SELECTIONVLM: ENABLING VLMS TO INTERACT PROGRAMMATICALLY WITH CAD

Anonymous authors
Paper under double-blind review

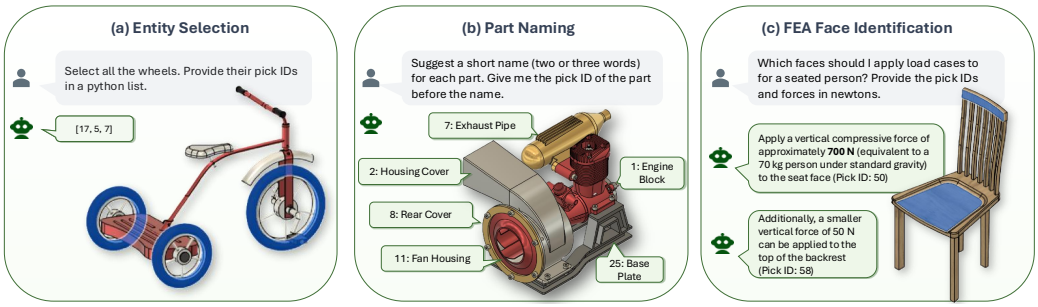


Figure 1: SelectionVLM performing three CAD tasks: (a) natural-language selection to a structured list of pick IDs, (b) assembly-wide part naming, and (c) face identification for FEA load cases. The CAD entities referenced by pick IDs are highlighted in blue overlay.

ABSTRACT

Modifying CAD geometry programmatically requires first identifying the entities to manipulate and then passing references to them to the appropriate functions in the CAD system's API. Vision-language models (VLMs), given only a canvas screenshot, can perceive these entities but cannot directly reference them, requiring the correspondence to be inferred from geometric or textual clues. We introduce SelectionVLM, a VLM augmented with a new modality that exposes information from the *selection buffer*: a per-pixel integer-valued image containing the pick IDs which 3D graphics software uses to allow interactive picking. Unlike RGB-based strategies such as object coloring, 2D bounding boxes, or numbered image markers, the selection buffer provides a direct correspondence between image pixels and CAD entities. We show that this representation enables SelectionVLM to substantially outperform RGB-only approaches on a range of entity identification tasks. Because pick IDs can also index structured per-entity metadata, supplied as part of the prompt, SelectionVLM can directly connect visual entities with their attributes, yielding further improvements on relational and spatial reasoning tasks.

1 INTRODUCTION

Selection lies at the heart of almost every Computer Aided Design (CAD) workflow. Before a CAD system can edit a model, the designer must first indicate which geometry the change applies to: a planar face to sketch on, a closed profile to extrude, a chain of edges to fillet, or the pair of faces on two components that must be brought into alignment. Only after these selections have been made can a command be invoked to act on the geometry that was picked. Selection is the primary mechanism through which designers convey their intent to the modeling kernel, and it precedes every operation that changes existing geometry.

Agentic workflows have the potential to automate many aspects of the way CAD models are created today. Language models are already being trained to drive CAD systems, issuing modeling commands and refining the result from the feedback the system returns (Ai et al., 2026; Gong et al., 2026; An et al., 2026), building assemblies whose parts must move relative to one another (Adler et al., 2026; Yu et al., 2026; Zhou et al., 2026), or operating the user interface directly (Man et al.,

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

2025). However, long-horizon work on an existing CAD design, carried out through a sequence of tool calls, needs the same primitive a designer relies on: the agent must point at a particular CAD entity (face, edge, body, or component occurrence) and obtain the *entity reference* that the CAD system's API expects.

Today this typically happens via a long and expensive detour. A vision-language model (VLM) is shown a screenshot of the modeling canvas and must work out for itself which pixels belong to which entry in the design tree. It queries the API for part names, bounding boxes, and other geometric clues, compares them against what it can see, and reasons at length to determine the reference. This is deliberate, effortful, System 2 reasoning (Kahneman, 2011; Wei et al., 2022) applied to a question that the CAD system can already answer exactly. It is slow, it fails whenever the available clues do not single out one entity from the rest, and it spends a large share of the token budget on bookkeeping rather than on design. A VLM should acquire entity references as effortlessly as a designer acquires them with a mouse click. Rather than inferring the correspondence between pixels and CAD entities via multiple tool calls, it should simply know it, so that identifying geometry becomes a perceptual act rather than a deductive one, and the references needed by the scripting API are available the moment the scene is in view.

This work asks two related questions: how can VLMs be grounded in CAD entities for reasoning over rendered models, and how can structured information already present in CAD systems be incorporated into that reasoning process? To this end, we introduce SelectionVLM, a VLM augmented with a second visual modality: a per-pixel *selection buffer* that indexes the CAD entities present in the rendered scene (Akenine-Möller et al., 2018). The selection buffer is an integer-valued image that CAD systems already render for interactive object picking, in which every pixel stores a *pick ID* that maps to an entity reference in the CAD system's data structures (see Figure 2). Figure 1 illustrates three representative uses: selecting entities, naming parts, and identifying faces for downstream simulation setup. Alongside SelectionVLM, we investigate whether off-the-shelf VLMs can be grounded in CAD entities through prompting strategies that require neither model adaptation nor architectural changes. Using controlled experiments, we investigate the performance of alternative entity-reference mechanisms and examine whether these references can also link the model to entity-specific CAD metadata, such as 3D bounding boxes and designer-provided text labels.

Our contributions are as follows:

- We introduce the selection buffer as a modality for vision language models, and present SelectionVLM, a VLM that reads it and can therefore identify CAD entities in a form that resolves directly to the entity references the CAD system's API consumes.
- We describe a data generation pipeline, architecture, and training recipe for adding this modality to open weights VLM models.
- We benchmark the modality against three practical alternatives for identifying CAD entities in a screenshot, namely distinctive coloring, 2D bounding boxes, and numbered image markers, and show that the selection-buffer modality achieves superior performance on a matched task suite.

2 Related Work

Entity-addressable visual grounding. Several lines of work have extended VLMs with additional image-like modalities, most commonly segmentation masks (Hua et al., 2024; Jahagirdar et al., 2026) and depth images (Cai et al., 2024). These approaches differ in how a region is represented: Alpha-CLIP adds a mask as a fourth image channel, and FineCaption uses this representation for region captioning at multiple granularities (Sun et al., 2024; Hua et al., 2024); others represent boxes, masks, points, or free-form shapes as object tokens that the language model attends to alongside image patches (Zhang et al., 2023; Yuan et al., 2024; You et al., 2024; Jahagirdar et al., 2026; Cao et al., 2025). Region references can also be encoded directly in the rendered image without modifying the model: red circles can redirect CLIP's attention (Shtedritski et al., 2023), while Set-of-Mark prompting overlays numeric labels on segmented regions (Yang et al., 2023). Such overlays motivate our color and marker baselines, but become difficult to use as assemblies grow: colors must be reused and markers crowd or occlude small entities. Chat-Scene is closest to our work in assigning identifier tokens to 3D object proposals (Huang et al., 2024). However, all these methods

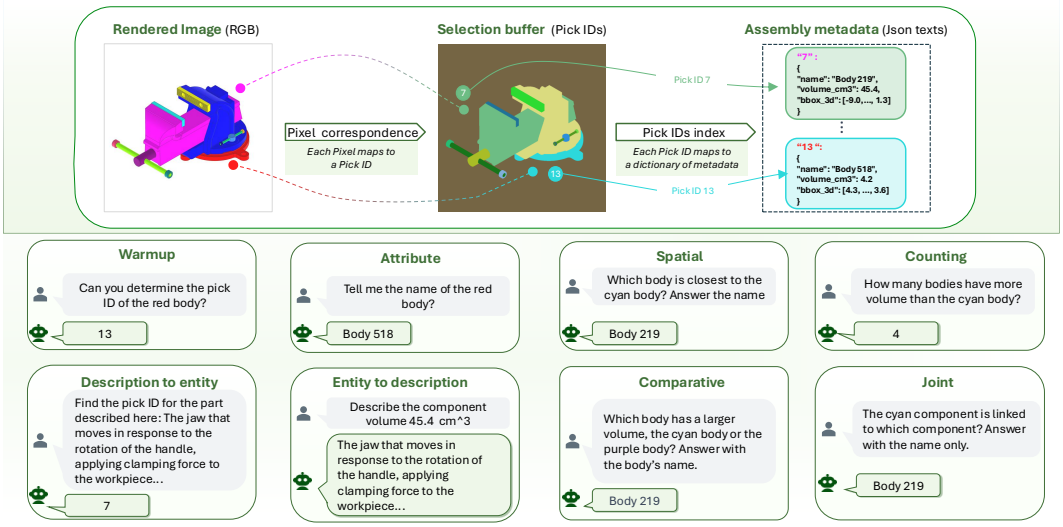


Figure 2: Top: The relationship between a rendered RGB image, its selection buffer, and the corresponding assembly metadata. Each image pixel maps to a pick ID in the selection buffer, and each pick ID indexes the metadata associated with that CAD entity. Colored callouts trace two example entities across these representations. Bottom: Example questions and ground-truth answers for the tasks described in Section 3.2.

either estimate regions from the scene or introduce references as visual cues; they do not expose the persistent CAD entity references that downstream APIs consume. SelectionVLM addresses this gap by encoding the CAD renderer's pixel-aligned selection buffer, whose identifiers directly resolve to CAD entities.

Agents acting through structured references. The problem we address has a close analogue outside CAD. Agents that operate graphical interfaces must also turn what they see into something they can act on, and that literature splits along the same line. Pixel-only agents are trained to ground a described interface element to a screen coordinate (Cheng et al., 2024; Qin et al., 2025), whereas agents given a document object model or an accessibility tree are handed the identifiers directly and act on them by name (Deng et al., 2023; Zhou et al., 2024). Structured documents are the more reliable interface, but the strongest results come from combining them with the rendered view rather than using either alone (Zheng et al., 2024). The selection buffer, together with the entity metadata it keys into, plays the role of an accessibility tree for the modeling canvas, with the distinction that it is a visual modality registered pixel for pixel against the render, so identity arrives with perception rather than as a parallel description the model must align to the image.

Language models for CAD. Recent work has language models create geometry, writing and refining modeling programs from a textual specification (Gong et al., 2026; An et al., 2026), assembling articulated models (Adler et al., 2026; Yu et al., 2026; Zhou et al., 2026), or driving CAD software through its scripting or user interface (Ai et al., 2026; Man et al., 2025). These systems generate geometry, whereas we are concerned with referring to geometry that already exists. That problem has typically been solved with graph networks, converting the boundary representation to a graph and matching entity embeddings to choose the geometry that aligns two parts into an assembly (Willis et al., 2022; Jones et al., 2021) or the faces and edges a modeling operation requires (Qi et al., 2026a;b). Boundary representations provide richer geometric and topological information than images, but models that operate on them are trained using CAD data alone, which is scarce compared with the corpora behind pretrained VLMs.

3 DATASET

Adding the selection buffer modality to a VLM requires data of a kind that does not exist in standard image collections: rendered views of real assemblies, the corresponding selection buffer information, the structured data the CAD system holds about every entity in the scene, and questions whose answers can only be reached by connecting the three. This section describes how we build such a corpus, and the suite of tasks defined on it.

We build on the Fusion 360 Gallery assembly dataset (Willis et al., 2022), a collection of 8254 assemblies authored by users of a commercial CAD system. Every assembly carries the hierarchy of nested sub-assemblies, the joints that constrain components to one another, designer-assigned names, per-body materials and densities, and recognized manufacturing features such as holes.

3.1 RENDERING VIEWS AND SELECTION BUFFERS

Each assembly is rendered from eight canonical three-quarter viewpoints, chosen to cover the corners of the bounding box so that parts occluded in one view are likely to be visible in another. Alongside every rendered image we capture the selection buffer and mapping from the pick IDs to the rendered entities. An example of a rendered image and corresponding selection buffer is shown at the top of Figure 2 and the multiple views in Appendix A.1

The renderings group entities at one of two levels of granularity: bodies or component occurrences. A body represents a solid object which is a leaf node of the assembly tree. A component occurrence may contain one or more bodies or an entire subassembly and is positioned in the global coordinate system via assembly transforms. The occurrence level groupings often segment the assembly into semantically meaningful chunks and the Fusion Gallery Assembly dataset may include designer-provided text annotations at either the body or occurrence level.

Every body or occurrence visible in a view is given a distinct display color drawn from a palette of 20 distinctive named colors. These colors allow entities to be identified both as part of the training objective and during evaluation. More details on the color pallet is provided in Appendix A.1.

In addition to providing a mechanism for the VLM to reference entities, we also want to allow the CAD system to supply its own knowledge about those entities to the VLM. Each training example therefore includes a structured record, keyed by pick ID, giving the properties of the entities in the view. These may optionally include the body or occurrence name specified by the designer, mass properties and axis-aligned bounding boxes. Short extracts from the assembly metadata are shown in Figure 2 and a full example is given in Appendix A.2

Two tasks require natural language descriptions of individual parts, which the design record does not contain. We generate them with the Qwen3-VL-32B-Instruct base model. The VLM is shown the part highlighted within its assembly context, an isolated view of the part and given whatever text the designer provided. The model is prompted to describe the part's shape and the role it plays in the assembly. These descriptions serve as the prompt in tasks that ask the model to find the part being described, and as the ground truth in tasks that ask it to describe a part it has been pointed at.

3.2 TASK SUITE

Tasks are generated automatically from the rendered views and the design record. Each includes a question, in some cases accompanied by the metadata for the full assembly, and a ground-truth answer, also derived from the CAD data. We group them into eight families, with examples in Figure 2. To avoid overfitting on specific wording, questions are expressed through a total of 210 paraphrased templates.

The families differ chiefly in how many steps of indirection separate the question from the answer. Warm-up tasks are direct: the model is told the color of an entity and asked for its pick ID, with no metadata in the prompt. These tasks are primarily used during training so that the association between the pick IDs in the buffer and the entities they denote can be learned. One-hop tasks add a single lookup, identifying an entity by color and then using its pick ID as a key, allowing one of its properties to be extracted from the assembly metadata. *Description to entity* tasks require the VLM to identify an entity via its description, then either provide its pick ID directly or use the

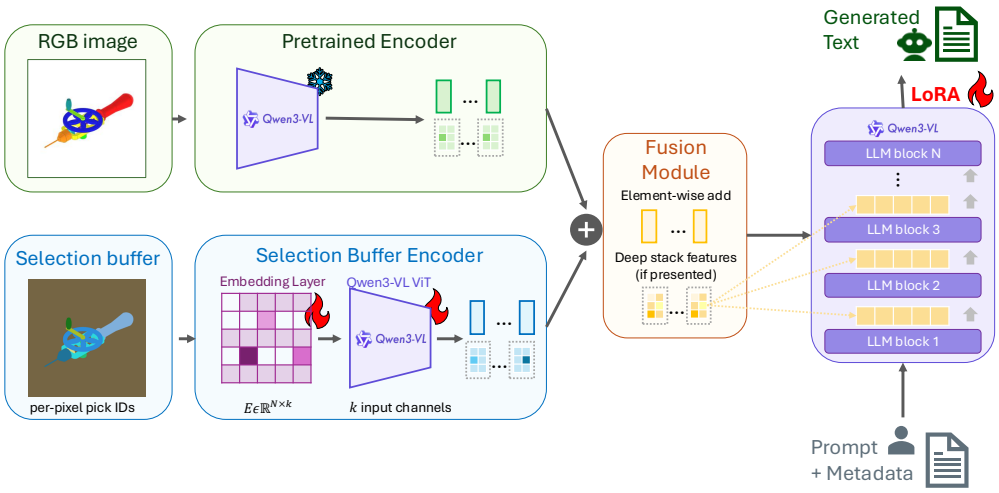


Figure 3: SelectionVLM architecture. A frozen pretrained Qwen3-VL vision transformer encodes the RGB image, while a parallel trainable encoder processes the pick IDs in the selection buffer using a learned embedding matrix $E \in \mathbb{R}^{N \times k}$ and a second vision transformer. The visual tokens and, when present, DeepStack features from the two encoders are combined by element-wise addition and passed to the LoRA-adapted Qwen3-VL language model.

metadata to determine a specified property for that entity. *Entity to description* tasks work in the other direction. The VLM is given either a pick ID or a property from the metadata and asked to provide a natural language description of the entity which possesses it. These tasks teach the VLM to connect properties in the metadata with the pick ID keys and the semantics of the objects they see in the scene. The remaining families are relational: they require the model to resolve two or more entities and then compare their masses, reason about the joint that connects them, derive their relative positions from bounding boxes or count the bodies whose mass or area property exceeds that of the given body. Answers that are a single value are scored by exact match, while the free-text descriptions from the *entity to description* tasks are scored with BLEU.

We divide the corpus by assembly, in the proportions $96\%/2\%/2\%$ for training, validation, and test (7510, 156, and 157 assemblies). Splitting at the level of the assembly rather than by task or view ensures that no assembly seen in training can appear at evaluation time, and that repeated parts within an assembly cannot leak across the boundary. A total of 92,497 images were rendered using the eight canonical views, 60,952 of which were segmented with body granularity and 31,545 with occurrence granularity. Each image allowed multiple tasks to be defined, referencing the pick IDs of different bodies and utilizing different task templates. In total 44,772,713 tasks were included, 43,011,403 in the training set 909,799 for validation and 851,511 in the test set.

4 Method

SelectionVLM augments Qwen3-VL-32B-Instruct (Yang et al., 2025) with a second vision encoder that ingests the selection buffer data, enabling the model to associate the objects it sees in the screenshot with their pick IDs. The architecture is shown in Figure 3.

Let $\mathbf{X} \in \mathbb{R}^{H \times W \times 3}$ denote the RGB rendering and $\mathbf{S} \in \{0, \ldots, N-1\}^{H \times W}$ its corresponding selection buffer, where $H = W = 1024$. Each integer entry $s_{u,v}$ of $\mathbf{S}$ is a pick ID, with 0 reserved for the background. The two inputs share the same 1024×1024 pixel grid, so $s_{u,v}$ identifies the CAD entity visible at the corresponding RGB pixel. We convert the discrete selection buffer into a dense k-channel image using a learned embedding function $\mathbf{e} : \{0, \ldots, N-1\} \to \mathbb{R}^k$. At each pixel location (u, v), the pick ID is replaced by its embedding: $\mathbf{z}_{u,v} = \mathbf{e}(s_{u,v}) \in \mathbb{R}^k$. The embedding function is parameterized by a learned matrix $\mathbf{E} \in \mathbb{R}^{N \times k}$, whose ith row is the embedding associated with pick ID i. Applying $\mathbf{e}$ at every pixel produces an embedded selection buffer $\mathbf{Z} \in \mathbb{R}^{H \times W \times k}$.

We process $\mathbf{Z}$ using a selection-buffer encoder derived from the pretrained Qwen3-VL-Instruct-32B image encoder. Because a selection buffer represents a substantially simpler visual signal than a natural image, we truncate the encoder to a configurable number of transformer blocks, treating its depth as a hyperparameter to reduce memory and computation. We also replace the input convolution so that it accepts the k channels of $\mathbf{Z}$ rather than the three channels of an RGB image. These design choices are validated with ablation studies in Appendix C.

The retained transformer blocks, positional embeddings, token merger, and DeepStack mergers (Meng et al., 2024) are initialized from the pretrained image encoder. The selection-buffer encoder consequently produces merged visual tokens and DeepStack features with the same spatial organization and dimensionality as those produced by the image encoder.

We combine the two streams by element-wise addition, summing their merged visual tokens and, when both branches produce them, the corresponding DeepStack features. The combined visual tokens then take their usual place among the text tokens in the language model's input, and the combined DeepStack features are injected into its early layers, exactly as in the base model. More details are given in Appendix B.3

5 TRAINING

We keep the pretrained image vision transformer fixed and adapt the language model with low-rank adapters (LoRA) (Hu et al., 2022) of rank $r = 16$, applied to the attention and feed-forward projections of every transformer block. The selection buffer encoder is trained in full. Rather than starting it from random weights, we initialize it from the pretrained image encoder: the two encoders share an architecture, so the transformer blocks, token merger, positional embeddings, and DeepStack mergers can all be copied across. Only the new input convolution and the embedding matrix $\mathbf{E}$, which have no counterpart in the image encoder, are learned from scratch. In total only 322M parameters are optimized, 134M in the LoRA adapters and 188M in the selection buffer encoder when it is truncated to a depth of 8 transformer blocks. Thinking mode is disabled in the chat template and no `<think>...</think>` tokens are used during training.

The tasks seen at training time are introduced via a curriculum, with *warmup* tasks, used initially and *attribute*, *description to entity* and *entity to description* tasks introduced after 625 steps. Other tasks onset at 2,260 steps. Gradient accumulation of 4 and a batch size of 8 was used. We train the model for 4 epochs (5,684 steps) in bf16 with DeepSpeed ZeRO-3 (Rajbhandari et al., 2020) across 8 H100 GPUs. A full training run takes approximately 79 hours on this hardware.

6 BASELINE METHODS

We compare our selection-buffer encoder against three alternative methods for referencing CAD entities that require no additional visual modality: color-keyed references, bounding boxes, and numbered markers. In each case, we replace the integer pick IDs used by our method with the corresponding reference representation:

color-keyed **baseline:** the integer pick IDs are replaced with distinctive color names (e.g. `sky_blue`, `dark_green`) that the model must infer from the RGB render.

bounding-box **baseline:** the pick IDs are replaced by 2D boxes $[x_1, y_1, x_2, y_2]$ in the 1024×1024 image space coordinate system. As the list of candidate bounding boxes is provided as the keys of the assembly metadata, the VLM only needs to choose between these alternatives, rather than outputting a 2D bounding box exactly.

set-of-mark **baseline:** small numbered markers are drawn at each entity's centroid, following the marker style from (Yang et al., 2023); pick IDs are the matching integers, which appear both in the image and in the metadata. See Appendix A.2.

When evaluating the *"description to entity"* tasks the target entity is identified by its natural-language description, so entity-resolution difficulty is matched across methods and only the pick ID representation differs. For other task families the bounding-box and set-of-mark baselines define the target using display color. The color-keyed baseline uses the 2D bounding-box to define the target since the color is already used as the reference representation. Tasks are built once and me-

Table 1: Accuracy on the *description to entity* task. The three baselines are evaluated with the pre-trained base model; the color-keyed baseline is additionally shown with LoRA fine-tuning and with LoRA plus vision tower fine-tuning.

Method	Accuracy (%)
Baselines	
Bounding-box	57.17
Set-of-mark	63.31
Color-keyed	67.41
Fine-tuned baselines	
Color-keyed + LLM LoRA	61.08
Color-keyed + LLM LoRA + vision tower	52.14
SelectionVLM (ours)	**74.12**

chanically re-keyed for each baseline, so every version contains the same questions, entities, views, and metadata.

7 Experiments and Results

7.1 Identifying objects from text descriptions

Our first experiment evaluates the *"description to entity"* task, which measures how accurately the model can resolve a natural-language description of a body or occurrence to the corresponding pick ID, and then use that key to look up the requested property in the assembly metadata. It is the most direct test of whether the model can connect what it sees to the pick IDs and CAD system data structures. We compare SelectionVLM against the three baselines methods described in Section 6, evaluated using the pretrained base model. Table 1 reports accuracy for each method. SelectionVLM outperforms all baseline methods by more than 6%, demonstrating the potential of using the additional modality.

Since the *color-keyed* baseline is the strongest of the three, we additionally fine-tune it under two settings: LoRA adapters trained with the same recipe used for SelectionVLM, and the same LoRA training with the vision tower additionally unfrozen. This fine tuning ensures the LLM is given the opportunity to associate the RGB values of the color palette with the color names used as pick IDs. Surprisingly fine tuning degrades performance on the *color-keyed* baseline task, which we believe is due to distribution shifts due to the brightly colored CAD renderings and VLM generated part descriptions.

7.2 Metadata and relational tasks

We next evaluate SelectionVLM on the remaining task families from Section 3.2, *attribute*, *comparative*, *joint*, *spatial*, *counting* and *entity to description*, using the same baselines from Section 6. We additionally include a SelectionVLM checkpoint trained only on *warmup*, *attribute*, and *entity to description* tasks, to test whether performance on relational families depends on having seen them during training. Figure 4 reports exact-match accuracy for all families except *entity to description*, where we plot BLEU score multiplied by eight so that the bars share a comparable vertical scale.

SelectionVLM leads on every task family. The margin over the strongest baseline ranges from 17 percentage points on *spatial* and *joint* tasks to nearly 30 points on the *attribute* task, where the model must resolve an indicated entity and then read a specific field from the assembly metadata. On *comparative* and *counting* tasks, which require reasoning over two or more entities, SelectionVLM reaches 50% and 70% accuracy respectively, roughly 19 and 21 points above the best baseline. Even on *entity to description*, where the output is free text rather than a metadata lookup, SelectionVLM reaches BLEU 0.10 compared with 0.05 for the bounding-box baseline.

The limited-task checkpoint matches the full model on the *attribute* task (96%) and exceeds it on *entity to description* (0.12 vs. 0.10 BLEU), both of which were present in its training mix. On every

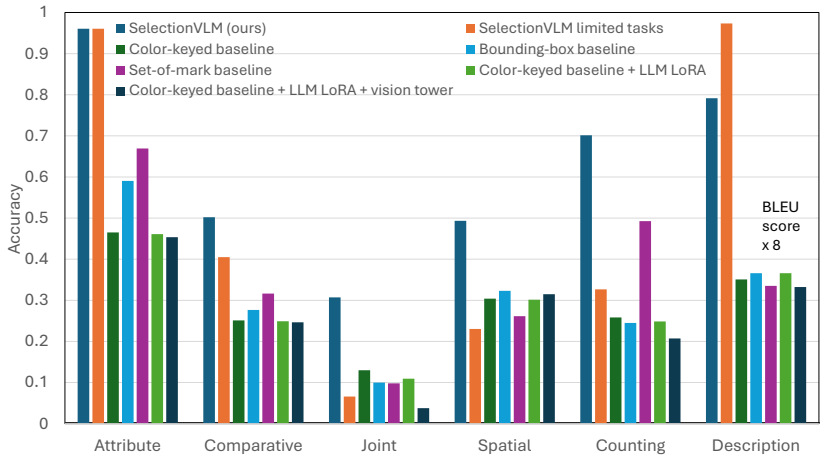


Figure 4: Performance on the task families not covered in Table 1. Bars show exact-match accuracy except for *entity to description*, where the score is BLEU multiplied by eight. We compare the fully trained SelectionVLM model, a checkpoint trained only on *warmup*, *attribute*, and *entity to description* tasks, and the three baselines from Section 6; the color-keyed baseline is additionally shown with LoRA fine-tuning and with LoRA plus vision-tower fine-tuning.

relational family it was not trained on, however, accuracy drops sharply: for example, *joint* falls from 31% to 7% and *counting* from 70% to 33%. This gap suggests that SelectionVLM does not merely memorize metadata fields seen during training, but learns to use pick IDs and the structured assembly record to answer questions that require multi-step lookup and comparison.

Among the baselines, the *set-of-mark* variant is the strongest on *counting* (49%), outperforming the *color-keyed* and *bounding-box* alternatives by more than 20 points. This is consistent with the original Set-of-Mark work (Yang et al., 2023), which reports that numeric markers drawn on the image help VLMs with enumeration. Even there, however, SelectionVLM retains a 21-point lead, indicating that the selection buffer provides a stronger grounding signal than centroid markers alone.

In Appendix C we perform ablation studies to validate some of the design choices made when creating the SelectionVLM architecture. We find that the width of the pick-ID embeddings supplied to the selection buffer encoder can become a bottleneck as the number of entities to distinguish increases. Expanding the channels from $k = 3$ to $k = 32$ improves performance by more than 7% in scenes containing 81–90 bodies. We also find that reducing the selection buffer encoder depth to eight layers has little effect on performance, while decreasing its trainable parameter count from 612M to 188M.

8 Limitations and Future Work

The quantitative results in the previous section show that SelectionVLM outperforms the baseline methods on a suite of entity-identification tasks. These benchmarks, however, do not capture the full range of reasoning required in practical CAD workflows. We therefore use rendered assemblies from the Fusion 360 Gallery dataset (Willis et al., 2022), together with their selection buffers, to qualitatively probe SelectionVLM on three more challenging scenarios. Figure 5 considers: selecting a suitable mating body for an assembly-modelling task of the kind studied in (Willis et al., 2022) and (Jones et al., 2021); reasoning about the global coordinate system when axis-aligned bounding boxes are provided as per-entity metadata; and explaining the operation of a mechanism.

On the assembly task, SelectionVLM can reason about the placement of parts in simple assemblies, but automated joint creation requires selection of individual faces or edges. As the number of entities in the selection buffer increases, the robustness with which the correct pick IDs can be found degrades, as shown in Appendix C. In future work we plan to investigate techniques for improving performance with large numbers of small entities.

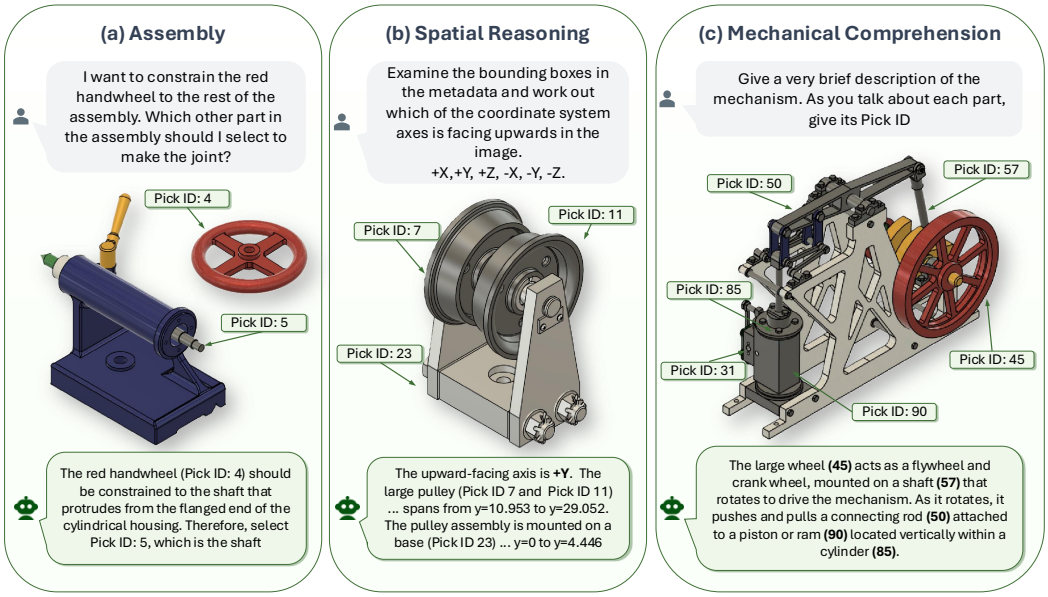


Figure 5: Qualitative examples investigating the ability of SelectionVLM to perform downstream tasks. From left to right: (a) selecting a mating body as part of an assembly modelling workflow, (b) using per-entity bounding boxes to gain understanding of the global coordinate system, and (c) open-ended mechanical comprehension of a beam engine.

Providing the 3D bounding boxes of bodies in the assembly metadata allows some understanding of the global coordinate system, but SelectionVLM often prefers to describe objects as being on the left or right hand side of the image rather than giving relative positions along the X, Y and Z axes. Future work will experiment with utilizing the depth buffer to embed global coordinates at every pixel, in the hope of improving the VLM's ability to generate 3D points and vectors suitable for manipulating the scene geometry.

Mechanical comprehension remains difficult when design-tree text is withheld, as in the example shown. SelectionVLM failed to recognize that the mechanism was intended to be a beam engine, instead describing it as a press, and misidentified the rod connecting the crank to the rocker beam. We expect that agentic workflows may help in future by allowing SelectionVLM to zoom in on parts of interest, inspect their geometry more closely, and refine its analysis iteratively.

9 Conclusions

In this paper, we introduced SelectionVLM, a vision-language model that incorporates the selection buffer as a second visual modality. SelectionVLM outperforms color-keyed, bounding-box, and Set-of-Mark alternatives across the evaluated task suite, while using pick IDs as keys into entity-specific CAD metadata for attribute, relational, and spatial reasoning. Our limited-task experiment shows that relational reasoning requires explicit training on the corresponding task families. The architecture ablations show that an 8-block selection-buffer encoder performs comparably to the full-depth encoder while reducing the trainable parameters in that branch from 612M to 188M; they also suggest that the wider $k = 32$ embedding improves robustness when many entities are present. Together, these results establish the selection buffer as an effective modality for grounding VLMs in the structured entity references maintained by CAD systems. This capability provides a foundation for future agentic workflows that reason about existing CAD models and act on them through CAD system APIs.

REFERENCES

Mitch Adler, Matthew Russo, and Michael Cafarella. Agent-aided design for dynamic CAD models. In *ACM Conference on AI and Agentic Systems (CAIS)*, 2026. doi: 10.1145/3786335.3813198.

Jiaxin Ai, Tao Hu, Xuemeng Yang, Shu Zou, Hairong Zhang, Daocheng Fu, Yu Yang, Hongbin Zhou, Nianchen Deng, Pinlong Cai, Zhongyuan Wang, Botian Shi, Kaipeng Zhang, and Licheng Wen. Comact: Reframing professional software manipulation via com-as-action paradigm. *CoRR*, abs/2606.13239, June 2026. URL http://dblp.uni-trier.de/db/journals/corr/corr2606.html#abs-2606-13239.

Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michał Iwanicki, and Sébastien Hillaire. *Real-Time Rendering*. A K Peters/CRC Press, 4th edition, 2018.

Jiyuan An, Jiachen Zhao, Fan Chen, Liner Yang, Zhenghao Liu, Hongyan Wang, Weihua An, Meishan Zhang, and Erhong Yang. PR-CAD: Progressive refinement for unified controllable and faithful text-to-CAD generation with large language models. *arXiv preprint arXiv:2604.19773*, 2026.

Wenxiao Cai, Iaroslav Ponomarenko, Jianhao Yuan, Xiaoqi Li, Wankou Yang, Hao Dong, and Bo Zhao. SpatialBot: Precise spatial understanding with vision language models. *arXiv preprint arXiv:2406.13642*, 2024.

Shengcao Cao, Zijun Wei, Jason Kuen, Kangning Liu, Lingzhi Zhang, Jiuxiang Gu, HyunJoon Jung, Liang-Yan Gui, and Yu-Xiong Wang. Refer to any segmentation mask group with vision-language prompts. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023.

Yifei Gong, Xing Wu, Wenda Liu, and Kang Tu. ToolCAD: Exploring tool-using large language models in text-to-CAD generation with reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23161–23188, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Hang Hua, Qing Liu, Lingzhi Zhang, Jing Shi, Zhifei Zhang, Yilin Wang, Jianming Zhang, and Jiebo Luo. FineCaption: Compositional image captioning focusing on wherever you want at any granularity. *arXiv preprint arXiv:2411.15411*, 2024.

Haifeng Huang, Yilun Chen, Zehan Wang, Rongjie Huang, Runsen Xu, Tai Wang, Luping Liu, Xize Cheng, Yang Zhao, Jiangmiao Pang, and Zhou Zhao. Chat-Scene: Bridging 3D scene and large language models with object identifiers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Soumya Jahagirdar, Walid Bousselham, Anna Kukleva, and Hilde Kuehne. When LLaVA meets objects: Token composition for vision-language models. *arXiv preprint arXiv:2602.04864*, 2026.

Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G. Kim, and Adriana Schulz. AutoMate: A dataset and learning approach for automatic mating of CAD assemblies. *ACM Transactions on Graphics*, 40(6), 2021.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

Brandon Man, Ghadi Nehme, Md Ferdous Alam, and Faez Ahmed. VideoCAD: A dataset and model for learning long-horizon 3D CAD UI interactions from video. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2025.

Lingchen Meng, Jianwei Yang, Rui Tian, Xiyang Dai, Zuxuan Wu, Jianfeng Gao, and Yu-Gang Jiang. DeepStack: Deeply stacking visual tokens is surprisingly simple and effective for LMMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 23464–23487, 2024.

Dacheng Qi, Chenyu Wang, Jingwei Xu, Tianzhe Chu, Zibo Zhao, Wen Liu, Wenrui Ding, Yi Ma, and Shenghua Gao. Pointer-CAD: Unifying B-Rep and command sequences via pointer-based edges & faces selection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026a.

Dacheng Qi, Chenyu Wang, Jingwei Xu, Yi Ma, and Shenghua Gao. Pointer-CAD v2: Plan-then-construct CAD generation with dimension-aware parametric precision. In *European Conference on Computer Vision (ECCV)*, 2026b.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, and Kuanye Li. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

Aleksandar Shtedritski, Christian Rupprecht, and Andrea Vedaldi. What does CLIP know about a red circle? visual prompt engineering for VLMs. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Zeyi Sun, Ye Fang, Tong Wu, Pan Zhang, Yuhang Zang, Shu Kong, Yuanjun Xiong, Dahua Lin, and Jiaqi Wang. Alpha-CLIP: A CLIP model focusing on wherever you want. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Karl D. D. Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. JoinABLe: Learning bottom-up assembly of parametric CAD joints. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023.

Haoxuan You, Haotian Zhang, Zhe Gan, Xianzhi Du, Bowen Zhang, Zirui Wang, Liangliang Cao, Shih-Fu Chang, and Yinfei Yang. Ferret: Refer and ground anything anywhere at any granularity. In *International Conference on Learning Representations (ICLR)*, 2024.

Shuiping Yu, Yandong Guan, Zhan Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. Articad: Articulated cad assembly design via multi-agent code generation. *ArXiv*, abs/2604.10992, 2026. URL https://api.semanticscholar.org/CorpusID:287432372.

Yuqian Yuan, Wentong Li, Jian Liu, Dongqi Tang, Xinjie Luo, Chi Qin, Lei Zhang, and Jianke Zhu. Osprey: Pixel understanding with visual instruction tuning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 28202–28211, 2024.

Shilong Zhang, Peize Sun, Shoufa Chen, Min Xiao, Wenqi Shao, Wenwei Zhang, Kai Chen, and Ping Luo. GPT4RoI: Instruction tuning large language model on region-of-interest. *arXiv preprint arXiv:2307.03601*, 2023.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning (ICML)*, 2024.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation. *arXiv preprint arXiv:2605.15187*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

A Dataset details

A.1 Colored multi-view assembly renderings

Each assembly is rendered from eight canonical views, with each body or component occurrence show in a distinctive color, as shown in Figure 6. Colors use widely recognizable names, which serve to identify objects during both training and evaluation. Table 2 lists the 20-color palette used.

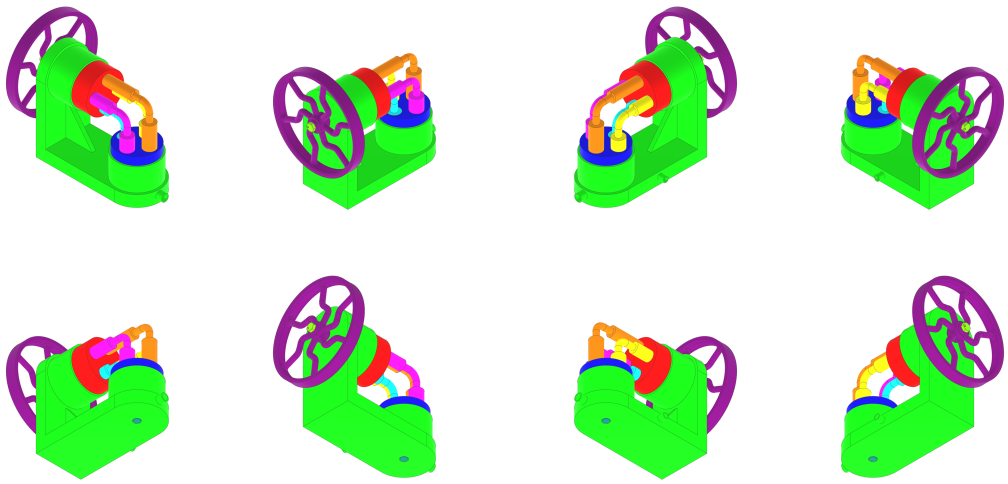


Figure 6: Multi-view images of the assembly with bodies rendered in vivid colors.

Table 2: Distinctive color palette for rendering bodies and occurrences. Colors are assigned randomly and are chosen to be as distinctive as possible. In assemblies with more than 20 visible entities, colors are reused.

#	Name	R	G	B
1	red	255	0	0
2	green	0	255	0
3	blue	0	0	255
4	yellow	255	255	0
5	cyan	0	255	255
6	magenta	255	0	255
7	orange	255	128	0
8	purple	128	0	128
9	lime	128	255	0
10	teal	0	128	128
11	rose	230	51	102
12	sky_blue	51	128	230
13	spring_green	0	204	102
14	amber	255	191	0
15	dark_blue	0	0	139
16	dark_green	0	100	0
17	dark_red	139	0	0
18	coral	255	127	80
19	steel_blue	70	130	180
20	olive	128	128	0

A.2 Assembly metadata

As well as understanding the correspondence between objects in a scene and their pick IDs, SelectionVLM is also trained to understand assembly metadata in the json format shown in Figure 7 which is provided in the prompt. This metadata allows additional information about each entity in

the image to be provided to the VLM as additional context. The keys in the metadata are pick IDs and the values are drawn from data held by the CAD system about each object. The questions in the *attribute*, *comparative*, *spatial*, *entity to description* and *description to entity* tasks, described in Section 3.2, are designed so that answering them correctly requires objects in the image to be identified, their pick IDs established and properties in the metadata under the pick ID key to be found and compared.

By training SelectionVLM to understand the connection between the image, pick IDs and the assembly metadata, we enable downstream workflows which can exploit this information. Figure 5 (b) shows that providing the 3d bounding boxes of each entity in the scene allows SelectionVLM to understand which of the coordinate system axes points upwards in the image. When text from the design is provided, we observe a big improvement in mechanical comprehension. For the example shown in Figure 5 (c), SelectionVLM incorrectly identifies the assembly as a press when the design text is withheld. When this text is provided as part of the metadata, the assembly is correctly described as a "beam engine mechanism, likely a steam engine or a piston-driven reciprocating engine with a rocking beam linkage system". The design text included "top cylinder head v2" (pick ID 85), "block 2nd iter v7" (pick ID 90) and "steam chest v3" (pick ID 31). These small contextual clues were enough to allow the entire mechanism to be understood.

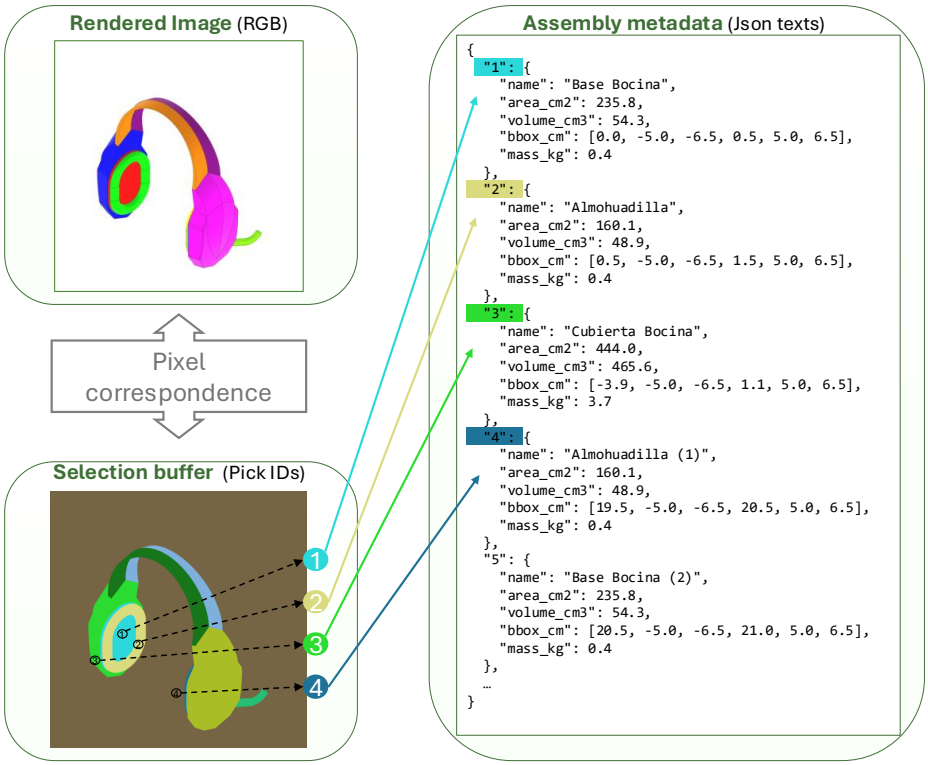


Figure 7: A CAD assembly and its corresponding assembly metadata. Assembly metadata is passed with the prompt for all task types other than *warmup* described in Section 3.2.

A.3 Set-of-Mark Renderings

The *set-of-mark* baseline described in Section 6 draws the pick IDs directly onto the image as small numbered markers. The markers are added at the centroid of each segment in the selection buffer and the marker style closely follows (Yang et al., 2023). Figure 8 shows an example of one of the images from the *set-of-mark* baseline. While many of the bodies can be identified without ambiguity, the markers have a tendency to overlap and in some cases completely occlude the object they reference.

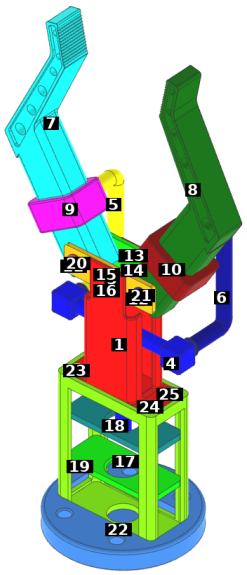


Figure 8: An image showing the numeric annotations used for the *set-of-mark* baseline

B SELECTION BUFFER ENCODER DETAILS

As described in Section 4, the selection buffer encoder is build from a trainable copy of the Qwen3-VL-Instruct-32B image encoder with minor modifications.

B.1 PRE-PROCESSING

All images and selection buffers used for training and evaluation were of resolution 1024×1024, allowing conversion to 64×64 patches of 16×16 pixels without downsampling. If different resolutions are required, the image data can be resized using the Qwen3-VL image processor (`Qwen2VLImageProcessorFast`). The selection buffer processor needs to perform this resizing independently, which can be implemented using nearest neighbor sampling of the pick IDs. As the Qwen3-VL image processor configuration defines a temporal patch size, t_p, of 2, the image pixels are duplicated in the temporal dimension. The selection buffer processor performs the same temporal duplication. The final data passed from the selection buffer processor is then an `int64` tensor of shape $(4096, 512)$ representing the 64×64 patches of $t_p \times 16 \times 16$ pick IDs per patch.

B.2 EMBEDDING

The learned embedding matrix $E \in \mathbb{R}^{N \times k}$ is then used to embed each PickID, producing a tensor of shape $(4096, 512, k)$, which is passed to the modified `Conv3d` layer at the start of the selection buffer encoder.

B.3 HOOK-BASED FUSION MECHANISM

SelectionVLM injects the selection buffer encoder's output into the base Qwen3-VL-Instruct-32B model without modifying any upstream code, as summarized in Section 4.

A wrapper module around the Qwen3-VL model registers a post-forward hook on the base model's vision encoder at construction time. The wrapper's forward function receives the batch, including the pick ID tensor passed from the processor. It runs the selection buffer encoder, producing a pooled output and a list of DeepStack features.

The base model then invokes its vision encoder on the RGB pixel tensor. When the registered post-forward hook fires, it adds the selection buffer embeddings to the vision encoder's output, element-

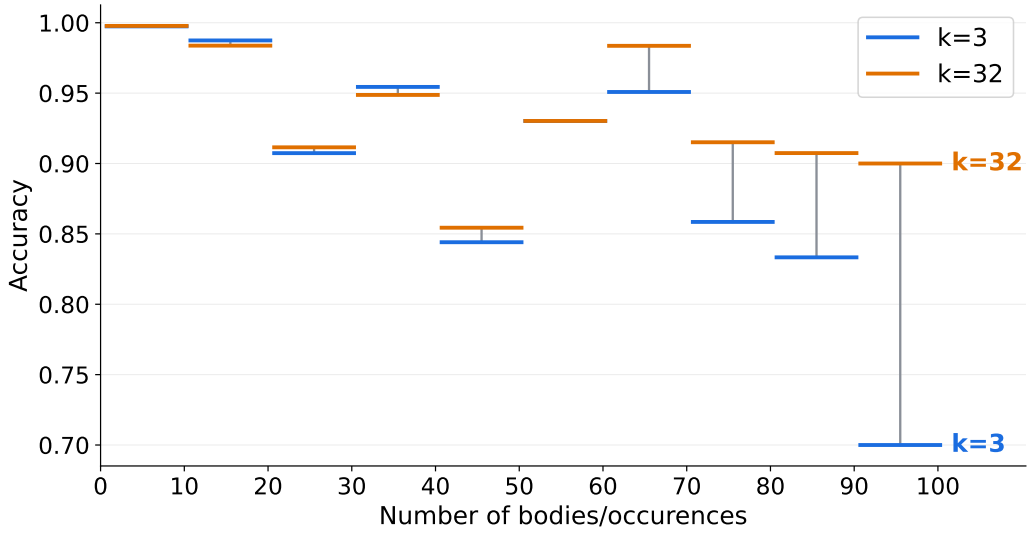


Figure 9: Channel-expansion ablation on the *warmup* task, by number of bodies in the assembly. Both models use the limited task mix from Section 7.2; only the selection-buffer embedding width differs ($k = 32$ vs. $k = 3$). Each marker spans a ten-body range and gives the accuracy over all samples in that range; assemblies above 100 bodies are omitted (0.6% of samples). The widths are indistinguishable up to 60 bodies, after which $k = 32$ leads in every bin.

wise for the pooled tokens and for each DeepStack feature. The remainder of the base model's forward pass runs on the combined embeddings exactly as in the unmodified model.

Autoregressive generation uses the same mechanism: the selection buffer encoder runs once during the prefill phase, when the vision encoder processes the image, and the hook fuses its output at that point.

C Ablations

In this section we report on ablation studies conducted to back up some of the design decisions used when building SelectionVLM.

Channel expansion. As described in Section 4, the pick ID at each pixel is mapped to a learned k-dimensional vector, producing a k-channel representation that serves as input to the selection-buffer encoder. Although the pretrained image encoder accepts three RGB channels, a three-dimensional embedding may provide insufficient representational capacity when many distinct entity identities occur in the same assembly. We test this hypothesis by training two SelectionVLM variants with otherwise identical hyperparameters. Our default model uses $k = 32$, whereas the ablated model uses $k = 3$, matching the input width of the RGB encoder and providing a learned analogue of a three-component color encoding.

Figure 9 reports *warmup* task accuracy as a function of the number of bodies in each assembly. Each marker represents a ten-body interval and gives the accuracy over all evaluation samples within that interval. Across the full evaluation set, the two variants perform similarly, achieving 95.7% accuracy for $k = 3$ and 96.0% for $k = 32$. For assemblies containing at most 60 bodies, their accuracies differ by less than one percentage point in every interval. For the reported intervals above 60 bodies, however, $k = 32$ consistently outperforms $k = 3$: 98.4% versus 95.1% for 61–70 bodies, 91.5% versus 85.9% for 71–80, and 90.7% versus 83.3% for 81–90. The consistent advantage of $k = 32$ indicates that a wider pick-ID embedding may improve robustness when many entity identities must be distinguished simultaneously. We therefore retain $k = 32$ as the default.

Selection buffer encoder depth. As described in Section 4, the depth of the selection-buffer encoder is determined by the number of transformer blocks retained from the pretrained image encoder. We hypothesize that encoding the simpler selection-buffer signal does not require the full depth of the vision transformer pretrained for RGB imagery. We evaluate this hypothesis by comparing two SelectionVLM variants with otherwise identical hyperparameters: our default of 8 blocks, which also disables the DeepStack pathway on the buffer branch, and the full pretrained depth with DeepStack features injected into the language model.

Across task families the two variants agree to within a few percentage points. On *attribute*, full depth reaches 95.6% versus 94.9% for the 8-block encoder; on *entity to description*, BLEU is 0.098 versus 0.087. Differences on *joint*, *spatial*, *counting*, and *hole* are similarly small and, in several cases, favor the shallower encoder (e.g. *spatial*: 29.3% vs. 24.2%). On *description to entity*, full depth reaches 75.2% exact-match accuracy versus 74.5% for the 8-block encoder. We therefore use the 8-block encoder in all experiments: it reduced the number of trainable parameters on the buffer branch from 612M to 188M with no measurable loss in task accuracy.

D AGENTIC IDENTIFICATION OF A SINGLE COMPONENT

In Section 1 we describe the agentic workflow which frontier models use to identify the objects in a screen shot from a CAD system. In this section we give two examples of the way Claude Opus-5, with medium thinking, solves the problem of identifying the red flywheel shown in Figure 5 (c). In both prompts below, the model is instructed not to use the text from the design, preventing the wheel being trivially identified as the component named "flywheel v1". Many professional designs include internal part numbers rather than semantic text, so we simulate the case where no useful text is available.

D.1 COLOR-BASED IDENTIFICATION.

When the color of the wheel is mentioned in the prompt, this clue is immediately exploited as the primary way the component is identified. As shown in Section 7.1, identification by distinctive colors is inferior to utilizing the selection buffer modality when both strategies are compared for the same base model.

Prompt

Identify the red wheel and select it. Do not use any text from the CAD system. Describe how the part was identified.

Model output

The red wheel is now selected—the whole spoked flywheel on the right side of the engine is highlighted, and nothing else is.

How I identified it without reading CAD names or labels:

- Took a screenshot of the view and identified a single red, spoked wheel at the right end of the assembly.
- Scanned every occurrence's appearance colour values and bounding-box geometry. Three parts had a red appearance (approximately RGB $(196, 53, 39)$), but only one had wheel-like proportions: a thin disc 8.47 cm across in two directions and 0.95 cm thick.
- The other two red parts were smaller, non-disc-shaped, and located elsewhere in the assembly.
- Selected the corresponding occurrence and confirmed the selection with a second screenshot.

D.2 GEOMETRY-ONLY IDENTIFICATION.

This prompt further restricts the model to using geometric analysis only.

Prompt

Identify the large wheel and select it. Do not use any text from the CAD system. Do not query any materials or colors. Just use geometric queries. Describe how the part was identified.

Model output

The large wheel is now selected—it is the large spoked flywheel on the right-hand side of the assembly.

I found it purely from geometry, with no reference to names, materials, or colours:

- Computed each body's bounding box, volume, and surface types.
- Grouped circular edges by shared axis and coincident axis line to identify parts with rotational symmetry.
- Ranked axisymmetric candidates by outer radius and disc-ness (diameter relative to axial thickness).
- One body was clearly dominant: it had an outer diameter of 8.47 cm, an axial thickness of 0.95 cm, a diameter-to-thickness ratio of approximately 7.4, 12 concentric circular edges at four distinct radii about a single Y axis, and a central bore of radius 0.32 cm.
- The next-largest axisymmetric bodies measured only 2.54 cm and 2.09 cm in diameter, making the ranking unambiguous.

The outermost surface reaches a radius of 4.235 cm, whereas the furthest vertices reach only 3.529 cm. This is consistent with a rounded, revolved wheel tread rather than a flat-sided disc, providing additional evidence for the identification.

D.3 SelectionVLM using the selection buffer modality

Prompt

Select the wheel. Give me the pick IDs in a Python list.

Model output

```
[41]
```

D.4 Comparison of token usage

The agentic workflows investigated above were performed using Cursor as the harness. The harness and tool definitions add considerable token overhead to the procedure, which is not directly comparable with the workflow using SelectionVLM. In Table 3 we show both the total token usage and the tokens used for tool calls, tool responses and associated reasoning around the agentic part of the workflow. This excludes system prompts, tool definitions and the final model output, providing a number which is can be compared with SelectionVLM. We see that SelectionVLM is more than an order of magnitude more efficient, requiring 50 tokens for the prompt and chat template, 1026 tokens to process the input image and 4 tokens of output.

Table 3: Token usage for identifying the flywheel

	Color-based prompt	**Geometry-only prompt**	**SelectionVLM**
Total token usage	54k	69k	1080
Tool calls and responses	17k	23k	1030

E LLM USAGE

The authors wish to acknowledge the great help which LLM tools provided in the creation of this paper. LLMs were used for literature searching, agentic coding and during the final writeup. All the key ideas in the paper are from the authors and we have extensively checked, corrected and reworded all LLM output.

E.1 LITERATURE SEARCHES PRIOR TO IMPLEMENTATION

ChatGPT Deep research was used for extensive literature searching to discover techniques for modifying VLMs which might be suitable for processing selection buffer information. Relevant papers were then carefully studied by the authors and are now summarized in Section 2.

E.2 AGENTIC CODING

The libraries used for renderings the dataset, creating the tasks and adding the selection buffer modality were all written using coding agents.

Dataset creation required extensive manual verification and corrections. Each task type and template was carefully reviewed. Considerable effort went into ensuring that the difficulty of the baseline tasks matched the task given to SelectionVLM.

Planning and initial implementation for the selection buffer encoder was carried out with Claude Opus-4.6. The key aspects of architecture were specified by the authors, including mapping the pick IDs into k-dimensional embeddings to allow a large number of entities to be distinguished and reusing copies of layers from the original vision encoder to make the embeddings easy to merge. The approaches used in (Hua et al., 2024; Sun et al., 2024) and (Cao et al., 2025) were also discussed in the initial design phase. The requirement to additionally provide DeepStack features was raised by Claude. Debugging and iteration on the design was carried out using Composer 2.5. The final architecture reported in the paper closely follows the authors initial design.

E.3 WRITING THE PAPER

During code implementation and subsequent experiments, extensive notes were maintained to assist with writing the paper. The paper sections were initially written by the authors as bullet points with links to the relevant notes. These were expanded into full sections by a combination of Claude Opus-4.8 and Composer 2.5. Extensive re-wording was then carried out to correct errors and improve style.