $\pm 0.6\%$
Octo	78.9 $\pm 1.0\%$	85.7 $\pm 0.9\%$	**84.6** $\pm 0.9\%$	51.1 $\pm 1.3\%$	75.1 $\pm 0.6\%$
Diffusion Policy	78.3 $\pm 1.1\%$	**92.5** $\pm 0.7\%$	68.3 $\pm 1.2\%$	50.5 $\pm 1.3\%$	72.4 $\pm 0.7\%$

Figure 4: **Top.** The setup for some of the real-world tasks. (a) banana in green bowl (b) red cube in brown bag (c) zucchini in front of green cube (d) tomato left of lettuce. **Bottom.** The table reports the success rates on the real-world tasks and number of successful trials.



(a) (b) (c) (d)

	OpenVLA	**Oat-VLA**
In-distribution tasks	**52**% (13/25)	**72**% (18/25)
Place the banana in the green bowl	70% (7/10)	70% (7/10)
Place the red cube in the brown bag	50% (3/6)	33% (2/6)
Place the tomato left of the lettuce	60% (3/5)	100% (5/5)
Place the zucchini in front of the green cube	0% (0/4)	100% (4/4)
Out-of-distribution tasks	**29**% (7/24)	**46**% (11/24)
Place the rubber duck in the green bowl	40% (4/10)	30% (3/10)
Place the mushroom in the brown bag	0% (0/6)	50% (3/6)
Place the purple die left of the lettuce	0% (0/4)	50% (2/4)
Place the zucchini in front of the red hexagon	75% (3/4)	75% (3/4)
Overall	**41**% (20/49)	**59**% (29/49)

LoRA fine-tuning is performed using a batch size of $8 \times 48 = 384$ for Oat-VLA and $8 \times 16 = 128$ on an 8xH100 node. Despite the lower number of learnable parameters, the batch-size is smaller than during full fine-tuning because FSDP [57] is not enabled during LoRA fine-tuning in OpenVLA. Other training settings are the same as in OpenVLA. At the given settings, Oat-VLA processes 384 examples/second on an H100 node, while OpenVLA processes 197 examples/second.

LIBERO benchmark. Following OpenVLA [10], evaluations are performed using three seeds with 500 roll-outs per task per seed and we report average success rates and standard deviation. Our results are in Table 1, where we also compare to the results of Octo [13] and Diffusion Policy [58]. Similarly to the full fine-tuning results, the performance of Oat-VLA is slightly higher than OpenVLA, in all the task suites, increasing the overall advantage over the other methods. We note that Oat-VLA achieves good performance at 30K training steps (see Appendix 7.4), while the released OpenVLA checkpoints [53] are around 60K steps. Both Oat-VLA and OpenVLA have lower performance than Octo in Goal and than Diffusion Policy in Object. This may be due to the diffusion process adopted in these methods, which provides more accurate action prediction.

Real-world benchmark. We collect a dataset of 320 trajectories on a robotic setup using a UFACTORY xArm 6 operating on a white tabletop. The setup is observed by the agent through the lenses of a RealSense D435 camera positioned in a corner of the table, using only the RGB image.

We evaluate on two sets of tasks: an in-distribution set and an out-of-distribution set. For the in-distribution tasks, at least 10 demos of such tasks are present in the dataset. For out-of-distribution tasks, we take the in-distribution tasks and swap some of the objects, e.g., the object to pick or place next to, so that the agent never saw such tasks in the dataset. For all the evaluations, we consistently re-position the objects with slight variations, e.g., different orientations or positions of an object. Further details are provided in Appendix 7.6.

Visualizations of some of tasks and the results are shown in Figure 4. Oat-VLA reports a consistent advantage in performance over OpenVLA both on in-distribution and out-of-distribution tasks. From qualitative assessment of the robot executions, we observed that Oat-VLA tends to be more precise

at picking and placing objects. Instead, OpenVLA would sometimes grasp in the air above the target object or place the object on the wrong side/on top of the target location.

4.3 Ablations

With our ablation study, we analyze different ways of condensing the information in the visual tokens and verify some of our design choices. We adopt the full fine-tuning setup on the LIBERO benchmark and train all models for 20K training steps. The ablations we analyze are as follows:

- **Single image token**: condensing information from all visual tokens into a single token, using attention pooling [59];
- **Object-centric tokens only**: extracting the object tokens, using the method described in Section 3.1. Using attention pooling to condense information;
- **Oat-VLA (attention pooling)**: using both object-centric tokens and agent-centric tokens, where the object-centric pooling is an attention pooling layer;
- **Oat-VLA (average pooling)**: object-centric tokens and agent-centric tokens, where object-centric pooling is an average pooling layer (default method for all other experiments).

The results are presented in Table 2. We observe that condensing information into multiple tokens for the objects, rather than into a single token, improves performance slightly, especially in LIBERO Spatial. The agent-centric tokens have a major positive effect on performance and, as we hypothesized, they are crucial to not incur any loss in performance. Finally, we found average pooling to work better than attention pooling, possibly due to its simplicity, i.e. no additional layers. We believe that other pooling strategies could be used to better condense the object token information, but we leave such direction for future work.

Table 2: Ablation experiments for design choices in Oat-VLA on LIBERO

	Spatial	Object	Goal	10	Average
Single image token	64.0%	74.0%	71.0%	31.0%	60.0%
Object-centric tokens only	75.0%	74.0%	67.0%	29.0%	61.3%
Oat-VLA (attention pool)	81.0%	86.0%	73.0%	41.0%	70.3%
Oat-VLA (average pool)	**86.5%**	**89.1%**	**80.3%**	**52.5%**	**77.1%**

5 Discussion and conclusion

Training VLAs at scale is an important problem as we progress towards multimodal robotics systems that handle both visual and language inputs. However, training VLAs comes with several challenges, particularly in terms of large compute resource requirements and long training times. With Oat-VLA, we propose a way to drastically reduce the training cost and time of VLAs by reworking the visual tokenization process using object-centric and agent-centric tokens.

Our approach implicitly introduces some inductive bias into the system. For the agent-centric tokens, we used the end-effector as a straightforward proxy for "the location where the agent interacts with the world". This should be generalized to allow multiple manipulators and non-prehensile manipulation with other parts of the robot. For the object-centric tokens, we employ object-centric models that perform semantic segmentation. Future work should aim to find object-agent-centric-like inductive biases that arise more naturally from the data.

In conclusion, we hope to inspire more work in the area of efficient visual tokenization for VLAs and other end-to-end trained robot policies. Additionally, we hope that our work will enable more research labs to work on the training of foundational VLA models.

6 Limitations

We have only tested Oat-VLA on a single-armed robot for pick and place tasks. It remains to be seen how Oat-VLA performs in a bi-manual setting, and on more complex tasks such as folding clothes.

Our prototype consists of three distinct models: FT-Dinosaur for the object-centric masks, Faster R-CNN for the gripper detector and OpenVLA as the VLA. It makes sense to let all parts share the same visual backbone. We do not expect any changes in success rates, but training time should improve in the order of 5% to 10%.

As shown in Appendix 7.7, using A100 or RTX A5000 GPUs, Oat-VLA offers no advantage in batch-of-1 inference speed, because in our analysis the inference time is dominated by loading the LLM weights from GPU RAM to GPU cache. This might change with different hardware architectures, or different action heads where the batch-size during roll-out increases due to parallel inference, such as for example in OpenVLA-OFT [28].

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[3] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[4] M. Klissarov, P. D'Oro, S. Sodhani, R. Raileanu, P.-L. Bacon, P. Vincent, A. Zhang, and M. Henaff. Motif: Intrinsic motivation from artificial intelligence feedback, 2023. URL https://arxiv.org/abs/2310.00166.

[5] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, P. Florence, C. Fu, M. G. Arenas, K. Gopalakrishnan, K. Han, K. Hausman, A. Herzog, J. Hsu, B. Ichter, A. Irpan, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal,

L. Lee, T.-W. E. Lee, S. Levine, Y. Lu, H. Michalewski, I. Mordatch, K. Pertsch, K. Rao, K. Reymann, M. Ryoo, G. Salazar, P. Sanketi, P. Sermanet, J. Singh, A. Singh, R. Soricut, H. Tran, V. Vanhoucke, Q. Vuong, A. Wahid, S. Welker, P. Wohlhart, J. Wu, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

[6] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[7] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[8] L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, M. Neumann, I. Alabdulmohsin, M. Tschannen, E. Bugliarello, T. Unterthiner, D. Keysers, S. Koppula, F. Liu, A. Grycner, A. Gritsenko, N. Houlsby, M. Kumar, K. Rong, J. Eisenschlos, R. Kabra, M. Bauer, M. Bošnjak, X. Chen, M. Minderer, P. Voigtlaender, I. Bica, I. Balazevic, J. Puigcerver, P. Papalampidi, O. Henaff, X. Xiong, R. Soricut, J. Harmsen, and X. Zhai. Paligemma: A versatile 3b vlm for transfer, 2024. URL https://arxiv.org/abs/2407.07726.

[9] S. Tong, E. Brown, P. Wu, S. Woo, M. Middepogu, S. C. Akula, J. Yang, S. Yang, A. Iyer, X. Pan, Z. Wang, R. Fergus, Y. LeCun, and S. Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal llms, 2024. URL https://arxiv.org/abs/2406.16860.

[10] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[11] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[12] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[13] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

[15] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[16] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control, 2024. *arXiv preprint arXiv:2410.24164*, 2025.

[17] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[18] Q. Gallouédec, E. Beeching, C. Romac, and E. Dellandréa. Jack of all trades, master of some, a multi-purpose transformer agent. *arXiv preprint arXiv:2402.09844*, 2024.

[19] D. Qu, H. Song, Q. Chen, Y. Yao, X. Ye, Y. Ding, Z. Wang, J. Gu, B. Zhao, D. Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

[20] H. Zhen, X. Qiu, P. Chen, J. Yang, X. Yan, Y. Du, Y. Hong, and C. Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.

[21] M. Zhu, Y. Zhu, J. Li, Z. Zhou, J. Wen, X. Liu, C. Shen, Y. Peng, and F. Feng. Objectvla: End-to-end open-world object manipulation without demonstration. *arXiv preprint arXiv:2502.19250*, 2025.

[22] X. Li, M. Liu, H. Zhang, C. Yu, J. Xu, H. Wu, C. Cheang, Y. Jing, W. Zhang, H. Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023.

[23] J. Liu, M. Liu, Z. Wang, P. An, X. Li, K. Zhou, S. Yang, R. Zhang, Y. Guo, and S. Zhang. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation. *Advances in Neural Information Processing Systems*, 37:40085–40110, 2024.

[24] T. Schmied, T. Adler, V. Patil, M. Beck, K. Pöppel, J. Brandstetter, G. Klambauer, R. Pascanu, and S. Hochreiter. A large recurrent action model: xlstm enables fast inference for robotics tasks. *arXiv preprint arXiv:2410.22391*, 2024.

[25] S. Haresh, D. Dijkman, A. Bhattacharyya, and R. Memisevic. Clevrskills: Compositional language and visual reasoning in robotics. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[26] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine. Robotic control via embodied chain-of-thought reasoning. In *Conference on Robot Learning*, 2024.

[27] X. Li, C. Mata, J. Park, K. Kahatapitiya, Y. S. Jang, J. Shang, K. Ranasinghe, R. Burgert, M. Cai, Y. J. Lee, and M. S. Ryoo. Llara: Supercharging robot learning data for vision-language policy. In *International Conference on Learning Representations*, 2025.

[28] M. J. Kim, C. Finn, and P. Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[29] M. Reuss, J. Pari, P. Agrawal, and R. Lioutikov. Efficient diffusion transformer policies with mixture of expert denoisers for multitask learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=nDmwloEl3N.

[30] M. S. Ryoo, A. Piergiovanni, A. Arnab, M. Dehghani, and A. Angelova. Tokenlearner: Adaptive space-time tokenization for videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[31] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[32] I. Leal, K. Choromanski, D. Jain, A. Dubey, J. Varley, M. Ryoo, Y. Lu, F. Liu, V. Sindhwani, Q. Vuong, et al. Sara-rt: Scaling up robotics transformers with self-adaptive robust attention. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6920–6927. IEEE, 2024.

[33] J. Wen, Y. Zhu, J. Li, M. Zhu, Z. Tang, K. Wu, Z. Xu, N. Liu, R. Cheng, C. Shen, Y. Peng, F. Feng, and J. Tang. Tinyvla: Toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025. doi:10.1109/LRA.2025.3544909.

[34] S. Belkhale and D. Sadigh. Minivla: A better vla with a smaller footprint. *URL https://github.com/Stanford-ILIAD/openvla-mini*, 2024.

[35] S. Xu, Y. Wang, C. Xia, D. Zhu, T. Huang, and C. Xu. Vla-cache: Towards efficient vision-language-action model via adaptive token caching in robotic manipulation. *arXiv preprint arXiv:2502.02175*, 2025.

[36] Y. Yue, Y. Wang, B. Kang, Y. Han, S. Wang, S. Song, J. Feng, and G. Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

[37] K. Pertsch, K. Stachowicz, B. Ichter, D. Driess, S. Nair, Q. Vuong, O. Mees, C. Finn, and S. Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[38] S. Ferraro, P. Mazzaglia, T. Verbelen, and B. Dhoedt. Focus: Object-centric world models for robotics manipulation. *arXiv preprint arXiv:2307.02427*, 2023.

[39] C. Sancaktar, S. Blaes, and G. Martius. Curious exploration via structured world models yields zero-shot object manipulation. *Advances in Neural Information Processing Systems*, 35:24170–24183, 2022.

[40] J. Yoon, Y.-F. Wu, H. Bae, and S. Ahn. An investigation into pre-training object-centric representations for reinforcement learning. *arXiv preprint arXiv:2302.04419*, 2023. URL https://arxiv.org/abs/2302.04419.

[41] D. Haramati, T. Daniel, and A. Tamar. Entity-centric reinforcement learning for object manipulation from pixels. *arXiv preprint arXiv:2404.01220*, 2024.

[42] S. Huang, H. Chang, Y. Liu, Y. Zhu, H. Dong, P. Gao, A. Boularias, and H. Li. A3vlm: Actionable articulation-aware vision language model. *arXiv preprint arXiv:2406.07549*, 2024.

[43] Y. Zhong, X. Huang, R. Li, C. Zhang, Y. Liang, Y. Yang, and Y. Chen. Dexgraspvla: A vision-language-action framework towards general dexterous grasping, 2025. URL https://arxiv.org/abs/2502.20900.

[44] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, et al. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[45] H. Fang, M. Grotz, W. Pumacay, Y. R. Wang, D. Fox, R. Krishna, and J. Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

[46] J. Shi, J. Qian, Y. J. Ma, and D. Jayaraman. Composing pre-trained object-centric representations for robotics from "what" and "where" foundation models, 2024. URL https://arxiv.org/abs/2404.13474.

[47] P. Li, Y. Wu, W. Li, Y. Huang, Z. Zhang, Y. Chen, S.-C. Zhu, T. Liu, and S. Huang. Controlmanip: Few-shot manipulation fine-tuning via object-centric conditional control. In *ICLR 2025 Robot Learning Workshop: Towards Robots with Human-Level Abilities*, 2025.

[48] A. Didolkar, A. Zadaianchuk, A. Goyal, M. Mozer, Y. Bengio, G. Martius, and M. Seitzer. Zero-shot object-centric representation learning. *arXiv preprint arXiv:2408.09162*, 2024.

[49] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

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

[51] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[52] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[53] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. openvla-7b-prismatic. *Hugging Face*, 2024. URL https://huggingface.co/openvla.

[54] S. Karamcheti, S. Nair, A. Balakrishna, P. Liang, T. Kollar, and D. Sadigh. Prismatic vlms: Investigating the design space of visually-conditioned language models, 2024. URL https://arxiv.org/abs/2402.07865.

[55] L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, M. Neumann, I. Alabdulmohsin, M. Tschannen, E. Bugliarello, T. Unterthiner, D. Keysers, S. Koppula, F. Liu, A. Grycner, A. Gritsenko, N. Houlsby, M. Kumar, K. Rong, J. Eisenschlos, R. Kabra, M. Bauer, M. Bošnjak, X. Chen, M. Minderer, P. Voigtlaender, I. Bica, I. Balazevic, J. Puigcerver, P. Papalampidi, O. Henaff, X. Xiong, R. Soricut, J. Harmsen, and X. Zhai. Paligemma: A versatile 3b vlm for transfer, 2024. URL https://arxiv.org/abs/2407.07726.

[56] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[57] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

[58] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[59] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

[60] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. *Advances in neural information processing systems*, 33:11525–11538, 2020.

[61] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[62] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Supplementary Material

7.1 Gripper detector

As mentioned in Section 3.2, Oat-VLA requires a gripper detector to identify the agent-centric tokens. We manually annotated the gripper location for 2000 images for three Open X-Embodiment subsets (Bridge, Fractal, FMB), and another 2000 annotated images from our own real-world dataset (see Section 4.2) and added in 50K LIBERO images, with known gripper locations from the simulator. The annotations indicate the end-effector position, i.e. the point midway between the two gripper-fingers. We then fine-tune a ResNet R-CNN [52] to detect a fictional 64×32 pixel bounding box around the end-effector position. The IoU of the resulting detector is $> 80\%$ on the holdout set.

7.2 Pre-training Oat-VLA on Open X-Embodiment details

The fine-tuning experiments in Section 4 are initialized from an Oat-VLA checkpoint that has been pre-trained on a subset of Open X-Embodiment as follows: starting from the weights of the pre-trained OpenVLA checkpoint [53] (LLama-2 7B, DINOv2 ViT-L/14 and SigLIP ViT-So400M/14, resolution 224×224), we fine-tune Oat-VLA to around 95% action token accuracy on the Bridge+FMB+Fractal subset of Open X-Embodiment for 235K steps.

Figure 5 shows the training action token accuracy as a function of training steps. Additionally, it shows a second experiment where we fine-tune on only the Bridge dataset. We note that Oat-VLA gets to high accuracy in 30K steps for Bridge, but the combined Bridge+FMB+Fractal dataset takes longer to reach 95% action token accuracy mark, i.e. about 10% of the compute budget used to train the original OpenVLA checkpoint [10].

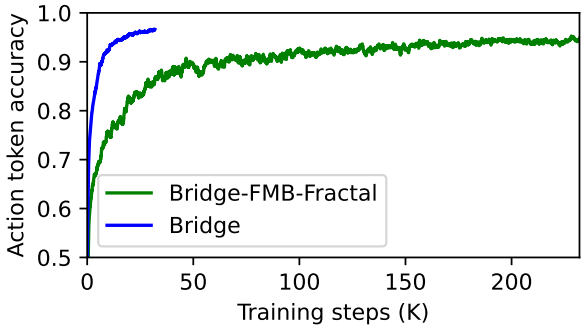


Figure 5: Training action token accuracy while fine-tuning Oat-VLA on Open X-Embodiment subsets, starting from the OpenVLA checkpoint. The plots are mean filtered using 250 training steps before and after.

7.3 Detailed LoRA fine-tuning results

Table 1 in Section 4.2 reports the highest success rate of Oat-VLA across several checkpoints (i.e. 20K, 30K, 40K, 50K training steps). This reporting style follows OpenVLA [53] which hand-picked the checkpoints for LIBERO Spatial (50K training steps), Object (50K training steps), Goal (60K training steps), and 10 (80K training steps). For transparency, Table 3 shows the success rate of Oat-VLA for each individual checkpoint. We note that Oat-VLA reaches very good performance across all LIBERO task suites at 30K training steps. In Table 4, we additionally show results for LoRA fine-tuning with a batch size of 8×32.

7.4 Additional full fine-tuning results

To determine if the higher performance of Oat-VLA is due to simply due to the larger batch sizes that it enables on the same hardware, or perhaps due to some other effect such as inductive bias, we

Table 3: **LoRA fine-tuning (8×48 batch size)** success rates evaluated for different training steps. 500 evaluations per LIBERO task suite, 3 seeds. OpenVLA results copied from [10].

	Oat-VLA				OpenVLA
LIBERO task suite	20K steps	30K steps	40K steps	50K steps	
spatial	86.5 ±1.4%	**87.3** ±0.7%	85.8 ±0.9%	85.3 ±0.6%	84.7 ±0.9%
object	**89.1** ±0.2%	88.2 ±0.6%	87.2 ±2.0%	88.1 ±0.7%	88.4 ±0.8%
goal	80.3 ±1.1%	81.7 ±0.9%	80.3 ±0.4%	**82.1** ±0.8%	79.2 ±1.0%
10	52.5 ±0.5%	**55.9** ±1.8%	52.7 ±1.8%	54.7 ±1.0%	53.7 ±1.3%
average	77.1 ±0.2%	**78.3** ±0.3%	76.5 ±0.9%	77.6 ±0.3%	76.5 ±0.6%

Table 4: **LoRA fine-tuning (8×32 batch size)** success rates evaluated for different training steps. 500 evaluations per LIBERO task suite, 3 seeds. OpenVLA results copied from [10].

	Oat-VLA					OpenVLA
LIBERO task	40K steps	50K steps	60K steps	70K steps	80K steps	
spatial	84.9 ±0.7%	**85.7** ±1.5%	85.5 ±1.0%	85.7 ±1.2%	84.3 ±0.7%	84.7 ±0.9%
object	86.9 ±1.2%	87.3 ±0.9%	87.6 ±0.8%	**88.7** ±0.9%	87.3 ±0.8%	88.4 ±0.8%
goal	79.9 ±1.6%	**80.2** ±1.4%	79.1 ±1.3%	79.9 ±0.6%	79.5 ±0.7%	79.2 ±1.0%
10	55.5 ±1.9%	54.2 ±0.6%	55.2 ±1.0%	54.5 ±0.7%	**55.9** ±1.5%	53.7 ±1.3%
average	76.8 ±0.7%	76.9 ±1.0%	76.9 ±0.4%	**77.2** ±0.8%	76.7 ±0.7%	76.5 ±0.6%

performed an additional Oat-VLA full fine-tuning experiment with a training batch size of 8×32 (matching the default batch size of OpenVLA) instead of 8×64 (as used in Section 4.1).

Figure 6 shows the results. We plot success rates on LIBERO, both relative to the number of training steps and relative to training time. We conclude that at 8×32 training batch size, Oat-VLA gets better success rates faster than at 8×64 but converges to around the same performance in the long run. For faster experiment turn-around, using Oat-VLA with a 8×32 appears advantageous. We also conclude that the improved success rates compared to OpenVLA are not just due to the increased batch size.

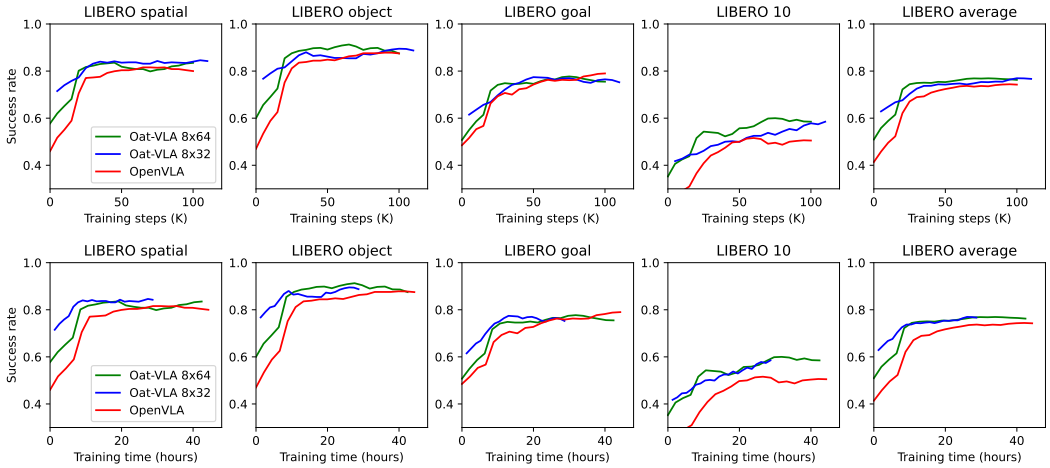


Figure 6: **Full fine-tuning** results at 8×64 and 8×32 batch sizes. Evaluationed on LIBERO, every 5K training steps, 100 evaluations per task suite, single seed. The plots are mean filtered using one evaluation before and after. **Top row**: success rates on LIBERO task suites relative to training steps. **Bottom row**: success rates on LIBERO task suites relative to training time, as measured on an 8×H100 node.

7.5 Training GPU memory usage and throughput

Table 5 shows the observed GPU memory usage and the throughput (number of training samples processed per second) for both LoRA and full fine-tuning. The maximum GPU memory usage (across the 8 GPUs in a single node) determines the maximum batch size. As can be seen, FSDP (enabled during full fine-tuning) is more balanced in terms of GPU memory usage than the LoRA implementation that OpenVLA uses, and allows for larger batch sizes despite the larger number of learnable parameters.

Table 5: GPU memory usage and throughput (training samples per second) on an 8×H100 node.

Model	Fine-tuning Method	Batch size	Mean GPU) memory (GiB)	Max GPU memory (GiB)	Training samples per second
OpenVLA	Full	8×32	57.6	61.6	157.4
Oat-VLA	Full	8×32	58.7	60.9	268.7
Oat-VLA	Full	8×64	78.9	79.6	320.4
OpenVLA	LoRA	8×16	68.0	74.1	224.6
Oat-VLA	LoRA	8×16	40.7	47.1	307.2
Oat-VLA	LoRA	8×32	55.8	62.4	353.3
Oat-VLA	LoRA	8×48	70.9	75.7	384.0

7.6 Real-world experiments

Dataset preparation. The dataset for real-world experiments is collected via human tele-operation, using a VR set and a controller to map the human motions and intentions, e.g., closing the gripper, to the robot. After collecting the trajectories, we apply some basic operations. First, we take the sequences collected at 60 Hz, and we subsample them at 10 Hz. Then, we extract actions in the format $[\Delta x, \Delta\theta, \text{grip}]$, that is delta position, delta orientation and the observed gripper position. Finally, we filter out no-motion operations, i.e. having Δx and $\Delta\theta$ equal to zero.

Evaluation. During evaluation, we map the VLA's commands to the robot as they are, with the exception of the gripper command. When the agent is closing the gripper, we close it by an additional 1-2 cms, to ensure the grasp of the object is secure. Otherwise, the agent will close the gripper barely enough to hold the object, as seen in the training dataset.

In the following, you find a detailed description of how each in-distribution task is executed and randomized. For out-of-distribution tasks, we follow the same procedures, but we swap one of the main actors in the scene (e.g. the object to pick or the placing target).

- *Place the banana in the green bowl.* There is a green bowl and 3 objects in the scene, a banana, a tomato and a blue hexagon, with assigned locations (see Figure 4). We perform 5 executions with the banana in one location, and 5 executions in another location. The banana orientation can be vertical (2/5 executions per location), horizontal (2/5 executions per location) or diagonal (1/5 executions per location).
- *Place the red cube in the brown bag.* There is a brown bag and 3 objects in the scene, a red cube, a banana, and a lettuce leaf, with assigned locations (see Figure 4). We perform 2 executions for the red cube in each location. The red cube orientation can be parallel to the robot's base, or tilted.
- *Place the tomato left of the lettuce.* There is 3 objects in the scene: a lettuce leaf, a carrot and a tomato. The object's positions are randomized, around certain areas. However, the main layout stays the same. We perform 5 executions with randomized locations. Success is assigned only for the tomato being placed on the table, close to the lettuce, and left of the lettuce.
- *Place the zucchini in front of the green cube.* There is 5 objects in the scene: a shape sorting box, a zucchini, a purple die, a rubber duck and a green cube. The object's positions are randomized, around certain areas. However, the main layout stays the same. We perform

4 executions with randomized locations. Success is assigned only for the zucchini being placed on the table, close to the green cube, and in front of the green cube.

7.7 Inference time, memory usage

As stated in Section 6, Oat-VLA does not offer an advantage in terms of execution time at inference time. Here we provide measurements on how inference time scales with batch size, and a detailed break-down of inference time.

7.7.1 Scaling of inference time with batch size

Figure 7 shows the inference time to generate a single action for various batch sizes. We measured on a single A100 GPU how long it takes the LLM to generate the action tokens from a tokenized prompt (i.e., excluding overhead such as language tokenization and the vision back-bone).

For OpenVLA the typical number of tokens in the prompt is in the order of 300 and the inference time grows quite linearly with the batch size. For Oat-VLA, the number of tokens is in the order of 60 and up to a batch size of 4 the inference time is constant as it is dominated by moving the weights of the LLM from GPU RAM to GPU cache. Methods that use parallel inference (e.g., OpenVLA-OFT [28]) might take advantage of this by increasing the batch size "for free".

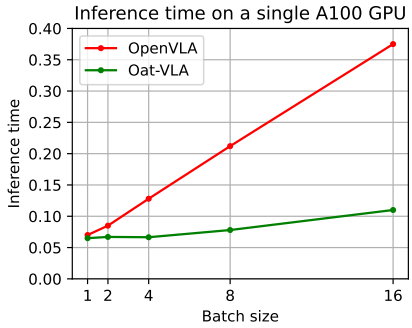


Figure 7: Inference time of the LLM, measured on a single A100 GPU, at different batch sizes.

7.7.2 Detailed break-down of inference time and memory usage

Here we provide additional data about the inference time of real-world experiments in the paper, performed on an RTX A5000. Under the prompt "place the banana in the green bowl", the action generation with OpenVLA takes 284ms, while with Oat-VLA it takes 268ms, *a reduction of 6% in inference time*. During inference time, the use of key-value caching reduces the amount of computation required after the first forward pass (i.e. first action token generated). The first forward pass takes 102ms in OpenVLA, of which 18ms are the visual encoding process. In Oat-VLA, the first forward pass takes 94ms, of which visual encoding takes 62ms. Following forward passes (for the remaining 6 action tokens) take 29ms for OpenVLA and 28ms for Oat-VLA. The memory usage for Oat-VLA is of 17.13GB and is higher than the 15.28GB required for OpenVLA, due to the additional modules employed. Thus, in addition to drastically reducing training time, Oat-VLA also slightly reduces the action prediction time, and overall inference time, at the costs of a relatively larger time and memory required for visual processing.

7.8 Ablation: agent-centric token grid size

Early on during the development of Oat-VLA, we performed an experiment to see if the size of the agent-centric token grid matters, with results shown in Table 6. We found the difference insignificant enough and settled for the smaller option.

Table 6: Success rates on LIBERO of an early version of Oat-VLA with two different agent-centric token grid sizes. 60K training steps (batch size 8×64), 100 evaluations per task suite, single seed.

LIBERO task suite	3×3 grid	5×5 grid
Spatial	77%	78%
Object	89%	90%
Goal	81%	79%
10	43%	44%

7.9 Ablation: object-centric tokenization: 7 slots or 15 slots

Early on during the development of Oat-VLA, we performed an experiment to see if the number of object-centric tokens matters, with results shown in Table 7. Again, we found the difference insignificant enough and settled for the smaller option.

Table 7: Success rates on LIBERO of an early version of Oat-VLA with two different numbers of object-centric tokens. 60K training steps (batch size 8×64), 100 evaluations per task suite, single seed.

LIBERO task suite	7 tokens	15 tokens
Spatial	82%	86%
Object	82%	86%
Goal	76%	72%
10	57%	40%

7.10 FT-Dinosaur slots as object-centric tokens

Early on in the development of Oat-VLA we experimented with using an object-centric representation directly as a way of regrouping image patch features into visual tokens. Specifically, we tried using *FT-Dinosaur* [48], a self-supervised segmentation model built on Slot Attention [60].

FT-Dinosaur learns a function $f : \mathbb{R}^{N \times F_v} \to \mathbb{R}^{M \times F_s}$ to compress the original ViT (Vision Transformer) patch embeddings into a fixed set of slots, where N is the number of ViT patches, M is the (user-specified) number of slots with $N \gg M$, and F_v and F_s are the ViT and slot feature dimensions, respectively. The model is trained with a reconstruction loss that decodes the original patch features from the slots, encouraging each slot to capture a coherent, semantically meaningful region of the scene. This yields natural segmentation while keeping the representation size constant across images, unlike typical segmentation pipelines whose output dimensionality varies with the number of masks. For this work we first train FT-Dinosaur to reconstruct DinoV2+SigLIP features on the Bridge [61] and LIBERO [62] datasets and then integrate it into OpenVLA. Crucially, integration into OpenVLA is done without the FT-Dinosaur reconstruction loss.

Starting from the OpenVLA checkpoint [53] we perform full fine-tuning on the full LIBERO dataset. Table 8 summarizes the performance obtained when the patch tokens of OpenVLA are replaced by seven FT-Dinosaur slots. To put this in perspective, seven slots compress the 16×16 ViT grid by a factor of 37. Compared with the original patch-based backbone, end-to-end slot fine-tuning narrows the gap on SPATIAL, OBJECT, and GOAL categories, but still lags far behind on 10 sequences.

Qualitative behavior. During roll-outs in LIBERO the gripper consistently reaches the vicinity of the target, yet fine-grained control is unreliable: the agent often mis-grasps, bumps, or drifts off contact after a correct approach. We attribute these failures to three complementary factors:

- *Lack of high-resolution cues.* With only seven slots, the policy may not "see" the millimeter-scale details needed for secure grasps or tight insertions.

Table 8: FT-Dinosaur object-centric results. 100 evaluations per LIBERO task suite, single seed. The Oat-VLA baseline is the evaluation of the 80K training step checkpoint in Section 4.1.

	Frozen FT-Dinosaur	Training FT-Dinosaur		Oat-VLA
	Training VLA	Frozen VLA	Training VLA	
Spatial	44%	43%	58%	82%
Object	58%	53%	79%	93%
Goal	44%	47%	56%	76%
10	14%	9%	10%	59%

- *Under-constrained fine-tuning.* Jointly training FT-Dinosaur without its decoder loss occasionally destabilizes learning, suggesting that the original reconstruction objective plays a crucial regularizing role.
- *Feature-space mismatch.* Even after fine-tuning, the slot features do not match the visual statistics of the original DinoV2+SigLIP patches used by OpenVLA.

These findings prompted us to start experimenting with agent-centric tokens, and at the same time we started using FT-Dinosaur only for its masks instead of for its slot features. This led to the Oat-VLA architecture described in Section 3.