Structure over Pixels: Learning Variable-Length Visual Programs

Piotr Wyrwiński [1] **Kacper Dobek** [1] **Krzysztof Krawiec** [1]

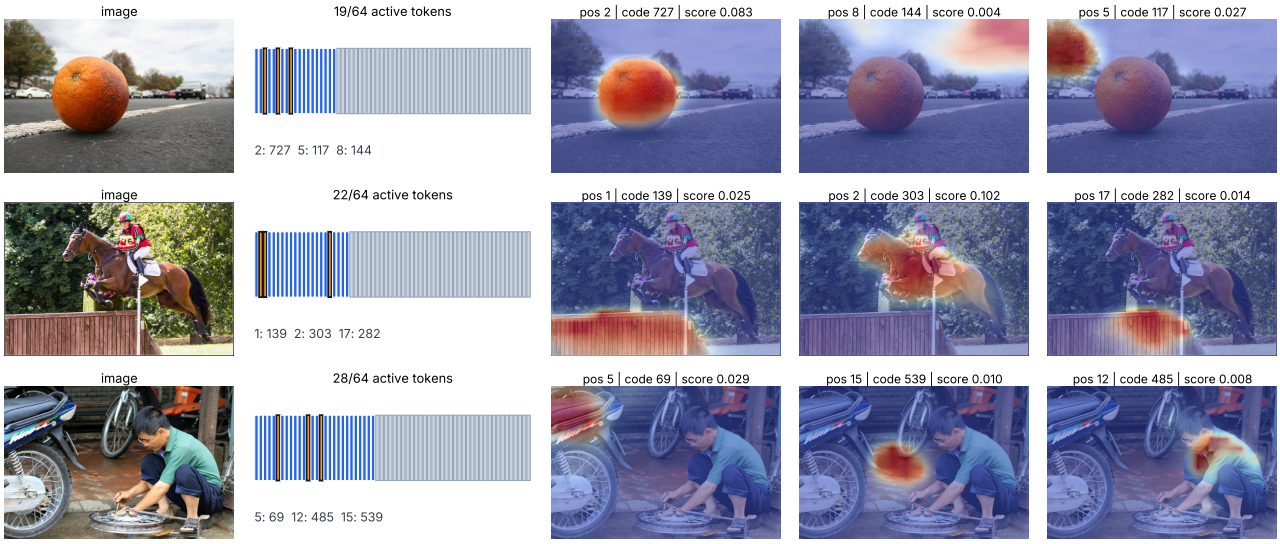


Figure 1. **Images as discrete adaptive programs.** Each image is encoded as a variable-length sequence of discrete code IDs. For each highlighted code position, we intervene by erasing that token from the program and reinterpreting the remaining sequence. The attribution panels show the resulting change in the DINO-aligned patch field: red denotes higher token influence and blue denotes lower influence within that panel. Localized red regions suggest that individual program tokens act as semantic handles over the scene. Heatmaps are normalized per panel for visualization; token scores are raw attribution magnitudes.

Abstract

Discrete visual tokenizers translate images into ordered sequences of codes, providing a natural representation for structural description of scenes. Yet existing adaptive tokenizers either require post-hoc search or select among a discrete set of pre-trained rates, rather than learning a continuous per-image sequence length coupled to the model and scene, and they typically train against pixel reconstruction, emphasizing texture rather than structure. We propose STROP, a discrete visual tokenizer architecture that forms structural scene representations and simultaneously learns how long an image's visual program should be. Using a four-phase curriculum supervised by local rate–distortion probes against frozen DINOv3 features, STROP optimizes a dedicated length head that estimates the active prefix length in a single forward pass. By bypassing pixel-level reconstruction gradients, the codebook is shaped entirely by the quality of higher-level latent representations. Program length grows with scene complexity, and signs of compositional structure emerge both in downstream dense-prediction transfer and in direct inspection of the learned code vocabulary.

[1]Institute of Computing Science, Poznan University of Technology, Poznan, Poland. Correspondence to: Piotr Wyrwiński <piotr.wyrwinski@cs.put.poznan.pl>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

1. Introduction

Deep neural networks prove unmatched at most computer vision (CV) tasks, mainly thanks to efficient representation learning. Their learned representations remain however verbose, distributed, convoluted, and thus hard to inspect and validate. These deficiencies are critical for many use cases that require robust scene interpretation, like robotics, planning, autonomous driving or reliable medical diagnosing. To address these limitations, models need to reason about scenes in terms of objects, their properties, and their spatial

relations. Such structural visual descriptions remain hard to learn, especially when supervision is scarce.

Object-centric learning and neural program induction pursue such descriptions in different ways (Locatello et al., 2020; Singh et al., 2022; Seitzer et al., 2023; Reed & de Freitas, 2016). *Discrete visual tokenizers* (DVTs) (Jia et al., 2025) offer a concrete starting point, since they already translate images into quasi-symbols. In those architectures, a scene is mapped to an ordered sequence of discrete codes (*program* in the following), which can be usually interpreted back into the spatial feature field. The model learns to generate such sequences without hand-designed predicates, object labels, or a-priori helper structures like scene graphs. Enforcing latent representations to discrete sequences of tokens from a fixed *codebook* facilitates inspection and, most importantly, opens the door to asking crucial research questions, like signs of code reuse, symbol grounding, or compositional structure. Figure 1 gives a qualitative preview of this view: images are represented as variable-length code sequences, and erasing individual codes induces localized changes in the recovered DINO-aligned field.

Most DVTs break the link between scene content and program length. VQ-VAE and VQGAN-style models produce a fixed spatial grid of codes, and recent 1D tokenizers replace that grid with a fixed number of latent slots (van den Oord et al., 2017; Esser et al., 2021; Yu et al., 2024b). The architecture chooses the sequence length before the image is seen. A smooth sky patch and a crowded street can change which codes are used, but both receive the same program length (sometimes referred to as code *budget*). This sits uneasily with cognitive principles like Gestalt principles, minimum description length, not to mention the Occam's razor. The number of tokens should correlate with the 'amount' of visual structure in the observed scene – this is the first step, if not a prerequisite, for forming principled structural representations.

Admittedly, recent discrete tokenizers make the budget adjustable. Some train decoders to tolerate prefixes; others use inference-time search, recurrent allocation, routing among fixed rates, or information-theoretic scores of input compressibility (Bachmann et al., 2025; Miwa et al., 2025; Yan et al., 2025; Duggal et al., 2025; Shen et al., 2025; Ye et al., 2025). The findings elaborated in those methods suggest that length control should be tightly linked with visual tokenization, rather than being delegated to some external mechanism.

Another prevailing characteristic in this area is the emphasis on fidelity of reconstruction. Indeed, previous works suggest that training DVTs with pixel-level reconstruction loss serves compression well. However, it can waste budget on capturing textural characteristics that carry little semantic or geometric structure. Feature-distillation tokenizers like BEiT v2 (Peng et al., 2022), DINOSAUR (Seitzer et al., 2023), REPA (Yu et al., 2025), and MAETok (Chen et al., 2025a) show that guiding training with teacher features and representation-aligned objectives can produce useful latent spaces and object-sensitive structure.

Motivated by the above findings, we propose STROP, a DVT architecture that comprises a pair of encoder-only transformers, generating and interpreting a causal VQ sequence; a learned length head decides how much of the sequence to expose. DINOv3 feature distillation supplies the training signal, and a detached pixel decoder serves as an inspection layer. We measure rate–quality curves and dense-prediction transfer, then inspect code reuse and spatial grounding. This gives four concrete contributions: **(i) Variable-length visual programs.** We supervise per-image program length with local rate–distortion probes of the trained tokenizer and use a prefix curriculum: random truncation first, predicted prefixes later. **(ii) A simple generator–interpreter tokenizer.** Two encoder-only transformers with learned queries write and read causally ordered VQ codes. DINOv3 feature distillation provides the main supervision. **(iii) Pixel probes without pixel training.** The convolutional decoder receives detached feature grids, so reconstructions are an inspection surface and pixel gradients do not shape the codebook. **(iv) Evaluation of representation and structure.** We report rate–quality tradeoffs and downstream dense-prediction performance, then analyze code reuse and spatial grounding as tests for a learned visual grammar.

2. Proposed approach

2.1. Architecture

Given an input image $x \in \mathbb{R}^{3\times H\times H}$, we seek a compact discrete program $\mathbf{c} = (c_1, \ldots, c_L)$ with $c_k \in \{1, \ldots, N\}$ from which a DINO-aligned patch field can be recovered, where L varies per image and $|CB|$ is the codebook size. The model (Figure 2) comprises five stages: a frozen *visual encoder* producing patch-wise features, a *program generator* that maps those features into a sequence of tokens, a vector-quantization bottleneck, an *interpreter* that expands the program back to a spatial feature grid, and a convolutional *decoder*. An auxiliary *length head* predicts L from the encoder features, enabling variable-length programs at inference without autoregressive sampling.

Encoder. We extract token-level features with a frozen DINOv3 ViT encoder (Siméoni et al., 2025). For an $H\times H$ image with patch size p, the encoder produces $S = (H/p)^2$ tokens of dimension d_{enc}. A layer norm followed by a linear projection maps every token from d_{enc} to a shared model dimension d, yielding source features $Z_e \in \mathbb{R}^{S\times d}$.

Program generator. To map the patch-based visual features into a sequence of tokens describing the scene, we

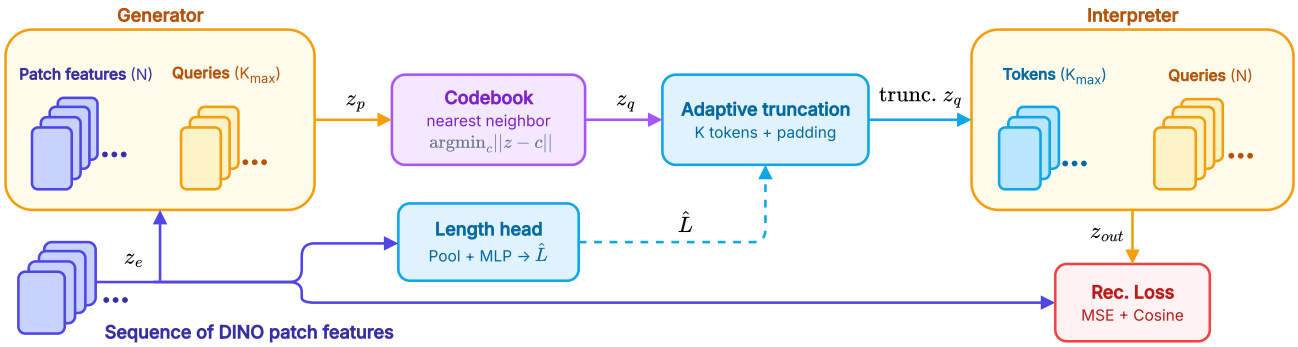


Figure 2. STROP architecture overview.

adopt an encoder-only transformer that processes Z_e with an appended sequence of K learnable query tokens. The concatenation $[Z_e; Q] \in \mathbb{R}^{(S+K)\times d}$ is processed by M standard transformer encoder layers. A causal mask prevents query token k from attending to query tokens $k' > k$, and source tokens from attending to any query, ensuring that no future-query information leaks through intermediate source representations. After encoding, only the K query outputs are retained as the pre-quantized, raw *program* $Z_p \in \mathbb{R}^{K\times d}$.

Vector quantization. A linear projection maps each token of Z_p into the codebook space $\mathbb{R}^{K\times d_c}$ ($d_c \ll d$), where an $|CB|$-entry codebook assigns discrete indices via nearest-neighbor lookup using Euclidean distance. The quantized tokens are projected back to dimension d, yielding $Z_q \in \mathbb{R}^{K\times d}$.

Length head. A lightweight MLP h_ϕ predicts the program length directly from the encoder features: $\hat{L} = K\sigma\big(h_\phi(\mathrm{pool}(Z_e))\big)$, where pool denotes mean pooling over the spatial dimension and σ is the sigmoid function. At inference, $\hat{L}$ is rounded and clipped to $[1, K]$; positions $k > \hat{L}$ in Z_q are masked from the interpreter. Training of this head is detailed in Section 2.3.

Interpreter. The interpreter mirrors the generator in mapping Z_q truncated to $\hat{L}$ tokens back to patch-level feature tokens. Padding is used only during training, where selected prefixes are padded for batching and masked so that the interpreter cannot attend to padded positions; no padded or zeroed program positions are passed at inference. The interpreter updates $(H/p)^2$ learnable grid queries by cross-attending to the retained tokens, treating the selected program as the key–value memory. The resulting grid-query outputs are projected with a 1×1 convolution and reshaped into a feature map $\hat{F} \in \mathbb{R}^{(H/p)\times(H/p)\times d}$ to obtain the patch-feature field.

2.2. Training objective

The entire training signal comes from frozen DINO patch features. The loss function is a composite of the following objectives.

Latent alignment. Given interpreted patches $\hat{F}$ and frozen DINO patches $F^\star$, we use $\mathcal{L}_{\text{lat}} = 1 - \frac{1}{P}\sum_{p=1}^{P}\cos(\hat{F}_p, F_p^\star) + \mathrm{MSE}(\hat{F}, F^\star)$, where the cosine term preserves teacher-feature geometry and MSE stabilizes scale.

Commitment loss $\mathcal{L}_{\text{commit}}$. The standard VQ regularization penalty $\lambda_{\text{q}}\|Z_p - \mathrm{sg}[Z_q]\|^2$ keeps pre-quantized embeddings close to their selected codebook entries.

Diversity loss $\mathcal{L}_{\text{div}}$. A utilization regularizer (λ_{div}=0.3), warmed up over 40k steps, penalizes non-uniform code usage and helps prevent codebook collapse

The full training objective is thus: $\mathcal{L} = \mathcal{L}_{\text{lat}} + \mathcal{L}_{\text{commit}} + \mathcal{L}_{\text{div}} + \mathcal{L}_{\text{len}}$, where $\mathcal{L}_{\text{len}}$ (Sec. 2.3) is activated in Phase 3 of the curriculum. The weights for each term are given in Table 5.

Codebook design. The vocabulary consists of $|CB|$=1024 entries of d_c=16 dimensions, updated via exponential moving average (EMA, τ=0.95) with ℓ_2-normalized codes. The low code dimension combined with ℓ_2 normalization places all entries on a hypersphere, stabilizing nearest-neighbor assignment and preventing dead codes from drifting far from the active manifold.

2.3. Adaptive program-length curriculum

The architectural setup is a fixed-budget bottleneck: given the ith sample x_i, the generator always emits K quantized tokens, but the interpreter only attends to the prefix of length $L_i \in \{1, \ldots, K\}$, with positions $k > L_i$ masked. We train the length head that predicts L_i with a four-phase curriculum (Fig. 3) outlined below and detailed in Appendix A.

Phase 1: Random truncation. The model is trained only on randomly truncated programs, which forces informative content to be front-loaded into earlier tokens. This phase is biased toward longer prefixes. The length head is not trained yet.

Phase 2: Oracle target estimation. We continue with random truncation but begin estimating for each sample how long the program *should* be examining the reconstruction quality for program prefixes that are slightly shorter

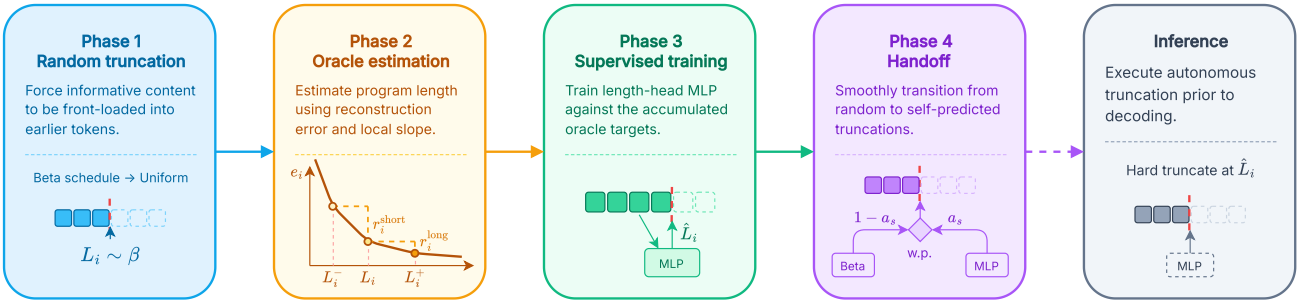


Figure 3. Training curriculum for autonomous truncation. **Phase 1:** Random Beta truncation front-loads information. **Phase 2:** Oracle lengths are estimated via reconstruction error and local slope. **Phase 3:** An MLP length-head is supervised on these oracle targets. **Phase 4:** A probabilistic handoff smoothly transitions from random to MLP-predicted lengths. **Inference:** Hard truncation relies solely on the learned MLP. Details on the curriculum can be found in Appendix A.

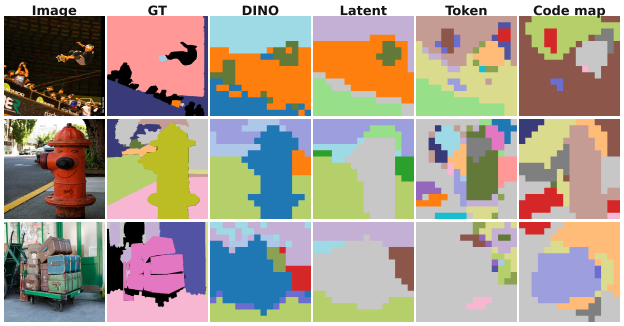


Figure 4. Unsupervised readouts on COCO-Stuff-27. DINO, Latent, and Token show k-means regions obtained from DINO teacher patch features, STROP latent patch features, and token-erasure attribution vectors, respectively. Colors in these panels denote arbitrary cluster IDs and are not semantic labels. Code-to-class shows a many-to-one mapping from program code IDs to semantic classes.

and longer than the drawn one. The 'right' lengths obtained in this way are not yet used to drive truncation in this phase—they are only accumulated.

Phase 3: Supervised length-head training. We activate a *length head*, a sigmoid MLP on pooled DINO encoder features, and train it against the targets calculated as in Phase 2. Truncation is still random, so the (not yet deployed) head learns from a stable target distribution.

Phase 4: Handoff to predicted lengths. We deploy the trained head by querying it at gradually increasing frequency. After a few epochs, all programs are truncated according to the predictions of the head, which continues to be trained throughout this phase.

At inference, no sampling is used: the estimated $\hat{L}_i$ is rounded and clipped to $[1, K]$, and the program is hard-truncated at that length before decoding.

3. Related work

Discrete visual tokenizers. Our model inherits the VQ-VAE lineage of discrete visual representations (van den Oord et al., 2017; Razavi et al., 2019), in which a continuous encoder is paired with a learned codebook. VQGAN (Esser et al., 2021) added adversarial and perceptual losses; subsequent work refined the prior with masked (Chang et al., 2022) or next-scale (Tian et al., 2024) prediction, replaced scalar VQ with finite-scalar (Mentzer et al., 2024) or residual (Lee et al., 2022) quantization, and stabilized the straight-through estimator (Fifty et al., 2025). A parallel line eliminates the learned codebook entirely: lookup-free quantization (LFQ) in MAGVIT-v2 (Yu et al., 2024a) binarizes each latent dimension to form an implicit exponential-size vocabulary, and Binary Spherical Quantization (BSQ) (Zhao et al., 2025) refines this by normalizing onto a hypersphere before binarization, bounding quantization error. Continuous relaxations such as SoftVQ-VAE (Chen et al., 2025b) and product-quantized variants such as ImageFolder (Li et al., 2024b) keep the budget fixed but compress the AR sequence. All of these tokenizers emit a fixed number of codes per image; the bottleneck capacity is set by the patch size, not by image content.

Variable-length tokenization and adaptive computation. A recent line compresses images into a small ordered set of *1D* tokens: TiTok shows that 32 tokens suffice for ImageNet reconstruction (Yu et al., 2024b). Several methods then make the budget itself adaptive. FlexTok (Bachmann et al., 2025) trains a rectified-flow decoder under nested dropout so any prefix of the token sequence decodes to a plausible image, and One-D-Piece (Miwa et al., 2025) applies the same idea (tail token drop) to a discrete tokenizer; ALIT (Duggal et al., 2025) runs recurrent rollouts that add latents until reconstruction quality saturates; ElasticTok (Yan et al., 2025) drops random tail tokens during training and selects a token count at inference via a fixed quality threshold; CAT (Shen et al., 2025) routes each image to one of three predetermined compression ratios using a caption-derived complexity score; and InfoTok (Ye et al., 2025) chooses among compression rates with an ELBO-based router. All of these methods either (i) require post-hoc search at inference or (ii) select among a discrete set of pre-trained rates. In contrast, STROP is trained end-to-end to *predict* a continuous per-image length via a four-phase curriculum. We frame the predictor as an instance of *adaptive computa-*

tion, a long line of work spanning Adaptive Computation Time (Graves, 2016), PonderNet (Banino et al., 2021), and recent token-level recursive depth in language models (Bae et al., 2025).

Feature-space and distillation-based reconstruction. Accumulating evidence indicates that semantic targets, rather than pixel targets, produce tokenizers that are more useful downstream. BEiT v2's VQ-KD distills a CLIP teacher into discrete codes for masked image modeling (Peng et al., 2022); RCG learns a diffusion prior over self-supervised representations (Li et al., 2024a); REPA (Yu et al., 2025) aligns the intermediate hidden states of a diffusion transformer with frozen DINOv2 features, and REPA-E (Leng et al., 2025) extends this signal end-to-end through the VAE tokenizer. VA-VAE (Yao & Wang, 2025) regularizes the VAE latent space toward a frozen vision foundation model; MAETok shows that masked feature prediction yields a discriminative tokenizer latent space (Chen et al., 2025a); and l-DeTok casts tokenizer training itself as latent denoising (Yang et al., 2025). The decoder in STROP distills frozen DINO patch features (Siméoni et al., 2025) via stop-gradient, accepting a hit on pixel fidelity in exchange for representational quality.

Compositional and object-centric representations. Our motivation that each code should correspond to a coherent visual concept or operation connects to slot-attention and object-centric learning. Slot Attention (Locatello et al., 2020) introduced iterative competitive cross-attention; SLATE (Singh et al., 2022) replaced its pixel-mixture decoder with a slot-conditioned autoregressive transformer over discrete tokens; DINOSAUR (Seitzer et al., 2023) showed that reconstructing self-supervised features (rather than pixels) is what makes slots emerge on real images; SlotDiffusion (Wu et al., 2023) pairs slots with a latent-diffusion decoder. Wen et al. (Wen et al., 2025) take a complementary path, baking PCA-like ordering into the 1D token sequence so that each successive token contributes monotonically decreasing variance. STROP differs from this blueprint in using a *causally ordered* program rather than a permutation-invariant set, so length is meaningful and the prior is autoregressive. We build on recent analyses of compositional structure in generative models (Liang et al., 2024; Okawa et al., 2023) and on atomic-decomposition autoencoders (Newson & Traonmilin, 2023).

Generator–interpreter framing and query-based decoders. The terminology echoes Neural Programmer-Interpreters (Reed & de Freitas, 2016), which learn explicit symbolic programs. Architecturally, our use of a small set of learnable queries that compress a long input into a fixed latent, paired with a mirrored decoder that re-expands it through grid queries, follows the query-based design space of DETR (Carion et al., 2020), Perceiver (Jaegle et al., 2021), and Perceiver IO (Jaegle et al., 2022). STROP's contribution within this space is the *variable-length* prefix interface and its end-to-end length supervision.

4. Evaluation Protocol

We evaluate STROP as an executable discrete representation. Structural diagnostics test whether active program tokens induce localized, reusable effects; downstream probes test how much task information is preserved in frozen DINO features, interpreted STROP fields, and raw program vectors. All probes are trained after STROP is frozen and are not part of tokenizer training.

Program structure. Let Y be the interpreted DINO-aligned patch field from the full active program, and Y_{-k} the field after deleting active token k and reinterpreting the remaining prefix. We define the counterfactual erasure map $\Delta_{p,k} = 1 - \cos(Y_p, Y_{-k,p})$, which measures the patchwise effect of editing the program rather than visualizing attention. We assign patches to their most influential token, $a(p) = \arg\max_k \Delta_{p,k}$, compare the induced regions to available semantic/object masks. For pair interactions, let $D_i(p) = \Delta_{p,i}$ and let $D_{ij}(p)$ denote the erasure map after deleting tokens i and j jointly; we define $\mathrm{Syn}_{ij}(p) = D_{ij}(p) - D_i(p) - D_j(p)$. Active pairs are compared against random active pairs and inactive-token controls.

Code reuse and unsupervised readouts. We summarize vocabulary structure with active-code count, effective vocabulary size, usage concentration, code purity, and code entropy. We also compare unsupervised region readouts from DINO patches, interpreted STROP patches, erasure-attribution vectors, and code-to-class mappings (Fig. 4). These analyses test non-random program structure, not supervised segmentation quality.

Supervised downstream probes. For semantic segmentation, we train spatial probes on frozen DINO patches and interpreted STROP patches, and a spatial-query probe on active quantized vectors z_q (Fig. 5); we report mIoU, pixel accuracy, and mean class accuracy on PASCAL VOC 2012 (Everingham et al., 2015) (referred to as VOC), COCO-Stuff-27 (Caesar et al., 2018), ADE20K (Zhou et al., 2019), and Cityscapes (Cordts et al., 2016). For multi-label classification, we train linear probes on pooled representations and report mAP on VOC and COCO-Stuff-27. For depth estimation, we train lightweight depth probes and report standard depth metrics on NYUv2 (Nathan Silberman & Fergus, 2012).

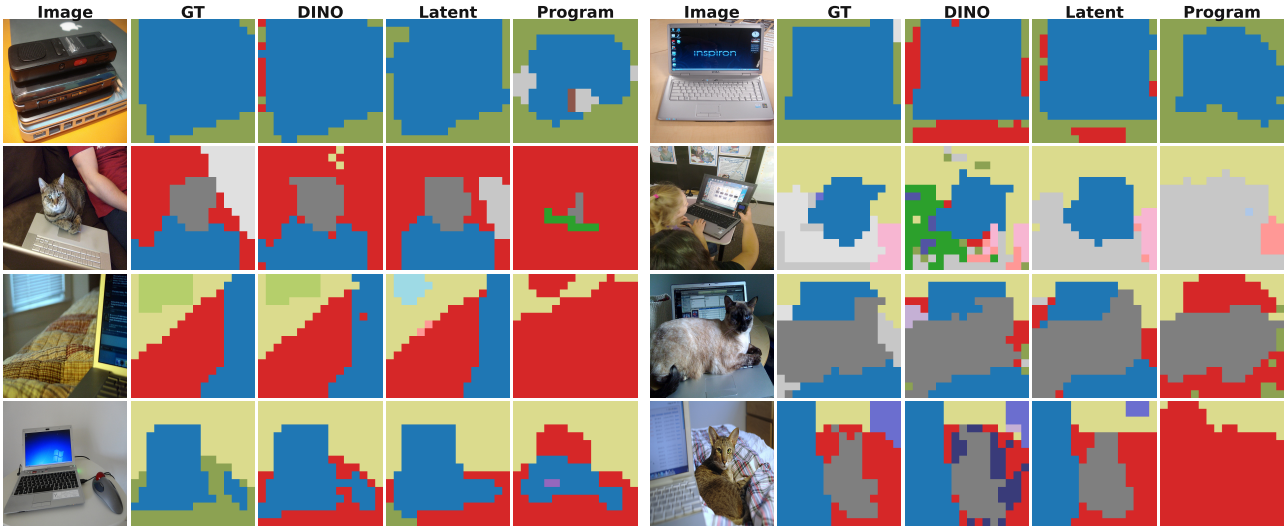


Figure 5. Supervised downstream segmentation probes on COCO-Stuff-27. DINO and Latent show predictions from linear patch probes trained on frozen DINO teacher features and STROP latent patch features, respectively. Program shows predictions from a probe trained only on the quantized program representation.

Table 1. Latent reconstruction quality and codebook utilization, averaged over five evaluation datasets (VOC, COCO-Stuff-27, ADE20K, Cityscapes, NYUv2). Cos and R^2 vs. frozen DINO teacher patches; CB% = fraction of 1024-entry codebook used; Eff.% = perplexity-derived effective utilization.

	Trained on COCO					Trained on ImageNet				
Variant	Cos↑	R^2 ↑	RMSE↓	CB%↑	Eff.%↑	Cos↑	R^2 ↑	RMSE↓	CB%↑	Eff.%↑
S/P-S	0.855	0.734	0.196	56.1	44.7	0.849	0.724	0.199	56.2	44.5
S/P-M	0.835	0.697	0.208	98.0	78.6	0.851	0.727	0.198	98.1	72.0
B/P-S	0.801	0.643	0.246	53.2	41.2	0.763	0.583	0.266	77.2	60.0
B/P-M	0.801	0.642	0.246	94.8	78.7	0.801	0.643	0.246	91.2	75.3
L/P-M	0.725	0.526	0.198	95.9	81.7	0.723	0.527	0.198	92.2	72.9

5. Results

5.1. Experimental setup

We train five model variants by sweeping the visual encoder's (DINOv3 ViT-S/B/L (Siméoni et al., 2025)) and generator/interpreter's capacity (small STROP-S, medium STROP-M). The encoder is always frozen; trainable parameters comprise the generator, interpreter, quantizer projections, and length head. Unless otherwise stated, models use programs of length K=64 with 2D patch and learned prefix positional encodings, a vector quantizer with $|CB|$=1024 codes of dimension d_c=16, ℓ_2-normalized code vectors, EMA updates with τ=0.95, and commitment weight β=1.0. The decoder consists of four upsampling blocks, each comprising an upsampling layer, a convolution, batch normalization, and a Gated Linear Unit (GLU). It is trained using a stop-gradient operation applied to the interpreter output. The interpreter mirrors the generator in every variant, sharing d_{model}, depth, number of heads (always 8), and feed-forward width. Table 6 in Appendix B lists all configurations with parameter budgets and training cost. Unless otherwise stated, training engages curriculum learning of the length head (Sec. 2.3, Appendix A), parametrized according to Table 4: Phase 1 lasts 100k and 200k steps for STROP-S and STROP-M respectively, and each of the subsequent three phases lasts 50k steps.

Datasets. All five variants are trained on both ImageNet-1k (Deng et al., 2009) and COCO 2017 (Lin et al., 2014) at 256×256 resolution. We additionally train on CLEVR (Johnson et al., 2016) at 128×128 with reduced program length (K=32) and smaller codebooks ($|CB| \in \{32, 64, 128\}$) to study codebook sizing and program structure on a controlled scene domain (Appendix F).

Optimization. All variants share the same optimizer and schedule: AdamW with peak learning rate 3×10^{-4}, batch size 32, and 5×10^5 total steps. The learning rate follows a warmup-hold-cosine schedule (30k warmup, hold to 100k, cosine decay to 10^{-4}).

5.2. Latent alignment and scaling

Table 1 evaluates the executed discrete program by comparing its interpreted patch field to frozen DINOv3 features. Under the same 64-code, 1024-entry bottleneck, DINO-S is easiest to compress: DINO-S/STROP-S gives the best COCO alignment (.855 cosine, R^2 = .734), while DINO-

Table 2. Linear-probe mIoU (%, ↑) on four segmentation benchmarks. STROP is trained on COCO; FlexTok and One-D-Piece are off-the-shelf adaptive tokenizers without STROP training. Stuff-27 denotes COCO-Stuff-27. Teacher rows probe frozen DINO patch features as an upper bound. For STROP, latent probes the interpreter's spatial latent field directly, whereas program forms a patch-level feature field by attention-pooling over the active program token vectors z_q at each spatial location before applying the same linear probe. Selected variants are shown; for all results please refer to Appendix Table 10. The best non-teacher value in each column is **bolded**.

Encoder	Model	Repr.	VOC	Stuff-27	ADE20k	Citysc.
Adaptive tokenizer						
FlexTok	D12-D12-IN1k	latent	48.4	21.6	9.4	**33.3**
DINOv3-S	One-D-Piece-S-256	latent	16.9	10.6	2.9	19.9
Teacher (frozen DINO)						
DINO-S	—	teacher	68.3	36.3	14.5	36.1
DINO-B	—	teacher	78.4	44.6	24.9	55.4
STROP (ours)						
DINO-S	STROP-S	latent	58.9	32.5	9.6	23.9
DINO-S	STROP-S	program	38.7	20.0	6.6	24.2
DINO-B	STROP-S	latent	**67.5**	**39.2**	**15.4**	30.4
DINO-B	STROP-S	program	48.4	19.5	7.9	28.3
DINO-B	STROP-M	latent	61.9	36.3	13.4	25.9
DINO-B	STROP-M	program	43.9	18.0	6.3	22.6

S/STROP-M is best on ImageNet (.851, $R^2 = .727$). DINO-B remains around .80 cosine and $R^2 \approx .64$, and DINO-L drops to $\sim .72/.53$, consistent with a fixed-rate bottleneck facing higher-dimensional teacher features. Larger generator mainly increase vocabulary coverage: STROP-M uses 91–98% of the codebook, but this does not monotonically improve alignment, e.g., COCO DINO-S/STROP-S outperforms DINO-S/STROP-M despite lower codebook usage. Per-dataset alignment is in Appendix C.

Compression. With 256×256 inputs and a patch size of 16, every encoder emits a fixed grid of 256 patch tokens (with dimension D) per image, giving per-image feature tensors of shape 256×384 (DINO-S), 256×768 (DINO-B), and 256×1024 (DINO-L). STROP compresses each of these backbone-dependent tensors into an adaptive sequence of at most 64 discrete codes drawn from a codebook of size $|CB|$=1024 (d_c=16), corresponding to a hard ceiling of $64 \cdot \log_2 N = 640$ bits per image regardless of encoder choice. Against the continuous float32 source representations of $256{\cdot}D{\cdot}32$ bits, this yields worst-case (full 64-code) compression ratios of $\sim$4,915$\times$ (DINO-S), $\sim$9,830$\times$ (DINO-B), and $\sim$13,107$\times$ (DINO-L), with the use of length head typically leading to shorter sequences and higher ratios. The total source entropy the quantizer must absorb scales linearly with D, but the bit budget is held constant, so larger backbones face a strictly harder rate–distortion problem: DINO-L must compress $2.67\times$ more raw feature dimensions than DINO-S into the same 640-bit envelope.

5.3. Downstream decodability

We freeze STROP and run linear probes on three representations: frozen DINO features, the interpreted STROP field, and the raw quantised program. We also include two off-the-shelf adaptive tokenizers, FlexTok (Bachmann et al., 2025) and One-D-Piece (Miwa et al., 2025), as external baselines. Tables 2 and 3 report selected results, and the full segmentation, classification, and NYUv2 depth probes appear in Appendix E.

The interpreted STROP field consistently outperforms both adaptive tokenizer baselines on the tasks that test representational content. On the four segmentation benchmarks, STROP with a DINO-B encoder beats FlexTok by a clear margin on VOC, COCO-Stuff-27, and ADE20K, and reaches a comparable level on Cityscapes, where FlexTok retains an edge. One-D-Piece sits well below both. On multi-label classification, the gap widens. The best STROP variant matches or marginally exceeds the frozen DINO teacher and beats FlexTok by roughly ten mean-mAP points, while One-D-Piece trails by more than twenty. These comparisons suggest that pairing a DINO-aligned latent objective with the variable-length program interface preserves substantially more task-relevant content than tokenizers trained with pixel-leaning reconstruction objectives.

Depth follows the same pattern with a smaller margin. The DINO-B/STROP-S interpreter reaches the best NYUv2 numbers we measured among STROP variants, but still trails the strongest teacher. Detailed depth metrics are in the appendix.

Table 3. Linear probe mAP (%, ↑) on multi-label classification. STROP trained on ImageNet; FlexTok and One-D-Piece are off-the-shelf adaptive tokenizers (no STROP training). Stuff-27 = COCO-Stuff-27; Mean = mean of Stuff-27 and VOC mAP. Teacher uses mean-pooled DINO patch features; latent is mean-pooled over the interpreter's spatial latent field; program is attention-pooled over the active program token vectors z_q. Selected variants; for all results please refer to Appendix Table 9. Best non-teacher value per column is **bolded**.

Encoder	Model	Repr.	Stuff-27	VOC	Mean
Adaptive tokenizer					
FlexTok	D12-D12-IN1k	latent	53.2	61.7	57.4
DINOv3-S	One-D-Piece-S-256	latent	48.1	39.0	43.6
Teacher (frozen DINO)					
DINO-S	—	teacher	56.2	59.4	57.8
DINO-B	—	teacher	61.2	74.7	67.9
STROP (ours)					
DINO-S	STROP-S	latent	56.6	61.5	59.1
DINO-S	STROP-S	program	34.9	17.8	26.3
DINO-B	STROP-S	latent	**61.7**	74.1	67.9
DINO-B	STROP-S	program	36.1	18.5	27.3
DINO-B	STROP-M	latent	61.6	**75.2**	**68.4**
DINO-B	STROP-M	program	31.3	14.7	23.0

Direct probes on the raw program are weaker than probes on the interpreted field across every task. We read this gap as a lower bound on linear readability rather than as a measure of information loss. The probe sees active code vectors but not token order, prefix structure, token-token interactions, or the interpreter's learned spatial execution map. Running the same codes through the interpreter recovers much stronger task signal, which is why we treat the program as an executable representation rather than as a drop-in linear feature.

5.4. Analysis of trained models

Program length tracks scene complexity. STROP allocates longer programs to richer scenes. On CLEVR the correlation between active program length and ground-truth object count is strong (Pearson $r \approx 0.74$; Figure 7a). On COCO, where scene complexity is a noisier quantity, the correlation against the number of unique semantic classes is positive but weaker (Figure 7b). The same trend holds against the spatial variance of patch-level DINO features (Figure 8). CLEVR ablations and the full correlation statistics are in Appendix F.

LLM scene decoding. We further test whether frontier LLMs can decode CLEVR scene JSONs from code sequences in a 50-shot setting, with the prompt shown in Appendix G. Both Qwen3.5 and DeepSeek-V3.1 recover object count and spatial position better than a dataset-prior baseline. Neither model beats the baseline on attribute accuracy. The pattern suggests that the codes expose coarse layout to a generalist LLM more readily than they expose fixed attribute bindings such as color or material.

6. Limitations

The most visible limitation is the readability of the raw program. Direct probes on the quantized tokens trail interpreter-mediated probes across every downstream task we ran (Section 5.3). STROP codes are therefore best viewed as an executable intermediate representation rather than as drop-in linear features. We cannot yet separate how much of this gap is due to the shallow pooling used by the probe and how much is due to information loss at quantization. Stronger program-native probes that respect token order and prefix length are needed to give a sharper answer.

Latent alignment is also an incomplete predictor of transfer. The DINO-S encoder gives the best reconstruction scores in Table 1, but DINO-B is often stronger for segmentation and depth. Useful structure in the teacher features can therefore survive quantization even when the raw reconstruction loss is higher, and the current metrics do not capture this fully. A more discriminative latent-quality measure, perhaps one that weights patch directions by their downstream usefulness, would help diagnose this behavior.

The Cityscapes segmentation result is a third caveat. STROP with a DINO-B encoder beats FlexTok on three of the four segmentation benchmarks but lags on Cityscapes (Table 2). FlexTok's larger training signal and pixel-aware decoding may be better suited to the dense, fine-grained street scenes that benchmark contains. Whether this gap reflects the choice of teacher, the program-length budget, or the COCO training distribution is an open question.

Finally, LLM scene decoding (Appendix G) recovers coarse layout but not reliable attribute binding. The learned code-

book is therefore not yet a fully transparent symbolic interface for downstream language models. Closing this gap, either by changing the codebook structure or by aligning code identities with a captioning signal, is a direction we leave to future work.

7. Conclusions and future work

We presented STROP, a discrete visual tokenizer that adapts its program length to scene complexity through a four-phase curriculum supervised by local rate–distortion probes against frozen DINOv3 features. The interpreted latent field tracks the teacher well enough to retain most of its downstream signal on segmentation, classification, and depth probes. Token-erasure diagnostics show that individual program tokens act as spatially coherent handles over the scene. Compared with the two adaptive tokenizers we used as external baselines, FlexTok and One-D-Piece, STROP preserves more task-relevant content on every benchmark we measured except Cityscapes.

We see three concrete directions for follow-up. First, the rate–quality frontier can be pushed further by scaling the codebook, the code dimension, and the maximum program length beyond the current configuration. The worst-case 640-bit envelope is held fixed across the encoder sweep, and preliminary scans suggest that adding a few more bits per token closes a meaningful fraction of the DINO-L alignment gap. Second, the readability gap between raw program probes and interpreter-mediated probes calls for program-native probing protocols. Token-order ablations, shuffled-program interpreter controls, and decoders that read the program as a sequence rather than as a set would help isolate what the prefix order actually encodes. Third, the LLM scene-decoding results suggest that the codebook already exposes layout but not attribute binding. Grammar inference over learned code sequences, paired with LLM-based visual question answering conditioned on program token IDs, is a promising route to make the codebook a usable symbolic interface.

Acknowledgements

PW is funded by the statutory funds of Poznan University of Technology and the Polish Ministry of Science and Higher Education, grant no. 0311/SBAD/0774 and Research Grant of National Science Center, grant no. 2024/53/N/ST6/03961. KD and KK are funded by the statutory funds of Poznan University of Technology and the Polish Ministry of Science and Higher Education, grant no. 0311/SBAD/0770 and the Research Grant of National Science Center, grant no. 2025/57/B/ST6/03737. We gratefully acknowledge the Polish high-performance computing infrastructure PCSS PLCloud for providing computational resources and support under grant no. pl0603-01.

References

Bachmann, R., Allardice, J., Mizrahi, D., Fini, E., Kar, O. F., Amirloo, E., El-Nouby, A., Zamir, A., and Dehghan, A. FlexTok: Resampling images into 1D token sequences of flexible length. In *International Conference on Machine Learning (ICML)*, 2025.

Bae, S., Kim, Y., Bayat, R., Kim, S., Ha, J., Schuster, T., Fisch, A., Harutyunyan, H., Ji, Z., Courville, A., and Yun, S.-Y. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Banino, A., Balaguer, J., and Blundell, C. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Caesar, H., Uijlings, J., and Ferrari, V. Coco-stuff: Thing and stuff classes in context, 2018. URL https://arxiv.org/abs/1612.03716.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*, 2020.

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. MaskGIT: Masked generative image transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Chen, H., Han, Y., Chen, F., Li, X., Wang, Y., Wang, J., Wang, Z., Liu, Z., Zou, D., and Raj, B. Masked autoencoders are effective tokenizers for diffusion models. In *International Conference on Machine Learning (ICML)*, 2025a.

Chen, H., Wang, Z., Li, X., Sun, X., Chen, F., Liu, J., Wang, J., Raj, B., Liu, Z., and Barsoum, E. SoftVQ-VAE: Efficient 1-dimensional continuous tokenizer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025b.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Duggal, S., Isola, P., Torralba, A., and Freeman, W. T. Adaptive length image tokenization via recurrent allocation. In *International Conference on Learning Representations (ICLR)*, 2025.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

Fifty, C., Junkins, R. G., Duan, D., Iger, A., Liu, J. W., Amid, E., Thrun, S., and Ré, C. Restructuring vector quantization with the rotation trick. In *International Conference on Learning Representations (ICLR)*, 2025.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, 2021.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., Hénaff, O., Botvinick, M. M., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations (ICLR)*, 2022.

Jia, J., Gao, J., Xue, B., Wang, J., Cai, Q., Chen, Q., Zhao, X., Jiang, P., and Gai, K. From principles to applications: A comprehensive survey of discrete tokenizers in generation, comprehension, recommendation, and information retrieval, 2025. URL https://arxiv.org/abs/2502.12448.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890, 2016. URL http://arxiv.org/abs/1612.06890.

Lee, D., Kim, C., Kim, S., Cho, M., and Han, W.-S. Autoregressive image generation using residual quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Leng, X., Singh, J., Hou, Y., Xing, Z., Xie, S., and Zheng, L. REPA-E: Unlocking VAE for end-to-end tuning with latent diffusion transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Li, T., Katabi, D., and He, K. Return of unconditional generation: A self-supervised representation generation method. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a.

Li, X., Qiu, K., Chen, H., Kuen, J., Gu, J., Raj, B., and Lin, Z. ImageFolder: Autoregressive image generation with folded tokens. 2024b.

Liang, Q., Liu, Z., Ostrow, M., and Fiete, I. How diffusion models learn to factorize and compose. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Mentzer, F., Minnen, D., Agustsson, E., and Tschannen, M. Finite scalar quantization: VQ-VAE made simple. In *International Conference on Learning Representations (ICLR)*, 2024.

Miwa, K., Sasaki, K., Arai, H., Takahashi, T., and Yamaguchi, Y. One-D-Piece: Image tokenizer meets quality-controllable compression. *arXiv preprint arXiv:2501.10064*, 2025.

Nathan Silberman, Derek Hoiem, P. K. and Fergus, R. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Newson, A. and Traonmilin, Y. Disentangled latent representations of images with atomic autoencoders. In *Sampling Theory and Applications Conference (SampTA)*, 2023.

Okawa, M., Lubana, E. S., Dick, R. P., and Tanaka, H. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. *arXiv preprint arXiv:2310.09336*, 2023.

Peng, Z., Dong, L., Bao, H., Ye, Q., and Wei, F. BEiT v2: Masked image modeling with vector-quantized visual tokenizers. *arXiv preprint arXiv:2208.06366*, 2022.

Razavi, A., van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Reed, S. and de Freitas, N. Neural programmer-interpreters. In *International Conference on Learning Representations (ICLR)*, 2016.

Seitzer, M., Horn, M., Zadaianchuk, A., Zietlow, D., Xiao, T., Simon-Gabriel, C.-J., He, T., Zhang, Z., Schölkopf, B., Brox, T., and Locatello, F. Bridging the gap to real-world object-centric learning. In *International Conference on Learning Representations (ICLR)*, 2023.

Shen, J., Tirumala, K., Yasunaga, M., Misra, I., Zettlemoyer, L., Yu, L., and Zhou, C. CAT: Content-adaptive image tokenization. *arXiv preprint arXiv:2501.03120*, 2025.

Siméoni, O., Vo, H. V., Seitzer, M., Baldassarre, F., Oquab, M., Jose, C., Khalidov, V., Szafraniec, M., Yi, S., Ramamonjisoa, M., Massa, F., Haziza, D., Wehrstedt, L., Wang, J., Darcet, T., Moutakanni, T., Sentana, L., Roberts, C., Vedaldi, A., Tolan, J., Brandt, J., Couprie, C., Mairal, J., Jégou, H., Labatut, P., and Bojanowski, P. DINOv3. *arXiv preprint arXiv:2508.10104*, 2025.

Singh, G., Deng, F., and Ahn, S. Illiterate DALL-E learns to compose. In *International Conference on Learning Representations (ICLR)*, 2022.

Tian, K., Jiang, Y., Yuan, Z., Peng, B., and Wang, L. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Wen, X., Zhao, B., Elezi, I., Deng, J., and Qi, X. "Principal components" enable a new language of images. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Wu, Z., Hu, J., Lu, W., Gilitschenski, I., and Garg, A. Slot-Diffusion: Object-centric generative modeling with diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yan, W., Zaharia, M., Mnih, V., Abbeel, P., Faust, A., and Liu, H. ElasticTok: Adaptive tokenization for image and video. In *International Conference on Learning Representations (ICLR)*, 2025.

Yang, J., Li, T., Fan, L., Tian, Y., and Wang, Y. Latent denoising makes good visual tokenizers. *arXiv preprint arXiv:2507.15856*, 2025.

Yao, J. and Wang, X. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Ye, H., He, Q., Han, J., Li, P., Fan, J., Hao, Z., Reda, F., Balaji, Y., Chen, H., Liu, S., Yao, A., Zou, J., Ermon, S., Wang, H., and Liu, M.-Y. InfoTok: Adaptive discrete video tokenizer via information-theoretic compression. *arXiv preprint arXiv:2512.16975*, 2025.

Yu, L., Lezama, J., Gundavarapu, N. B., Versari, L., Sohn, K., Minnen, D., Cheng, Y., Gupta, A., Gu, X., Hauptmann, A. G., Gong, B., Yang, M.-H., Essa, I., Ross, D. A., and Jiang, L. Language model beats diffusion - tokenizer is key to visual generation. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=gzqrANCF4g.

Yu, Q., Weber, M., Deng, X., Shen, X., Cremers, D., and Chen, L.-C. An image is worth 32 tokens for reconstruction and generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b.

Yu, S., Kwak, S., Jang, H., Jeong, J., Huang, J., Shin, J., and Xie, S. Representation alignment for generation: Training diffusion transformers is easier than you think. In *International Conference on Learning Representations (ICLR)*, 2025.

Zhao, Y., Jiang, H., Xu, Z., Yang, C., Adeli, E., and Krähenbühl, P. Spherical leech quantization for visual tokenization and generation, 2025. URL https://arxiv.org/abs/2512.14697.

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

A. Adaptive program-length curriculum details

This appendix gives the full parameterization of the adaptive-length curriculum used in Section 2.3.

The generator emits a fixed sequence of K quantized tokens for every image, but the interpreter attends only to a prefix of length $L_i \in \{1, \dots, K\}$ for sample x_i. Tokens with positions $k > L_i$ are masked during training; at inference they are not passed to the interpreter. Curriculum design therefore reduces to the prediction of L_i during training, which we schedule across the following four phases.

Phase 1: Random truncation. The model is trained only on randomly truncated programs, which forces informative content to be front-loaded into earlier tokens. Lengths are sampled as

$$u_i \sim \mathrm{Beta}(\alpha_t, 1), \qquad L_i = \mathrm{clip}\big(\mathrm{round}\big(L_{\min}^{\mathrm{trunc}} + u_i(K - L_{\min}^{\mathrm{trunc}})\big)\,,\, L_{\min}^{\mathrm{trunc}},\, K\big)\,, \tag{1}$$

with the schedule $\alpha_s = \alpha_0 + (1 - \alpha_0)\min(s/T_{\mathrm{bias}}, 1)$, where s denotes the global optimization step. Since $\alpha_0 > 1$, early training is biased toward longer prefixes (easier reconstruction, faster initial convergence) and anneals linearly to a uniform distribution at $s = T_{\mathrm{bias}}$.

Phase 2: Oracle target estimation. We continue with random truncation but begin estimating for each sample how long the program *should* be. The base oracle uses the feature-space reconstruction error at the active length, $e_i(L_i) = \mathcal{L}_{\mathrm{lat}}(\hat{F}_i(L_i), F_i^\star)$, normalized by a batch-level EMA $\bar{e}_t = \rho\,\bar{e}_{t-1} + (1-\rho)\,\frac{1}{B}\sum_i e_i(L_i)$:

$$L_i^{\mathrm{base}} = \mathrm{clip}\Big(\tfrac{e_i(L_i)}{\bar{e}_t + \epsilon}\,\beta K,\; L_{\min}^{\mathrm{oracle}},\; L_{\max}^{\mathrm{oracle}}\Big)\,. \tag{2}$$

Samples reconstructing worse than the EMA running average receive longer targets; easier samples receive shorter ones. The hyperparameter β sets the average target length when $e_i \approx \bar{e}_t$. Targets are not yet used to drive truncation in this phase—they are only accumulated.

As L_i^{base} conflates sample difficulty with length adequacy (a hard sample has large error at any length), we complement it with a signal of the local slope of the rate–feature-distortion curve at L_i, which captures how much the *additional* tokens reduce error. For a probe radius δ, we calculate the reconstruction error at the clipped neighbors $L_i^{\pm} = \mathrm{clip}(L_i \pm \delta,\, L_{\min}^{\mathrm{trunc}},\, K)$ and form one-sided relative slopes

$$r_i^{\mathrm{short}} = \max\Big(0,\; \tfrac{e_i^- - e_i}{e_i + \epsilon}\Big)\,, \qquad r_i^{\mathrm{long}} = \max\Big(0,\; \tfrac{e_i - e_i^+}{e_i + \epsilon}\Big)\,, \tag{3}$$

which measure how much shortening hurts and how much lengthening still helps. We average over non-clamped probes, maintain EMAs $\bar{r}^{\mathrm{short}}, \bar{r}^{\mathrm{long}}$, and squash with a temperature-τ saturation $u_\bullet = \tanh(\bar{r}^\bullet/\tau) \in [0, 1]$. The pair $(u_{\mathrm{short}}, u_{\mathrm{long}})$ induces a soft three-way decision through the partition of unity

$$w_{\mathrm{compress}} = (1 - u_{\mathrm{short}})(1 - u_{\mathrm{long}}), \quad w_{\mathrm{keep}} = u_{\mathrm{short}}(1 - u_{\mathrm{long}}), \quad w_{\mathrm{extend}} = u_{\mathrm{long}}, \tag{4}$$

corresponding to three regimes: *compress* when the curve is flat near L_i, *keep* near the elbow (only shortening hurts), and *extend* whenever lengthening still pays. Extension dominates by design—unused marginal utility should be exploited regardless of the lower-side signal. The corrected target is

$$\tilde{L}_i = \mathrm{clip}\big(m\,L_i^{\mathrm{base}},\; L_{\min}^{\mathrm{oracle}},\; L_{\max}^{\mathrm{oracle}}\big)\,, \qquad m = \sum_k m_k\, w_k, \tag{5}$$

with defaults $m_{\mathrm{compress}} < 1$, $m_{\mathrm{keep}} = 1$, and $m_{\mathrm{extend}} > 1$.

Phase 3: Supervised length-head training. We activate a length head $\hat{L}_i = K h_\phi(\mathrm{pool}(Z_{e,i}))$, a sigmoid MLP on pooled DINOv3 encoder features, supervised against the oracle target:

$$\mathcal{L}_{\mathrm{len}} = \tfrac{1}{B}\sum_i \Big(\tfrac{\hat{L}_i - \mathrm{sg}[\tilde{L}_i]}{K}\Big)^2\,. \tag{6}$$

The objective $\mathcal{L}_{\mathrm{len}}$ is optimized only with respect to the head parameters ϕ from this point onward. During Phase 3, the head is being trained but not yet deployed: truncation is still random (Eq. 1), so h_ϕ learns from a stable target distribution before controlling the active prefix.

Table 4. Curriculum phase schedule (in training steps $\times 10^3$) and shared hyperparameters. STROP-S variants begin the curriculum earlier to match their faster convergence; all other settings are identical across variants and datasets.

	STROP-S	STROP-M
Phase 1: random truncation only	0–100k	0–200k
Phase 2: + oracle targets	100k–150k	200k–250k
Phase 3: + length-head training	150k–200k	250k–300k
Phase 4: handoff ramp	200k–250k	300k–350k
Predicted lengths only	250k–500k	350k–500k
Shared curriculum hyperparameters		
Oracle: β=0.75, ρ=0.999, $L_{\min}^{\text{oracle}}$=2		
Probes: δ=2, τ=0.3, slope EMA=0.99		
Modulators: m_{compress}=0.4, m_{keep}=1.0, m_{extend}=1.3		
Truncation: $L_{\min}^{\text{trunc}}$=8, λ_{len}=1.0		

Table 5. Training objective and quantizer configuration shared across all model variants.

Component	**Configuration**
Program	K=64 tokens, 2D patch + learned prefix positional encoding
Quantizer	N=1024, d_c=16, ℓ_2-normalized, EMA (τ=0.95), β=1.0
Decoder	FastGAN, trained with stop-gradient on interpreter output
$\mathcal{L}_{\text{lat}}$	cosine + MSE vs. frozen DINOv3 patches, λ_{lat}=1.0
$\mathcal{L}_{\text{commit}}$	commitment loss, λ_{q}=1.0
$\mathcal{L}_{\text{div}}$	diversity regularizer, λ_{div}=0.3, 40k-step warmup

Phase 4: Handoff to predicted lengths. Let $s^{(4)}_{\text{ramp,start}}$ and $s^{(4)}_{\text{ramp,end}}$ be the first and last steps of the handoff ramp. We linearly ramp $a_s = \text{clip}((s - s^{(4)}_{\text{start}})/(s^{(4)}_{\text{end}} - s^{(4)}_{\text{start}}), 0, 1)$ and truncate the program by sampling based on the predicted length $\hat{L}_i$ as

$$L_i = \begin{cases} \text{clip}(\text{round}(\hat{L}_i), 1, K) & \text{w.p. } a_s, \\ L_i^{\text{rand}} & \text{w.p. } 1 - a_s, \end{cases} \tag{7}$$

Thus, Phase 4 gradually replaces random truncations with predicted ones, thereby avoiding the covariate shift of an abrupt switch. After $s \geq s^{(4)}_{\text{end}}$, the ramp has completed: all remaining training steps use predicted truncation lengths, while the length head continues to be updated by $\mathcal{L}_{\text{len}}$. The head continues to be trained with $\mathcal{L}_{\text{len}}$ throughout this phase.

At inference, no sampling or length loss is used: $\hat{L}_i$ is rounded and clipped to $[1, K]$, and the program is hard-truncated at that length before decoding.

Table 4 lists the phase schedule used for all main variants. The only difference between STROP-S and STROP-M is the onset of Phase 2: smaller generators converge faster during random truncation, so the curriculum begins at 100k steps rather than 200k. Each subsequent phase occupies a 50k-step window. All oracle, probe, and modulator hyperparameters are shared.

B. Architectural details

Table 6 lists the STROP variants used in the main experiments, including backbone, generator–interpreter size, parameter count, and single-H100 training time.

Table 6. Model variants trained on both COCO and ImageNet. The interpreter mirrors the generator's architecture. Training time is wall-clock on a single NVIDIA H100 (ImageNet).

Encoder	Model	d_{model}	Layers	FFN	Trainable (M)	Total (M)	Time (h)
DINO-S	STROP-S	256	8	1024	13.5	35.1	27
DINO-S	STROP-M	512	10	2048	64.3	85.9	45
DINO-B	STROP-S	256	8	1024	14.2	99.8	32
DINO-B	STROP-M	512	10	2048	65.1	150.8	50
DINO-L	STROP-M	512	10	2048	65.6	368.8	66

C. Alignment

Table 7 expands Table 1 with per-dataset latent alignment and codebook statistics.

Table 7. **Latent alignment and codebook utilization across evaluation datasets.** Cosine similarity, R^2, and RMSE measure agreement between the interpreter's predicted patch latents and frozen DINO teacher patch features. RMSE is computed per patch-feature scalar. CB% is the fraction of the 1024-entry codebook used at least once by active program tokens; Eff. CB% is the entropy-effective codebook utilization, $100 \cdot \exp(H(\text{code}))/1024$. Perp. denotes $\exp(H(\text{code}))$. Variant denotes Encoder/STROP.

		Trained on COCO					Trained on ImageNet				
Variant	Dataset	Cos↑	R^2 ↑	RMSE↓	CB%↑	Eff.%↑	Cos↑	R^2 ↑	RMSE↓	CB%↑	Eff.%↑
S/STROP-S	VOC	0.859	0.742	0.196	57.0	49.7	0.863	0.750	0.193	54.3	48.5
	Stuff-27	0.858	0.742	0.193	61.9	50.2	0.854	0.734	0.196	63.9	49.1
	ADE20K	0.854	0.731	0.195	58.7	45.6	0.854	0.731	0.195	58.4	47.4
	Citysc.	0.849	0.722	0.199	50.6	35.5	0.828	0.685	0.212	53.4	33.8
	NYUv2	0.853	0.731	0.195	52.1	42.6	0.848	0.721	0.198	51.1	43.8
S/STROP-M	VOC	0.841	0.709	0.208	98.6	81.0	0.865	0.753	0.192	98.7	80.0
	Stuff-27	0.839	0.707	0.205	99.7	84.4	0.854	0.734	0.196	100.0	82.5
	ADE20K	0.835	0.696	0.207	99.5	78.2	0.857	0.736	0.193	99.3	73.4
	Citysc.	0.834	0.693	0.209	95.6	68.5	0.834	0.694	0.209	94.4	49.8
	NYUv2	0.825	0.679	0.212	96.5	81.0	0.844	0.716	0.200	97.9	74.4
B/STROP-S	VOC	0.804	0.650	0.245	54.6	47.4	0.782	0.614	0.257	78.2	67.4
	Stuff-27	0.810	0.661	0.240	57.5	48.1	0.768	0.594	0.263	85.0	68.4
	ADE20K	0.792	0.629	0.250	58.1	41.9	0.761	0.581	0.265	79.3	64.4
	Citysc.	0.799	0.639	0.252	46.0	30.3	0.744	0.551	0.281	70.3	39.7
	NYUv2	0.798	0.637	0.243	49.8	38.1	0.759	0.576	0.263	73.3	60.2
B/STROP-M	VOC	0.808	0.656	0.243	94.8	81.2	0.819	0.674	0.236	92.2	82.3
	Stuff-27	0.810	0.661	0.240	98.7	85.2	0.805	0.654	0.242	96.9	82.8
	ADE20K	0.794	0.631	0.249	96.5	82.8	0.803	0.647	0.244	92.7	80.3
	Citysc.	0.797	0.633	0.254	91.1	64.0	0.780	0.605	0.263	85.5	59.5
	NYUv2	0.794	0.630	0.245	92.8	80.3	0.796	0.636	0.243	88.8	71.4
L/STROP-M	VOC	0.745	0.557	0.196	95.6	86.8	0.759	0.582	0.190	93.5	80.2
	Stuff-27	0.748	0.565	0.193	98.5	87.5	0.739	0.554	0.195	95.3	81.4
	ADE20K	0.714	0.510	0.198	97.2	87.0	0.723	0.528	0.195	93.5	76.8
	Citysc.	0.713	0.505	0.204	93.9	64.3	0.688	0.467	0.211	88.6	58.2
	NYUv2	0.703	0.491	0.201	94.3	83.1	0.708	0.504	0.198	90.1	68.0

D. Pairwise Token Synergies Align with Semantic Regions

Pairwise interactions provide another view of the structure learned by the discrete image programs. As shown in Fig. 6, erasing two tokens jointly can produce localized changes that align with coherent semantic regions, suggesting that the learned program contains compositional handles over the scene.

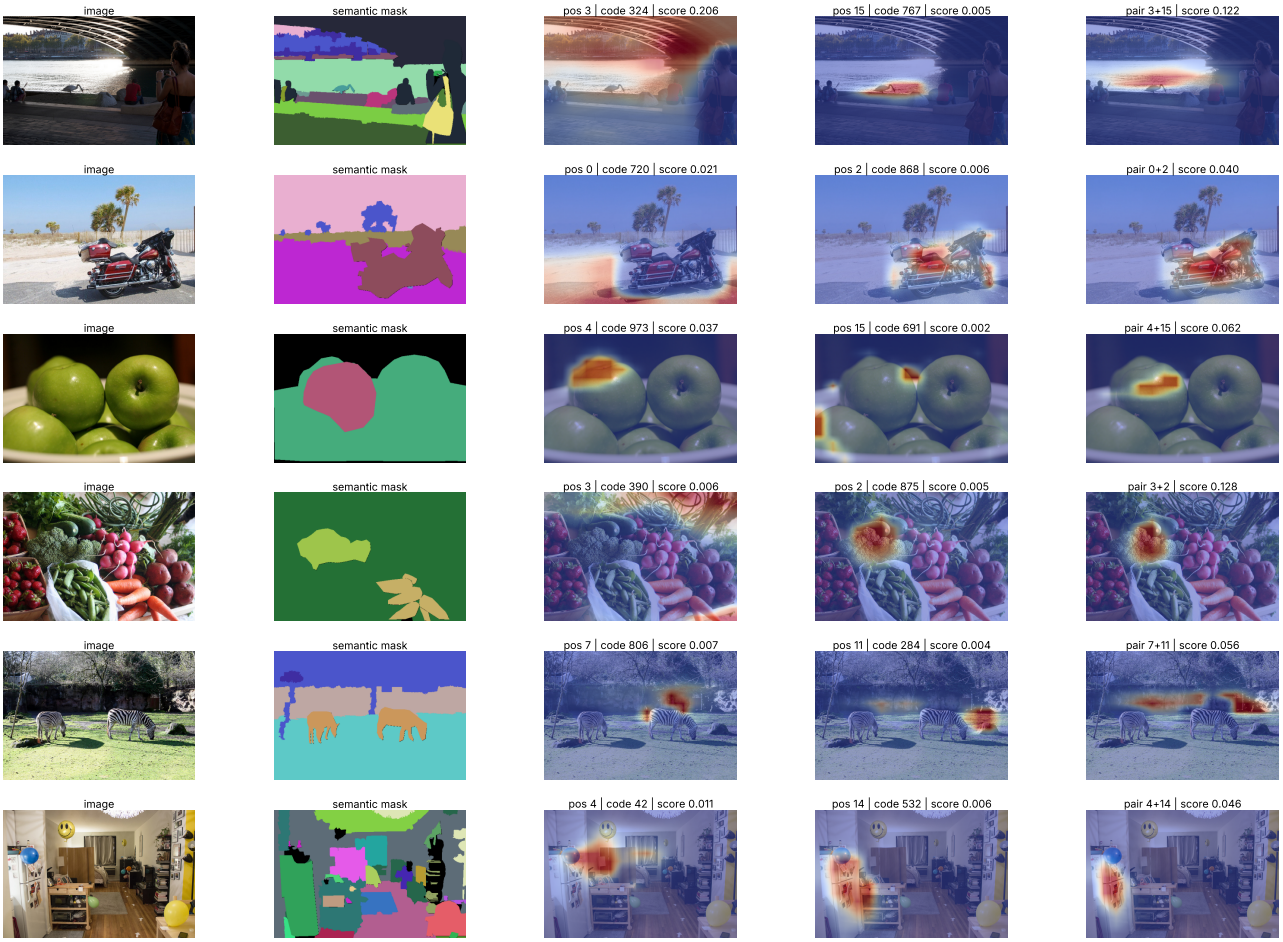


Figure 6. **Pairwise program-token synergies.** We erase pairs of discrete program tokens and reinterpret the remaining sequence, then visualize the resulting change in the DINO-aligned patch field. The selected COCO-Stuff-27 examples show that high-synergy token pairs often produce localized responses aligned with semantic mask regions, suggesting that program tokens can act as compositional semantic handles. Heatmaps are normalized per panel for visualization; pair scores are raw attribution magnitudes.

E. Downstream probe details

We report full downstream probes for teacher, interpreted-field, and program representations. Tables 8–9 give multi-label classification, Tables 10–11 give semantic segmentation, and Tables 12–13 give NYUv2 depth.

Table 8. Linear probe mAP (%, ↑) on multi-label classification. STROP trained on COCO; FlexTok and One-D-Piece are off-the-shelf adaptive tokenizers (no STROP training). Stuff-27 = COCO-Stuff-27; Mean = mean of Stuff-27 and VOC mAP. Teacher uses mean-pooled DINO patch features; latent is mean-pooled over the interpreter's spatial latent field; program is attention-pooled over the active program token vectors z_q. Best non-teacher value per column is **bolded**.

			Multi-label mAP ↑		
Encoder	Model	Repr.	Stuff-27	VOC	Mean
Adaptive tokenizer					
FlexTok	D12-D12-IN1k	latent	53.2	61.7	57.4
FlexTok	D12-D12-IN1k	program	33.9	9.5	21.7
DINOv3-S	One-D-Piece-S-256	latent	48.1	39.0	43.6
DINOv3-S	One-D-Piece-S-256	program	30.6	9.6	20.1
Teacher (frozen DINO)					
DINO-S	—	teacher	56.2	59.4	57.8
DINO-B	—	teacher	61.2	74.7	67.9
DINO-L	—	teacher	59.7	68.4	64.0
STROP (ours)					
DINO-S	STROP-S	latent	56.6	61.5	59.1
DINO-S	STROP-S	program	34.9	17.8	26.3
DINO-S	STROP-M	latent	56.5	59.4	58.0
DINO-S	STROP-M	program	30.5	13.3	21.9
DINO-B	STROP-S	latent	**61.7**	74.1	67.9
DINO-B	STROP-S	program	36.1	18.5	27.3
DINO-B	STROP-M	latent	61.6	**75.2**	**68.4**
DINO-B	STROP-M	program	31.3	14.7	23.0
DINO-L	STROP-M	latent	59.9	68.3	64.1
DINO-L	STROP-M	program	34.5	16.1	25.3

Table 9. Linear probe mAP (%, ↑) on multi-label classification. STROP trained on ImageNet; FlexTok and One-D-Piece are off-the-shelf adaptive tokenizers (no STROP training). Stuff-27 = COCO-Stuff-27; Mean = mean of Stuff-27 and VOC mAP. Teacher uses mean-pooled DINO patch features; latent is mean-pooled over the interpreter's spatial latent field; program is attention-pooled over the active program token vectors z_q. Best non-teacher value per column is **bolded**.

			Multi-label mAP ↑		
Encoder	Model	Repr.	Stuff-27	VOC	Mean
Adaptive tokenizer					
FlexTok	D12-D12-IN1k	latent	53.2	61.7	57.4
FlexTok	D12-D12-IN1k	program	33.9	9.5	21.7
DINOv3-S	One-D-Piece-S-256	latent	48.1	39.0	43.6
DINOv3-S	One-D-Piece-S-256	program	30.6	9.6	20.1
Teacher (frozen DINO)					
DINO-S	—	teacher	56.2	59.4	57.8
DINO-B	—	teacher	61.2	74.7	67.9
DINO-L	—	teacher	59.7	68.4	64.0
STROP (ours)					
DINO-S	STROP-S	latent	55.7	57.6	56.6
DINO-S	STROP-S	program	32.3	15.0	23.7
DINO-S	STROP-M	latent	56.0	59.7	57.9
DINO-S	STROP-M	program	34.1	14.3	24.2
DINO-B	STROP-S	latent	60.2	69.6	64.9
DINO-B	STROP-S	program	32.6	14.0	23.3
DINO-B	STROP-M	latent	**60.5**	**73.4**	**67.0**
DINO-B	STROP-M	program	30.1	12.6	21.3
DINO-L	STROP-M	latent	59.1	68.2	63.6
DINO-L	STROP-M	program	32.2	13.9	23.1

Table 10. Linear-probe mIoU (%, ↑) on four segmentation benchmarks. STROP is trained on COCO; FlexTok and One-D-Piece are off-the-shelf adaptive tokenizers without STROP training. Stuff-27 denotes COCO-Stuff-27. Teacher rows probe frozen DINO patch features as an upper bound. For STROP, latent probes the interpreter's spatial latent field directly, whereas program first forms a patch-level feature field by attention-pooling over the active program token vectors z_q at each spatial location, and then applies the same linear probe. Selected variants are shown; the full table is in the appendix. The best non-teacher value in each column is **bolded**.

			Segmentation mIoU ↑			
Encoder	Model	Repr.	VOC	Stuff-27	ADE20k	Citysc.
Adaptive tokenizer						
FlexTok	D12-D12-IN1k	latent	48.4	21.6	9.4	**33.3**
FlexTok	D12-D12-IN1k	program	31.9	15.6	3.3	17.3
DINOv3-S	One-D-Piece-S-256	latent	16.9	10.6	2.9	19.9
DINOv3-S	One-D-Piece-S-256	program	10.1	7.2	1.8	14.2
Teacher (frozen DINO)						
DINO-S	—	teacher	68.3	36.3	14.5	36.1
DINO-B	—	teacher	78.4	44.6	24.9	55.4
DINO-L	—	teacher	76.2	44.9	26.3	60.1
STROP (ours)						
DINO-S	STROP-S	latent	58.9	32.5	9.6	23.9
DINO-S	STROP-S	program	38.7	20.0	6.6	24.2
DINO-S	STROP-M	latent	52.0	29.5	7.7	17.6
DINO-S	STROP-M	program	35.5	16.6	4.1	18.4
DINO-B	STROP-S	latent	**67.5**	**39.2**	**15.4**	30.4
DINO-B	STROP-S	program	48.4	19.5	7.9	28.3
DINO-B	STROP-M	latent	61.9	36.3	13.4	25.9
DINO-B	STROP-M	program	43.9	18.0	6.3	22.6
DINO-L	STROP-M	latent	59.7	36.2	13.1	26.3
DINO-L	STROP-M	program	43.9	18.8	6.3	20.9

Table 11. Linear probe mIoU (%, ↑) on four segmentation benchmarks. STROP trained on ImageNet; FlexTok and One-D-Piece are off-the-shelf adaptive tokenizers (no STROP training). Stuff-27 = COCO-Stuff-27. Teacher rows show the frozen DINO upper bound; latent is mean-pooled over the interpreter's spatial latent field; program is attention-pooled over the active program token vectors z_q. Best non-teacher value per column is **bolded**.

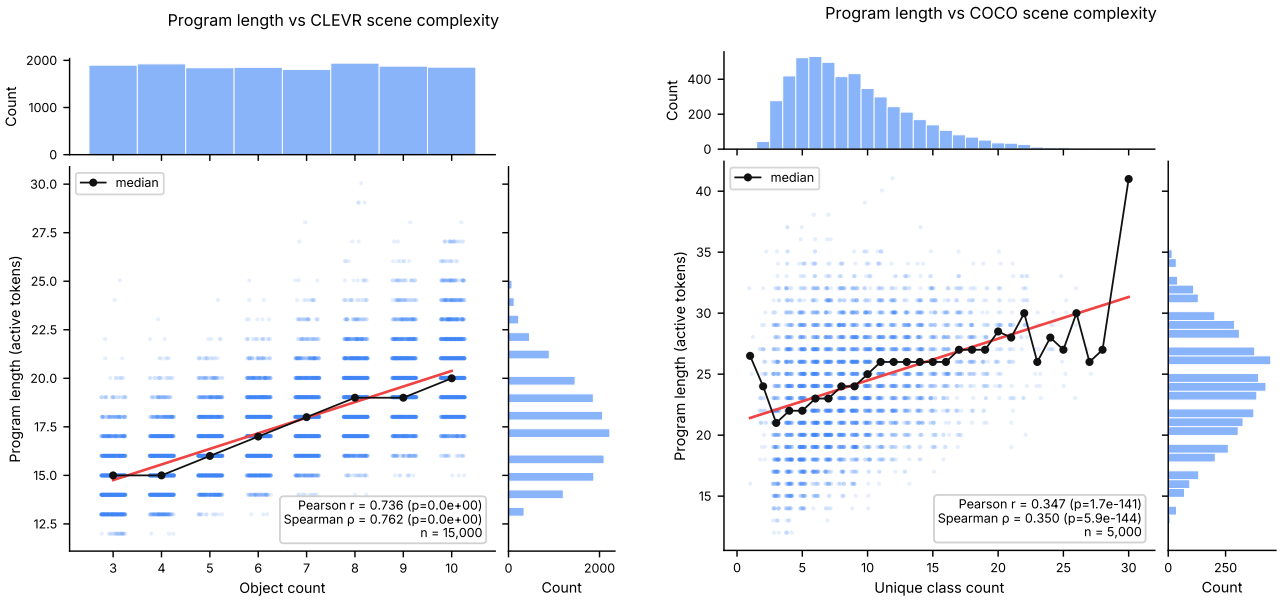
			Segmentation mIoU ↑			
Encoder	Model	Repr.	VOC	Stuff-27	ADE20k	Citysc.
Adaptive tokenizer						
FlexTok	D12-D12-IN1k	latent	48.4	21.6	9.4	**33.3**
FlexTok	D12-D12-IN1k	program	31.9	15.6	3.3	17.3
DINOv3-S	One-D-Piece-S-256	latent	16.9	10.6	2.9	19.9
DINOv3-S	One-D-Piece-S-256	program	10.1	7.2	1.8	14.2
Teacher (frozen DINO)						
DINO-S	—	teacher	68.3	36.3	14.5	36.1
DINO-B	—	teacher	78.4	44.6	24.9	55.4
DINO-L	—	teacher	76.2	44.9	26.3	60.1
STROP (ours)						
DINO-S	STROP-S	latent	57.6	29.8	8.7	21.9
DINO-S	STROP-S	program	41.7	18.4	6.0	23.7
DINO-S	STROP-M	latent	56.3	28.8	8.1	18.9
DINO-S	STROP-M	program	43.3	17.2	4.8	18.9
DINO-B	STROP-S	latent	61.9	32.2	10.8	20.8
DINO-B	STROP-S	program	40.9	18.9	5.9	19.2
DINO-B	STROP-M	latent	63.8	**35.5**	**13.3**	24.2
DINO-B	STROP-M	program	44.9	19.6	6.3	21.5
DINO-L	STROP-M	latent	**64.8**	35.1	13.2	28.5
DINO-L	STROP-M	program	46.8	17.9	6.1	22.8

Table 12. Linear probe depth estimation on NYUv2. STROP trained on COCO; metrics computed over valid depth pixels. RMSE in meters; AbsRel and δ_1 in %. Teacher rows show the frozen DINO upper bound; latent uses the interpreter's spatial latent field; program uses the active program token vectors z_q. Best non-teacher value per column is **bolded**.

			NYUv2 Depth		
Encoder	Model	Repr.	RMSE ↓	AbsRel ↓	δ_1 ↑
Teacher (frozen DINO)					
DINO-S	—	teacher	0.719	16.9	73.1
DINO-B	—	teacher	0.608	14.7	78.9
DINO-L	—	teacher	0.642	15.5	76.6
STROP (ours)					
DINO-S	STROP-S	latent	0.754	18.1	71.5
DINO-S	STROP-S	program	0.819	20.6	66.5
DINO-S	STROP-M	latent	0.804	19.9	68.2
DINO-S	STROP-M	program	0.848	24.0	62.2
DINO-B	STROP-S	latent	**0.669**	**16.4**	**75.7**
DINO-B	STROP-S	program	0.830	18.2	67.5
DINO-B	STROP-M	latent	0.747	19.0	70.3
DINO-B	STROP-M	program	0.854	22.2	61.8
DINO-L	STROP-M	latent	0.727	17.8	72.4
DINO-L	STROP-M	program	0.744	18.3	71.0

Table 13. Linear probe depth estimation on NYUv2. STROP trained on ImageNet; metrics computed over valid depth pixels. RMSE in meters; AbsRel and δ_1 in %. Teacher rows show the frozen DINO upper bound; latent uses the interpreter's spatial latent field; program uses the active program token vectors z_q. Best non-teacher value per column is **bolded**.

			NYUv2 Depth		
Encoder	Model	Repr.	RMSE ↓	AbsRel ↓	δ_1 ↑
Teacher (frozen DINO)					
DINO-S	—	teacher	0.719	16.9	73.1
DINO-B	—	teacher	0.608	14.7	78.9
DINO-L	—	teacher	0.642	15.5	76.6
STROP *(ours)*					
DINO-S	STROP-S	latent	0.816	19.4	68.0
DINO-S	STROP-S	program	0.836	19.6	66.3
DINO-S	STROP-M	latent	0.790	18.5	70.5
DINO-S	STROP-M	program	0.845	21.0	65.2
DINO-B	STROP-S	latent	0.793	19.7	68.2
DINO-B	STROP-S	program	0.931	20.5	63.4
DINO-B	STROP-M	latent	0.772	18.5	71.2
DINO-B	STROP-M	program	0.903	22.8	61.0
DINO-L	STROP-M	latent	**0.755**	**17.6**	**71.8**
DINO-L	STROP-M	program	0.809	20.3	67.8



(a) CLEVR; codebook size 32.

(b) COCO; codebook size 1024.

Figure 7. Program length and scene complexity analysis on CLEVR and COCO. STROP generally produces longer programs as the number of objects increases. On COCO, program lengths are notably higher for scenes with under-represented class counts (class counts are extracted from annotations).

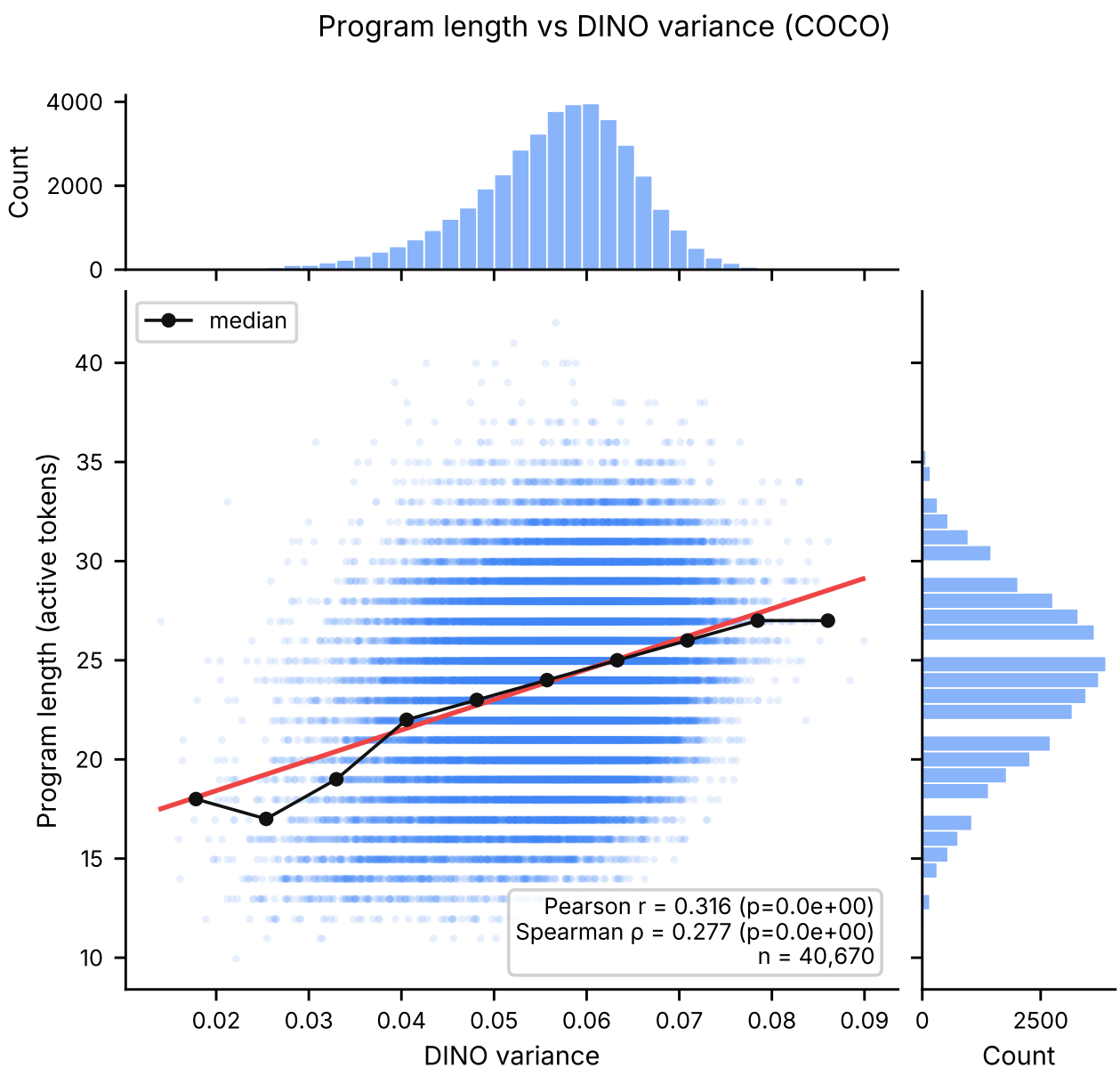


Figure 8. Program length and scene complexity analysis on COCO using variance of DINO embeddings as a proxy for scene complexity.

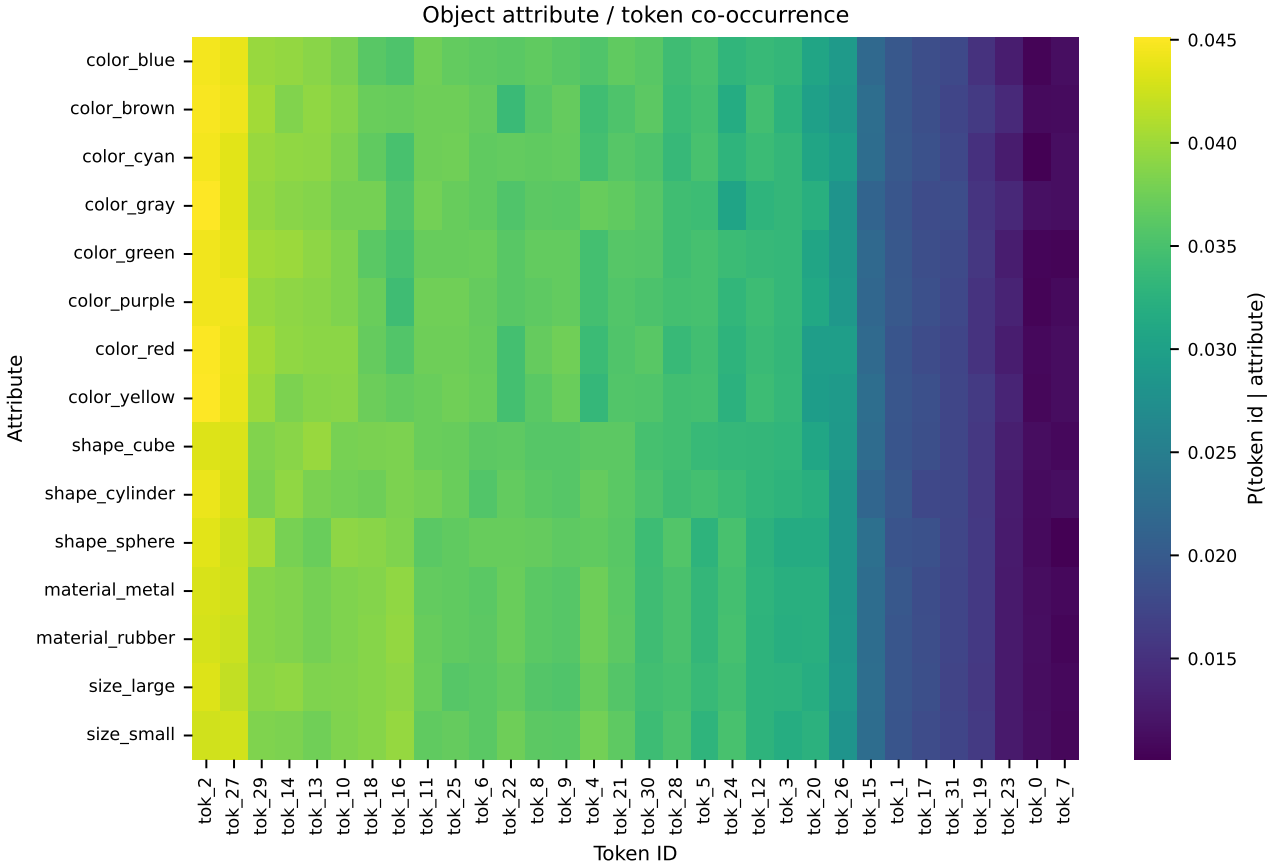


Figure 9. Contingency heatmap for CLEVR attributes and token ids; codebook size 32.

F. CLEVR

We train additional variants on CLEVR (Johnson et al., 2016) at 128×128 resolution to study codebook sizing and program structure on a controlled scene domain. All CLEVR models use a DINOv3-S encoder with STROP-S generator/interpreter (d_{model}=256, 8 layers), and K=32 program tokens. The only variable is codebook size $N \in \{32, 64, 128\}$; all other hyperparameters match the main configuration (Table 5). Training runs for 5×10^5 steps with batch size 128 on a single H100 ($\sim$27h each).

Table 14 reports results. All three codebook sizes achieve $R^2 > 0.93$, confirming that the architecture can faithfully represent CLEVR scenes through a short discrete program. The N=128 variant reaches the highest R^2 (.948) and SSIM (.787). Notably, N=64 shows a dip in alignment (R^2=.931) that is explained by its truncated codebook utilization equal to 95%, meaning nearly every code is active within the adaptive prefix and the vocabulary has little remaining capacity. At N=32 the codebook is larger relative to scene complexity, giving more headroom (91%), while at N=128 the vocabulary is comfortably under-utilized (86%).

Curriculum vs. fixed-length programs. To isolate the cost of adaptive-length training, we compare the curriculum variants ($N \in \{64, 128\}$) against matched fixed-length baselines that always use all K=32 tokens with no halting mechanism.

Table 15 shows that fixed-length programs outperform curriculum variants on R^2, with gains of .018–.030. Fixed-length training is also $\sim$10% faster in wall time (24.7–24.9h vs. 27.3h) because it avoids the halting head forward pass, oracle target computation, and phase-dependent loss modulation. This gap is expected: the curriculum solves a strictly harder problem—learning *when to stop* in addition to *what to encode*—while the fixed-length model always has access to all K tokens.

The curriculum's value is not in raw metric improvement but in the capability it unlocks: variable-length programs that allocate fewer tokens to simple scenes and more to complex ones (see Fig. 7). On CLEVR, where scene complexity varies from 3 to 10 objects, learned program lengths range from 8 to 32 tokens, enabling a $\sim2.5\times$ compression ratio on the simplest scenes with no manual tuning.

Table 14. CLEVR codebook-size variants (DINO-S / STROP-S, K=32, 128×128).

N	Cos↑	R^2↑	SSIM↑	LPIPS↓	CB%
32	.971	.943	.775	.173	73.5
64	.965	.931	.756	.187	**94.7**
128	**.973**	**.948**	**.787**	**.162**	77.8

Table 15. Curriculum vs. fixed-length training on CLEVR (DINOv3-S / STROP-S, K=32, 128×128, 500k steps). Fixed-length models use all 32 tokens; curriculum models learn an adaptive prefix via the four-phase schedule (Sec. 2.3).

N	Mode	R^2↑	CB%	Time (h)
64	Fixed	**.961**	88.6	**24.7**
64	Curriculum	.931	94.7	27.3
128	Fixed	**.966**	94.2	**24.9**
128	Curriculum	.948	77.8	27.3

G. LLM Scene Decoding from Learned Visual Codes

We evaluate whether frontier large language models can decode continuous CLEVR scene descriptions directly from discrete codebook tokens produced by a learned visual tokenizer. Given a sequence of integer codes from a DINO-S encoder (with codebook sizes $|CB| \in \{32, 64, 128\}$), the models must predict the full scene structure as a JSON string, including object count, their attributes and continuous (x, y) positions. We benchmark three frontier LLMs (Qwen3.5-397B-A17B, DeepSeek-V3.1-vLLM-2, and GPT-OSS-120B, all from Hugging Face) against a *Naive Baseline*, which uses no language model and predicts a fixed "average scene" derived entirely from the statistical distributions of the few-shot pool. This baseline establishes a lower bound measuring how much of a scene can be guessed from prior dataset statistics alone, without decoding the actual visual tokens. All models are evaluated on 15 held-out test images using a 50-shot random in-context learning setup, with one retry permitted on parse failure.

Key findings. Two observations stand out. First, the stronger frontier models (Qwen3.5 and DeepSeek) decode *spatial* structure substantially better than the naive baseline across all codebook sizes, with DeepSeek achieving the best count error (0.93 at $|CB|$=64) and the best position errors (3.15 MSE / 1.84 MAE at $|CB|$=128). Second, and more strikingly, *no model surpasses the naive baseline's attribute accuracy of* 56.8%: while the models recover "where" and "how many" (Count Err), binding tokens to discrete textual properties such as color or material remains beyond their reach without dataset priors. The smaller GPT-OSS-120B exhibits a different failure mode entirely, with count errors ($\sim$ 2.4–2.8) at or above the naive baseline despite achieving competitive position errors at $|CB|$=32, suggesting it leans more heavily on memorized layout statistics than on token-level decoding.

Attribute–token co-occurrence. Figure 9 shows the contingency table between CLEVR object attributes (color, shape, material, size) and assigned token IDs. The heatmap reveals no strong systematic correspondence—consistent with the observation that attribute identity is not yet reliably encoded in individual codebook entries. Understanding whether attribute information is distributed across token combinations rather than localized in single codes is an important direction for future work.

We document the exact prompt used to elicit scene predictions from each frontier LLM. All models were queried at temperature 0.0 with 50 random in-context examples preceding the test query. To minimize token consumption across the 50-shot context, attribute values are emitted in an abbreviated form, with a per-prompt legend instructing the model to use the short codes.

Attribute abbreviations. Object attributes are remapped to single- or two-character codes, summarized in Table 17. The same scheme is applied uniformly to the few-shot examples, the schema description in the prompt, and the expected model output, so that the model never needs to translate between long and short forms.

Prompt template. The prompt below is templated with the codebook size $|CB|$ (substituted for {`codebook_size`}), the maximum code index $|CB| - 1$ (substituted for {`codebook_max`}), the inline abbreviation legend ({`abbr_legend`}),

Table 16. Scene decoding performance of frontier LLMs compared to a statistical baseline. Parse Fails denote outputs with invalid JSON structures (max one retry). Metrics are averaged across 15 test examples. Best LLM result per metric in **bold**.

Model	$\|CB\|$	Parse Fails	Count Err ↓	Attr Acc ↑	Pos MSE ↓	Pos MAE ↓
Naive Baseline	N/A	0 / 15	2.27	0.568	3.84	2.22
GPT-OSS-120B	32	1 / 15	2.36	0.469	2.45	1.71
GPT-OSS-120B	64	3 / 15	2.83	0.536	3.26	1.87
GPT-OSS-120B	128	1 / 15	2.36	0.524	4.61	2.13
Qwen3.5-397B-A17B	32	0 / 15	1.43	0.510	3.22	1.90
Qwen3.5-397B-A17B	64	0 / 15	1.00	0.458	3.37	2.00
Qwen3.5-397B-A17B	128	0 / 15	1.53	0.505	3.17	1.91
DeepSeek-V3.1-vLLM-2	32	1 / 15	1.36	0.513	3.68	2.11
DeepSeek-V3.1-vLLM-2	64	0 / 15	**0.93**	**0.543**	3.64	1.97
DeepSeek-V3.1-vLLM-2	128	0 / 15	1.13	0.487	**3.15**	**1.84**

Table 17. Attribute abbreviation scheme used in both few-shot examples and model outputs.

Color	**Shape**	**Size**	**Material**
gray → `gr`	cube → `cu`	small → `S`	rubber → `R`
red → `r`	sphere → `sp`	large → `L`	metal → `M`
blue → `bl`	cylinder → `cy`		
green → `gn`			
brown → `br`			
purple → `pr`			
cyan → `cy`			
yellow → `y`			

and an optional reasoning suffix ({`think_suffix`}) used only for DeepSeek-V3.1 to elicit step-by-step reasoning prior to JSON emission.

Decoding prompt

```
Predict a CLEVR scene (3--10 objects) from integer codes produced by a visual
tokenizer (codebook size={codebook_size}, codes in [0,{codebook_max}]).
Output compact JSON using short keys and abbreviated values:
{abbr_legend}
Schema: {"o":[{"s":"<shape>","c":"<color>","z":"<size>",
"m":"<mat>","p":[x,y]},...]}
Position: x,y in [-3,3], 2 decimal places.
Example: {"o":[{"s":"cu","c":"r","z":"L","m":"M","p":[1.23,-0.45]}]}
JSON only. No prose.{think_suffix}
```

Reasoning suffix. For DeepSeek-V3.1 and GPT-OSS-120B, the placeholder {`think_suffix`} is replaced with the following string; for Qwen3.5 it is left empty.

DeepSeek reasoning suffix

```
Before your final answer, reason step-by-step inside <think> and </think> tags.
Then output the JSON.
```

Decoding settings. All models were queried with temperature 0.0 to encourage deterministic outputs. A single retry was permitted on JSON parse failure, after which the example was counted as a parse failure in Table 16.