FlexPart: Text-Controlled Semantics-Aware 3D Part Segmentation at Flexible Granularity

Yiqun Zhao[1,2*] Jiachen Xu[2] Yihao Zhi[3,2*] Yunfei Zhao[2] Zibo Zhao[2] Xintong Han[2]
Chunchao Guo[2,†] Shenghua Gao[1,4,†]

[1]The University of Hong Kong [2]Tencent Hunyuan
[3]The Chinese University of Hong Kong, Shenzhen [4]Shenzhen Loop Area Institute

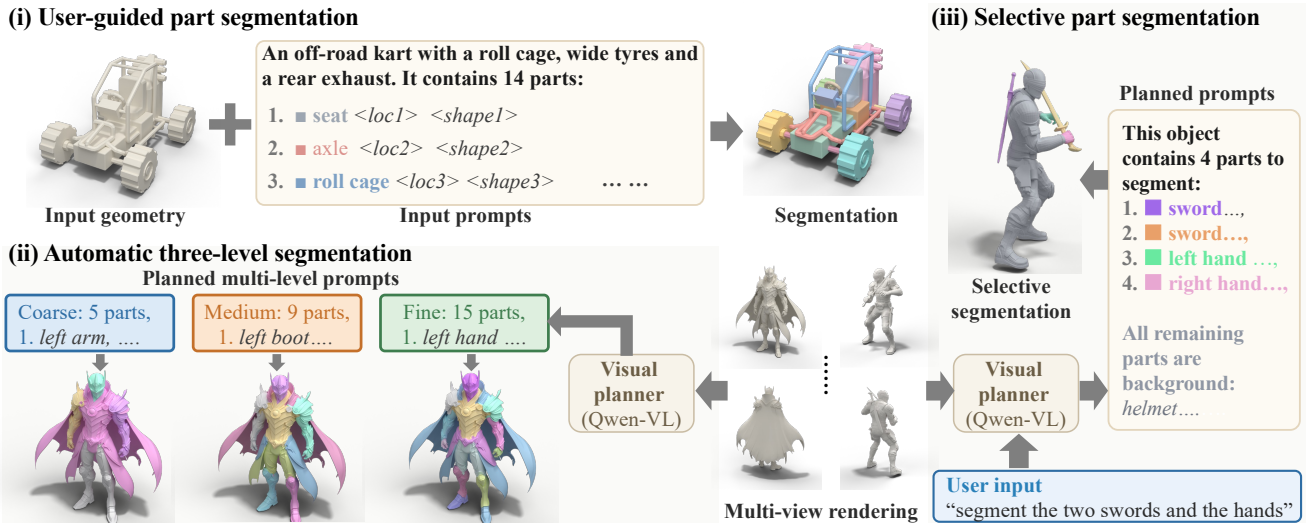


Figure 1. **A unified framework for text-driven, semantics-aware 3D part segmentation.** FLEXPART partitions a given 3D geometry following text prompts. A prompt defines the desired label space using semantic part names augmented with location (<loc>) and shape (<shape>) descriptors. **(i)** Users can manually author this detailed item list for precise control. **(ii)** To streamline interaction, a visual planner can automatically construct these prompts from multi-view renderings. By simply varying the planned text prompts, the FLEXPART model dynamically outputs coarse, medium, or fine partitions. **(iii)** Given a simple, free-form user command, the planner isolates the requested parts and automatically assigns the remaining parts to the background, enabling selective segmentation.

Abstract

A 3D asset admits multiple valid part decompositions, while semantics-aware segmentation should reflect both the desired granularity and the parts a user wants to extract. Ideally, a practical segmentation model should strive for a simple principle: ***what you ask for is what you get****. However, realizing this goal is hindered by two core obstacles: the lack of multi-granularity training data and the absence of a holistic control mechanism. We propose* FLEXPART*, a unified framework for text-controlled 3D mesh segmentation. To establish flexible control, we introduce a holistic* control prompt *comprising semantic part names with location and shape clauses. To support this mechanism, we utilize an automated Vision-Language Model (VLM) pipeline to construct a large-scale dataset of multi-granularity partitions and their corresponding prompts. Driven by this data,* FLEXPART *generates a view-independent* part-identity field *in native 3D space, directly assigning discrete part IDs to mesh faces. To compel the model to respect semantic conditions rather than rely on geometric priors, we train the model using randomized background dropping. Furthermore, an inference-time VLM planner can automatically complete descriptive clauses from 2D renderings, allowing users to simply name the desired parts in ordinary words.* FLEXPART *with planner-completed prompts outperforms the strongest*

[*]Work done during an internship at Tencent Hunyuan.
[†]Corresponding authors.

evaluated baseline by 15% *in mIoU, even when that baseline receives ground-truth guidance. On our held-out* FLEXPART-BENCH*,* FLEXPART *leads at each granularity and improves average mIoU by* 22% *over the strongest baseline. With only the requested part names supplied by the user, it also improves IoU by* 35% *over the strongest click-driven baseline on selective segmentation.*

1. Introduction

3D part segmentation provides explicit boundaries on mesh surfaces, facilitating downstream applications such as part-level editing, rigging, animation, and 3D printing. Crucially, a single 3D asset can be decomposed in multiple valid ways depending on the user's intent: coarse functional groups for retargeting, individual accessories for texturing, or specific named parts for localized edits (as shown in Fig. 1). However, two core obstacles remain in achieving this flexible, intent-driven control. First, existing 3D datasets lack multi-granularity semantic annotations. Second, the absence of a holistic, view-independent control signal often leads to fragmented boundaries and unresolved occlusions. Therefore, developing a unified framework that couples a global semantic control signal with multi-granularity decomposition is essential to realize the user-centric principle: **what you ask for is what you get**.

Regarding the first obstacle, existing 3D datasets typically provide only a single, fixed partition per geometry, mostly sourced from open-source repositories or fixed artist creations, lacking consistent multi-granularity semantics-aware annotations. Recognizing this data scarcity, recent methods often lift 2D priors [10, 17, 20, 29, 32, 38, 46, 47] or directly cluster 3D features [18] heuristically. Consequently, their partitions are inevitably biased toward data-intrinsic boundaries and are largely governed by low-level geometric cues rather than explicit user intent. For example, models trained on fixed geometry segmented data rigidly segment an "arm" into sub-parts, failing to align with diverse user intents such as animating the whole arm or editing individual segments. In essence, without multi-granularity data, achieving truly flexible, intent-driven control remains fundamentally out of reach.

Beyond the availability of suitable training data, the mechanism used to formulate and inject user intent is equally critical. While some works attempt to control granularity through global heuristics, such as a scale parameter [38], a clustering threshold [29], or a target part count [18], these mechanisms remain semantically blind, failing to specify which boundaries actually matter. On the other hand, using single-view 2D masks [13] introduces severe view dependence, leaving occluded regions unspecified. Consequently, the reliance on localized or view-dependent signals prevents existing methods from producing coherent, holistic partitions. Closest to our work, recent methods explore control via explicit text schemas [49]. However, they still struggle to offer flexible user-intent control, as these text schemas lack spatial and geometric grounding.

We overcome these two obstacles and unlock flexible, intent-driven control by introducing a *control prompt* as our input signal, which comprises target part names paired with their specific location and shape clauses. This prompt dynamically defines the output label space: varying the prompt directly specifies a coarse, medium, or fine partition, while leaving unspecified regions as background naturally enables selective segmentation. To overcome the lack of multi-granularity training data, we design an automated annotation pipeline powered by a Vision-Language Model (VLM, specifically Gemini). This pipeline analyzes artist-authored components from HY3D-Bench [8] and hierarchically merges them based on semantic and functional relationships to form three granularities, subsequently generating the corresponding control prompts for each level. By applying this pipeline, we annotate $57K$ objects, yielding a dataset of $150K$ prompt-target pairs.

Building upon this large-scale dataset, we introduce FLEXPART, a unified framework for text-controlled 3D mesh segmentation. Our architecture builds upon generative surface attribute models [12, 13] by integrating a part-aware VAE and a text-conditioned Diffusion Transformer (DiT) [22]. Given the input mesh and the control prompt, the DiT generates a latent representation that decodes into a native 3D *part-identity field*. Querying this continuous field at the mesh's face centres yields discrete, binary-encoded part IDs, thereby directly labeling the input without altering its original geometry. Furthermore, to support selective segmentation, we introduce a *drop-to-background* training augmentation that randomly omits parts from the prompt, implicitly assigning their corresponding regions to the background. This diverse training strategy compels the model to follow the semantic prompt rather than being constrained by low-level geometric priors. During inference, a VLM-based planner automatically deduces the location and shape clauses from 2D renderings, unlocking a truly effortless experience where providing simple part names yields precise, intent-driven segmentation.

Our contributions are summarized as follows:

- We introduce a flexible, intent-driven control mechanism via holistic control prompts, and propose an automated VLM-based annotation pipeline to construct a large-scale, multi-granularity dataset with structured part descriptions.
- We propose FLEXPART, formulating text-controlled 3D segmentation as a native 3D part-identity field. Equipped with a randomized background dropping strategy, a single model seamlessly supports multi-granularity partitions and selective segmentation.
- Extensive experiments demonstrate that FLEXPART out-

performs evaluated baselines on PartObjaverse-Tiny, achieves state-of-the-art results at each granularity of our held-out FLEXPARTBENCH, and significantly improves selective part segmentation.

2. Related Work

Overcoming 3D data scarcity for part segmentation. Existing 3D datasets rarely provide multi-granularity semantic annotations for the same geometry, so most methods rely on external priors to infer parts. One line of work distills features from 2D image foundation models [10, 21, 26] into 3D, either by optimizing a per-shape feature field [38] or by predicting masks on multi-view renderings and lifting them onto the surface [4, 7, 29, 36]. Per-object optimization is costly, and mask fusion is sensitive to view coverage. Open-vocabulary approaches [17, 20, 32, 46, 47] leverage vision-language models by querying rendered views with part names and aggregating predictions back to 3D, but operating in 2D projected space introduces multi-view inconsistencies and fragmented boundaries on the mesh. A second line learns part features natively in 3D from curated supervision. PartField [18] clusters a learned hierarchy at a chosen level, and P3-SAM [19] predicts point-prompted masks and automatically selects and merges them into a part segmentation. These methods produce high-quality decompositions, yet because their training data contain only fixed geometric splits, the resulting partitions are governed by low-level geometric cues rather than user-intended semantic groupings.

Control signals for part segmentation. Given the geometry-biased representations above, various control mechanisms have been proposed, yet all exhibit critical limitations. Global heuristics like scales or target counts [18, 29, 38] remain semantically blind, failing to specify which boundaries actually matter. Localized prompts such as clicks or boxes [19, 48, 50] leave semantic grouping implicit, while single-view 2D guidance [13] introduces severe view dependence and leaves occluded regions unspecified. Consequently, these localized or view-dependent signals cannot produce coherent, holistic partitions. Closest to our work, schema-driven approaches [23, 49] use text schemas but lack spatial and geometric grounding; furthermore, they synthesize separate geometries rather than segmenting the input surface, and lack multi-granularity training. Unlike these methods, FLEXPART conditions on a holistic text prompt to directly generate a native 3D part-identity field, explicitly trained on multi-granularity annotations to achieve flexible, intent-driven segmentation.

Part-level 3D generation. While modern generators excel at producing high-quality holistic shapes [9, 11, 35, 41, 42, 45], recent research has expanded into part-level generation under two main paradigms. Single-stage methods synthesize all parts directly from an image or latent representation, automatically discovering the decomposition [2, 3, 15, 30, 43]. Consequently, users lack explicit control over the resulting part structure. Conversely, two-stage methods take an existing shape alongside its decomposition to generate the complete geometry of each individual part [5, 14, 28, 37, 39, 40], meaning their performance is strictly bounded by the input segmentation. Our method provides this segmentation from text, at a user-specified granularity.

3. Method

As illustrated in Fig. 2, we propose FLEXPART, a unified generative framework for semantics-aware 3D part segmentation driven by a holistic, view-independent text control signal. By simply adjusting the text prompt, users can flexibly control the segmentation granularity and select specific target parts. Building upon native 3D surface fields (Sec. 3.1), we formulate segmentation as the generation of a continuous 3D part-identity field. Unlike prior methods that predict noisy continuous colors, our field predicts discrete, binary-encoded part IDs that explicitly align with the prompt's label space (Sec. 3.2). Our architecture features a part-aware VAE to encode the mesh and its part IDs into a unified latent, alongside a text-conditioned dual-stream DiT that generates these latents from the holistic prompt. Finally, this robust text-to-3D grounding is enabled by training the framework on comprehensive multi-granularity data (Sec. 3.3).

3.1. Preliminary

Voxel-anchored latents. Following LATTICE [11], we encode an input point cloud $\mathcal{P}$ via cross-attention using anchors at the occupied voxel centres $\{\mathbf{v}_i\}$ of an N^3 grid:

$$\mathbf{z} = \psi\big(\mathbf{v}_i, \mathcal{P}\big) \in \mathbb{R}^{n \times d}, \tag{1}$$

where ψ denotes the encoder. These anchors naturally yield integer grid indices $\mathbf{V}$. Following previous work [44, 45], we train a text-conditioned flow-based diffusion model (DiT) [16] on these structured latents. During denoising, the indices $\mathbf{V}$ provide spatial guidance by serving as a rotary positional embedding (RoPE) [27] for each token. By explicitly grounding tokens in 3D space, this spatial guidance enables the model to precisely localize features, which is crucial for generating accurate and fine-grained surface attributes.

Dense surface fields. Each latent [11] is anchored at a local voxel, so it carries the information of the surface near that voxel. The decoded latent can therefore produce a continuous field $\mathbf{s}$ from $\mathbf{z}$ at any surface point $\mathbf{x}$,

$$\mathbf{s}(\mathbf{x}) = \varphi\big(\mathbf{x}, \mathbf{z}\big), \tag{2}$$

where φ denotes the VAE decoder. The predicted attribute $\mathbf{s}$ can represent an SDF for local geometric details, as in LATTICE [11], or an SDF and texture color separately, as in NaTex [12]. In our method, we reformulate this attribute

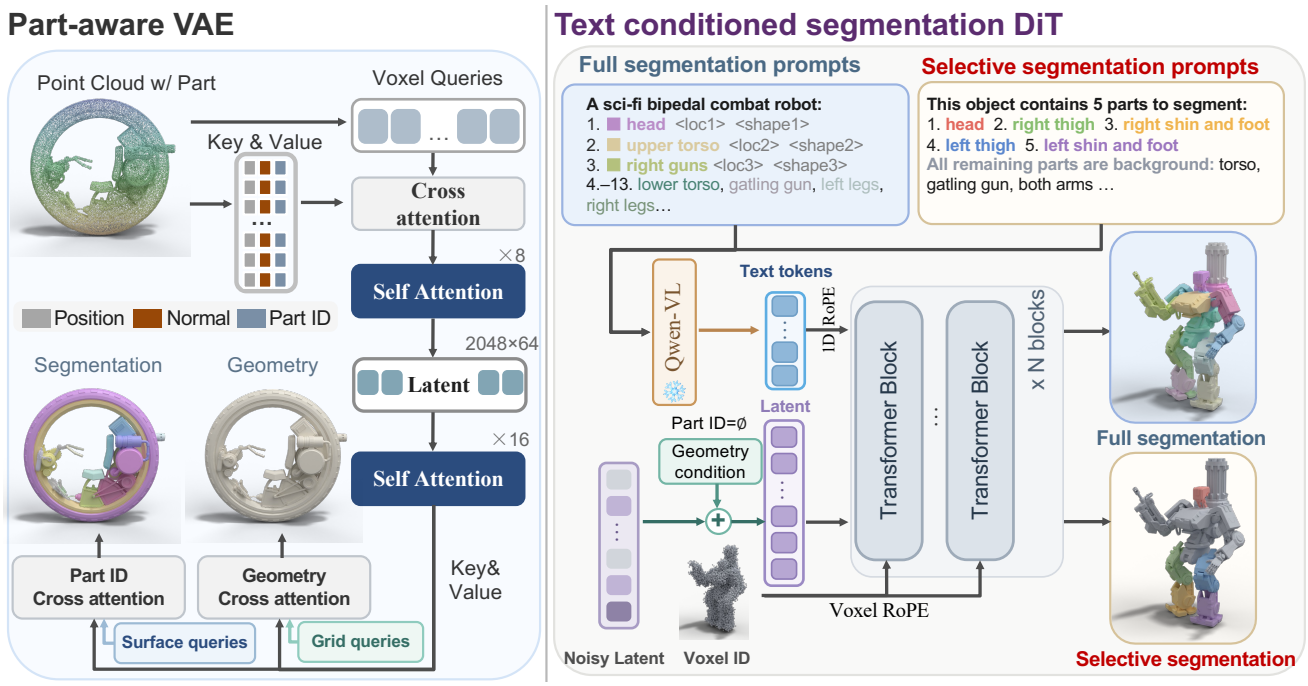


Figure 2. **FLEXPART.** *Left:* a part-aware VAE encodes a point cloud carrying position, normal, and part ID. Its decoder can be queried at any 3D coordinate, with two cross-attention modules for geometry (SDF) and a discrete, binary-encoded part ID. Face-centre queries produce the mesh labels. *Right:* a text-conditioned DiT generates the field's latent representation from the geometry latent (green) and encoded text tokens. A full decomposition (blue) and a selective input (red) differ only in the control prompts, and in the selective case an explicit background item absorbs parts that belong to the background. Each part item is visualized in the color corresponding to the input prompts.

as the SDF and binary-encoded part IDs, which is detailed in Sec. 3.2.

3.2. Formulating segmentation as text-controlled part-ID generation

Our method takes the input mesh M and a text prompt $\mathcal{T}$ describing K parts as an ordered list, optionally followed by a background item describing the remaining parts. The output is a continuous part-ID field; querying it at the face centres of the input mesh with Eq. 2 yields a per-face label for each of the F faces $\{f_1, \ldots, f_F\}$.

Prompt format. To provide holistic descriptions of every target part, each item is written as a semantic name with a location and a shape clause, so a full decomposition is conditioned on <sem$_1$><loc$_1$><shape$_1$>…<sem$_K$><loc$_K$><shape$_K$>. For example, one item might read "*right arm: upper right side of the torso, a tapering cylindrical limb*." A selective prompt keeps the wanted parts in that form and appends the remaining ones as background items <bg$_1$><bg$_2$>…, each still carrying its own description but all decoding to the single label bg, as in Fig. 2.

Binary-encoded part IDs. Prior generative segmentation methods represent parts using distinct RGB colors [13]. However, this continuous RGB space 256^3 often leads to noisy, fragmented predictions within a single part, requiring ad-hoc post-processing such as color clustering to recover part identity. To resolve this, we represent surface attributes using discrete, binary-encoded part IDs id $\in \{0,1\}^6$ to represent the surface attribute that labels the continuous space. Reserving the all-ones label for the background, this 6-bit format supports up to $K = 64 - 1$ distinct parts, which is sufficient to accommodate the fine-grained structural complexity of most objects. To effectively ground these IDs to user text, we map the i-th prompt item to the i-th binary ID. Unlike permutation-invariant approaches that lack structural constraints, this locally aligned prompting provides explicit ordering priors, yielding significantly more accurate part assignments.

Part-aware VAE. We keep the voxel anchors of Eq. 1 as queries, and the point cloud they attend to carries a part ID alongside each point's position and normal. An 8-layer self-attention encoder ψ encodes the shape into a 2048×64 latent. The decoder φ decodes the latent into fields. As the part field cannot be isolated from geometry information, the latent encodes both geometry and part information. We query the SDF using grid locations that make the latent aware of the location, while querying the six-bit part ID using mesh face centres as queries. We train the VAE jointly using

$$\mathcal{L}_{\text{vae}} = \lambda_{\text{kl}}\mathcal{L}_{\text{kl}} + \lambda_{\text{sdf}}\mathcal{L}_{\text{sdf}} + \lambda_{\text{part}}\mathcal{L}_{\text{part}}, \qquad (3)$$

where $\mathcal{L}_{\text{kl}}$ is the standard KL divergence regularizing the latent space, $\mathcal{L}_{\text{sdf}}$ applies a penalty for geometry reconstruction errors, and $\mathcal{L}_{\text{part}}$ uses Binary Cross-Entropy (BCE) to

supervise the part IDs. For VAE training, part indices are randomly shuffled before encoding to prevent order memorization.

Dual-stream DiT. We employ a dual-stream architecture, inspired by MM-DiT [1, 11], to jointly process the text condition and the 3D latent. First, a frozen Qwen3-VL [24] encoder extracts text tokens $\mathbf{T}$, which are injected with a 1D RoPE [34]. Simultaneously, to condition on the input geometry, we run the VAE encoder with all part IDs set to zero to produce a part-agnostic shape latent $\mathbf{z}_{\mathrm{s}}$. This geometry condition is added element-wise to the input noisy latent. These latent tokens are then provided with a 3D RoPE [11, 12] over the voxel indices $\mathbf{V}$ of Eq. 1. To prevent positional collisions, text indices are strictly offset past the voxel indices. At inference time, the DiT iteratively processes and denoises the latent from noisy $\mathbf{z}_t$ to clean latent $\mathbf{z}_0$. The clean latent $\mathbf{z}_0$ is passed to the VAE decoder φ to predict the 3D part segmentation with face centres as queries.

Objective. We train the model with a flow-matching [6, 16] objective. The target $\mathbf{z}_0$ is the VAE latent of the part-ID field. Given Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and a time step $t \sim U[0,1]$, we form the interpolated state $\mathbf{z}_t = t\mathbf{z}_0 + (1-t)\,\boldsymbol{\epsilon}$ and supervise the network on the two conditions above, the text tokens $\mathbf{T}$ and the shape latent $\mathbf{z}_{\mathrm{s}}$, to predict the velocity $\mathbf{z}_0 - \boldsymbol{\epsilon}$,

$$\mathcal{L}_{\mathrm{fm}} = \mathbb{E}_{t,\,\mathbf{z}_0,\,\boldsymbol{\epsilon}} \left\| \mathbf{v}_\phi\big(\mathbf{z}_t, t, \mathbf{T}, \mathbf{z}_{\mathrm{s}}\big) - (\mathbf{z}_0 - \boldsymbol{\epsilon}) \right\|_2^2. \quad (4)$$

3.3. Data construction

Partitions. We take the artist-authored components of HY3D-Bench [8] as atomic units. Inspired by CubePart [49], which uses set-of-mark renderings to group parts, we mark every component across views and utilize a vision-language model's multi-view reasoning ability for three-level partitions. Duplicate levels are dropped, so an object used in training yields up to three partitions.

Text annotation. For each partition we render every grouped part with its boundary overlay, annotate that part alone with the clauses of Sec. 3.2, then aggregate the results and have a language model refine them into one prompt. During training we augment it with *drop-to-background*: a random subset of items is dropped and their faces reassigned to bg. This is where bg acquires its meaning, so it is what enables selective segmentation, and it also strengthens the correspondence between a named item and its label. The detailed pipeline is described in App. A.1.

Inference. The semantic prompts are either provided directly by the user or automatically generated by a prompt planner. We instantiate this planner using a frozen Qwen3.8 [25] model, which processes multi-view renderings of the input mesh. The planner plays a critical role in selective segmentation scenarios: when a user specifies only a subset of target parts, the planner automatically infers and fills in the remaining background categories, ensuring a complete and valid partition of the entire 3D asset.

4. Experiments

4.1. Experimental setup

Training data. We run the annotation pipeline of Sec. 3.3 over 57K objects sampled from HY3D-Bench [8], holding the category proportions of the source library fixed. This yields $151{,}492$ prompt-target pairs. App. A.1 provides the annotation details.

Benchmarks. We evaluate our method on three main datasets. **PartObjaverse-Tiny** [38] contains 200 public objects, enabling direct baseline comparisons. Following P3-SAM [19], we use its watertight version. Its original part decomposition is used as given: our pipeline only writes the text description of each part and never edits a part boundary. **FLEXPARTBENCH** is our held-out benchmark of 364 assets proportionally sampled from PartNeXt and PartVerse-XL. Each is annotated at three granularities with strictly no training overlap. **Selective-190** comprises 190 manually constructed selective cases (ranging from 2 to 18 named parts) across 28 FLEXPARTBENCH objects. We manually inspected and removed objects with defective annotations or source geometries; App. A.4 details the construction and category distributions of all sets. Finally, we demonstrate that our method generalizes well to generated meshes.

Metrics. The baselines do not provide the ordered prompt-to-part-ID correspondence used by FLEXPART. We therefore compare all methods using class-agnostic mIoU [18, 19], averaging each ground-truth part's maximum IoU over predicted regions. For FLEXPART, we also evaluate the same predictions using *-direct* mIoU, which compares output region k with the ground-truth region for prompt item k, without rematching. For selective segmentation, we report named-part and background IoUs separately to expose foreground leakage.

Baselines. We compare against five methods, evaluated in their intended settings (inputs specified in the *Conditioning* column of Tab. 1). Find3D [20] takes a list of part names, identical to ours. SAMPart3D [38] requires a single scale value, which we calibrate to the benchmark granularities. PartField [18] clusters a learned feature field using the ground-truth part count. P3-SAM [19] automatically selects and merges predicted masks into a segmentation; we evaluate its native automatic mode without granularity control and exclude it from Tab. 2. SegviGen [13] requires a color-coded guidance map. We evaluate it using both ground-truth maps and maps automatically predicted by prompting SAM3 on a front view with the target part names. For FLEXPART, we evaluate a single model using two prompt variations: the annotated item list, and part names enriched with planner-completed location and shape clauses. SegviGen and FLEX-

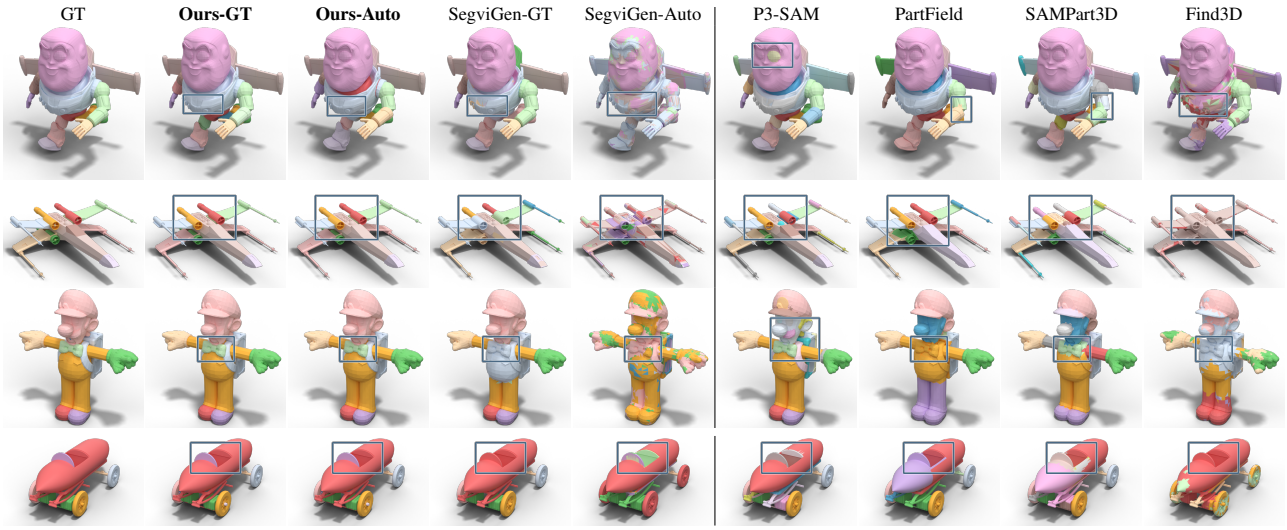


Figure 3. **Qualitative comparison on PartObjaverse-Tiny.** Left of the divider, *-GT* denotes ground-truth conditions (our annotated prompt; SegviGen's rendered guidance map), and *-Auto* denotes automatic conditions (our planner-completed names; SegviGen's predicted map). Right of the divider, baselines use their native inputs. As shown, baselines suffer from missing occluded parts (SegviGen), over-segmenting along geometric seams (PartField, P3-SAM, SAMPart3D), or imprecise boundaries (Find3D). FLEXPART avoids these issues, effectively following the semantic intent from GT.

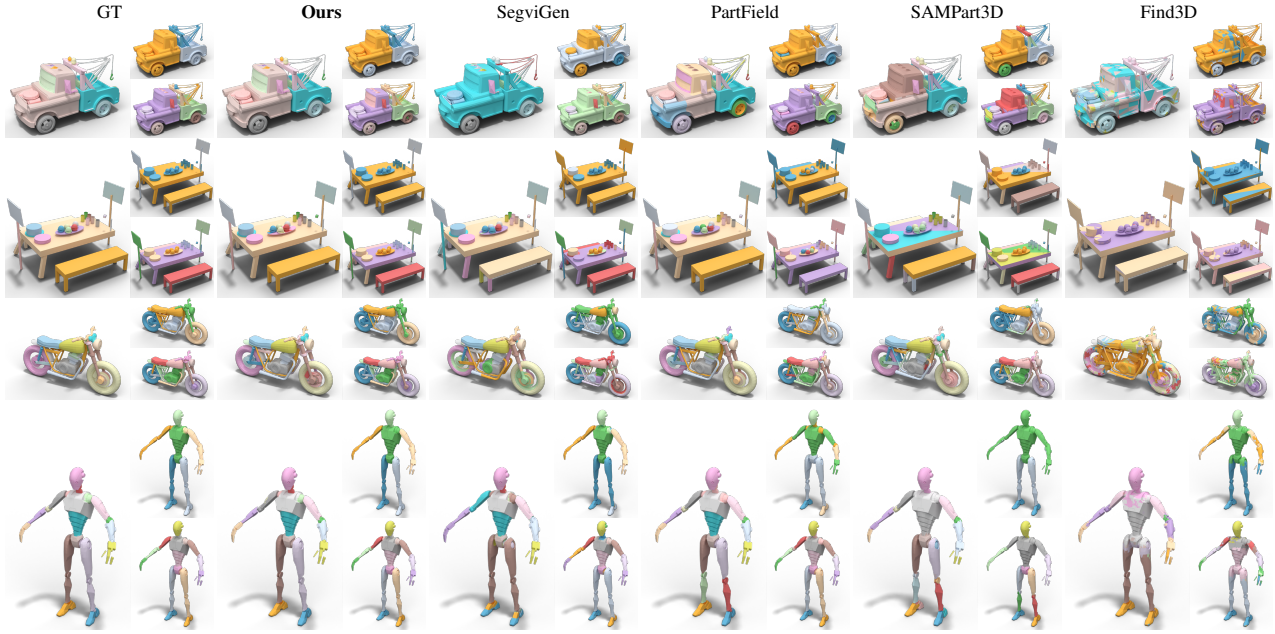


Figure 4. **Granularity control on FLEXPARTBENCH.** For each method, panels show fine (large), coarse (top right), and medium (bottom right) granularities. While baselines receive their optimal inputs (e.g., GT part counts for PartField), only FLEXPART (driven by level-specific item lists) successfully adapts its semantic decomposition across all three levels. Conversely, PartField and SAMPart3D are biased toward geometric boundaries over the requested semantic groupings, and Find3D suffers from severe surface fragmentation. Meanwhile, SegviGen struggles to follow the ground truth, since relying on a single reference map fails to resolve occlusions.

PART are evaluated on both annotated (*-GT*) and automatic (*-Auto*) conditions (Tab. 1, Fig. 3). All text-conditioned methods receive identical English part names. Finally, while CubePart [49] and MultiCube [23] are the closest schema-conditioned generative methods, MultiCube remains unreleased. As CubePart generates separate per-part meshes rather than per-face labels, direct quantitative comparison is infeasible; we thus provide qualitative comparisons in App. A.6. Detailed protocols are in App. A.4.

Implementation. Our implementation builds on the open-source Hunyuan3D 2.1 [31]. Specifically, we adopt the MM-DiT conditioning, part-aware VAE, and rotary position encoding strategies from LATTICE [11] and NaTex [12]. Text conditioning is processed using a frozen Qwen3-VL-4B-

Table 1. **Part segmentation on PartObjaverse-Tiny.** *Conditioning* denotes the input signals. *-GT* uses ground-truth annotations, while *-Auto* relies on automated inputs (e.g., predicted color maps or planner-completed lists). *-direct* evaluates our predictions without region-to-GT rematching, and still outperforms all baselines.

Method	Conditioning	mIoU ↑
Find3D [20]	part names	0.282
SAMPart3D [38]	one scale value	0.483
PartField [18]	target part count	0.517
P3-SAM [19]	none	0.560
SegviGen-GT [13]	part color map	0.594
SegviGen-Auto	part color map, predicted	0.413
FLEXPART-GT	full item list	0.691
FLEXPART-Auto	part names only	0.681
FLEXPART-direct IoU	full item list	0.614

Table 2. **Multi-granularity control.** Methods are evaluated across three granularities on FLEXPARTBENCH using their optimal inputs. FLEXPART significantly outperforms all baselines at every level, and even our *-direct* protocol (Tab. 1) exceeds all baselines on average. Although PartField performs reasonably well at the fine level by merely fitting the geometry, all baselines struggle at the coarse and medium levels where semantic understanding is required.

Method	Coarse ↑	Medium ↑	Fine ↑	Avg. ↑
Find3D [20]	0.364	0.154	0.086	0.201
SAMPart3D [38]	0.473	0.359	0.376	0.402
SegviGen [13]	0.592	0.466	0.409	0.489
PartField [18]	0.508	0.555	0.499	0.521
FLEXPART	0.763	0.635	0.511	0.637
FLEXPART-direct IoU	0.751	0.615	0.467	0.611

Instruct [24] encoder. During inference, we employ the flow-matching Euler sampler [6] with 30 steps and a guidance scale of 6.0. Detailed architectural configurations and the training schedule are provided in App. A.3.

4.2. Main results

Evaluation on PartObjaverse-Tiny. As shown in Tab. 1 and Fig. 3, FLEXPART achieves the highest accuracy. Lacking semantic control, PartField, P3-SAM, and SAMPart3D rely strictly on geometric boundary priors. Find3D uses text but lacks clean global assignments, while SegviGen's single-view guidance is vulnerable to occlusions. In contrast, our global text-driven approach inherently resolves these occlusions. Furthermore, we evaluate an automated setting (FLEXPART-Auto) that requires minimal user effort. Users simply provide part names, and our planner automatically completes the detailed location and shape clauses based on renderings. This automation incurs a negligible performance drop of just 0.010 mIoU compared to the fully annotated ground truth (*-GT*). In contrast, SegviGen-Auto suffers a severe 0.181 mIoU drop. Crucially, while SegviGen requires a 2D guidance map that is either tedious to paint or suboptimal to automate, our method only demands a simple text list, ensuring superior usability and robustness. Notably, even under a strict no-rematching protocol, our *-direct* variant still outperforms all baselines (Tab. 1).

Table 3. **Selective part segmentation.** Query IoU averages over the named parts, and BG IoU covers the background region. FLEXPART outperforms all click-driven baselines, even when relying on planner-generated prompts from names only. While baseline methods using solely part names (Find3D) fail to extract user-intended parts, our control prompts successfully achieve accurate selective segmentation.

Method	Input	Query IoU ↑	BG IoU ↑
Find3D [20]	part names	0.090	0.745
P3-SAM [19]	GT clicks	0.230	0.413
SegviGen [13]	GT clicks	0.456	0.764
FLEXPART (planner)	names only	0.615	0.895
FLEXPART (GT desc.)	full descriptions	0.657	0.933

Table 4. **Ablation.** *Full* averages the three granularities on a human category subset from FLEXPARTBENCH; *Selective* is the accuracy on Selective-190. Using an RGB palette instead of part IDs fragments cohesive regions, resulting in lower accuracy across both tasks. Furthermore, drop-to-background training is vital; it facilitates semantics-aware understanding and strictly enables selective segmentation.

	Full ↑	Selective ↑
FLEXPART	0.845	0.657
w/o drop-to-background	0.818	×
RGB, not Part IDs	0.776	0.646

Granularity control. Tab. 2 evaluates multi-granularity control on FLEXPARTBENCH, where FLEXPART outperforms all baselines at every level by simply updating the input text. In contrast, baselines are either impractical (SegviGen requires a new 2D map per level) or semantically blind (PartField specifies only region counts, not identities). Consequently, they remain rigidly tied to geometric boundaries and struggle with semantic abstraction. This is starkly evident in coarse functional grouping, where FLEXPART surpasses PartField by 0.255 mIoU. As Fig. 4 shows, modifying our text prompt seamlessly shifts partitions from broad assemblies to fine-grained accessories.

Selective part segmentation. Tab. 3 evaluates selective segmentation, where specific target parts are isolated and the rest merged into a single background region (Fig. 5). Because click-driven baselines cannot handle this semantic grouping, we supply them with ground-truth points inside each target part. Nevertheless, clicks only provide localization without semantic context. As a result, these baselines fail to capture the user's intent, struggling to separate the targets from the background cleanly. Find3D accepts text but similarly suffers from noisy boundary delineation. Our method uniquely satisfies user intent by assigning unselected geometry to a unified background region. Finally, our planner's automated name expansion recovers most of the accuracy of ground-truth descriptions.

Ablations. Tab. 4 validates our control prompt designs. First, drop-to-background training is essential for selective segmentation and improves full accuracy (0.818 to 0.845);

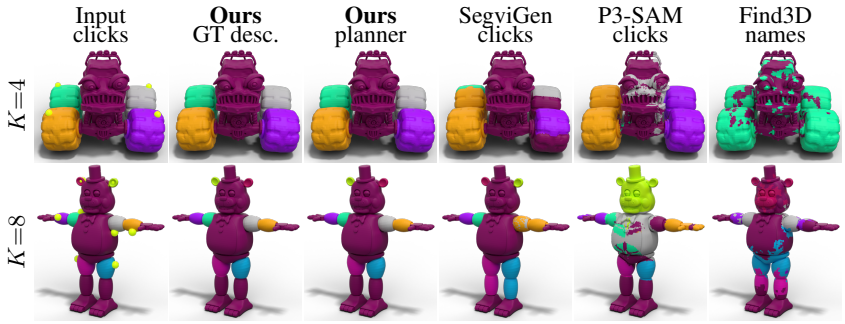


Figure 5. **Selective part segmentation.** The background parts become one brown region. Our two columns denote the same model, evaluated with either the ground-truth prompts or the planner-written descriptions. A click indicates the part the user wants to segment, which may lead to a region with ambiguous boundaries, as seen in P3-SAM and SegviGen.

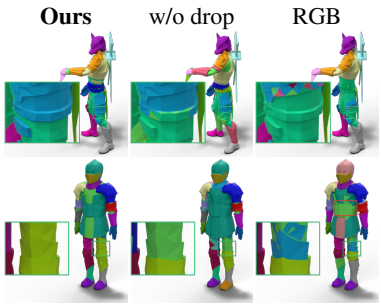


Figure 6. **Ablation study of two design choices.** The RGB palette fragments parts that part IDs keep whole (rectangles); w/o drop-to-background training reduces accuracy.



Figure 7. **Generalization to generated assets.** Four Hunyuan3D 3.1 meshes. Although these AI-generated meshes lack well-defined geometric boundaries, our method generalizes robustly by following semantic instructions rather than relying on rigid geometric priors.

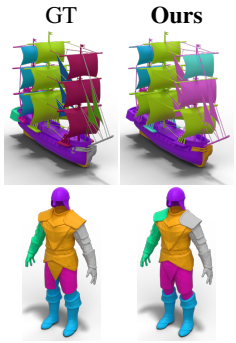


Figure 8. **Failure cases** with vague descriptions (location and shape).

explicitly prompting specific parts to the background label strengthens prompt-output alignment. Second, Part IDs strictly outperform an RGB palette. As Fig. 6 shows, continuous RGB spaces cause decoding ambiguity and surface fragmentation, whereas discrete Part IDs ensure clean, cohesive boundaries. We further ablate multi-granularity training and prompt–label order binding in App. A.2. Moreover, a fixed-geometry, fixed-K prompt ablation verifies that the model uses the requested semantics rather than only the number of regions (App. A.5).

Generalization to generated assets. To demonstrate generalization beyond artist-made assets, we evaluate on AI-generated meshes (Hunyuan3D 3.1), which present challenging topologies and boundary qualities (Fig. 7). Lacking ground-truth annotations, our automated planner autonomously constructs coarse, medium, and fine prompts relying solely on renderings and a target granularity. Granularity control transfers seamlessly to these generated assets: coarse prompts successfully separate broad functional groups, while fine prompts produce atomic decomposition.

5. Conclusion

We advance the user-centric principle for 3D part segmentation: **what you ask for is what you get**. We present PART, a unified text-controlled framework that overcomes two core obstacles: the lack of multi-granularity data and the absence of flexible, intent-driven control. We introduce a holistic prompt to dynamically define the target label space. A large-scale, VLM-annotated dataset supports this design, enabling text-guided segmentation. Generating a native 3D part-identity field conditioned on these prompts, combined with randomized background dropping, ensures the model effectively follows semantic instructions rather than rigid geometric priors. Extensive experiments demonstrate PART's significant superiority over strong baselines in both multi-granularity and selective segmentation tasks.

Limitations and future work. Fig. 8 illustrates limitations in resolving part boundaries and distinguishing similar parts. Location clauses may not uniquely identify repeated or symmetric components, merging two of a ship's five sails, and overlapping descriptions could leave the boundary between adjacent parts ambiguous, as between a knight's legs and leg armour. Even when the model recovers the intended regions, it could assign their identifiers incorrectly, as reflected in the -direct results in Tab. 1 and Tab. 2. A promising direction is an agentic workflow in which a language model and the user iteratively refine the prompt after inspecting the returned partition.

References

[1] Black Forest Labs. Flux.2: Frontier visual intelligence. https://bfl.ai/blog/flux-2, 2025. Accessed: 2026-05-02. 5

[2] Minghao Chen, Roman Shapovalov, Iro Laina, Tom Monnier, Jianyuan Wang, David Novotny, and Andrea Vedaldi. Partgen: Part-level 3d generation and reconstruction with multi-view diffusion models. *arXiv preprint arXiv:2412.18608*, 2024. 3

[3] Minghao Chen, Jianyuan Wang, Roman Shapovalov, Tom Monnier, Hyunyoung Jung, Dilin Wang, Rakesh Ranjan, Iro Laina, and Andrea Vedaldi. Autopartgen: Autoregressive 3d part generation and discovery. *arXiv preprint arXiv:2507.13346*, 2025. 3

[4] Ken Deng, Yunhan Yang, Jingxiang Sun, Xihui Liu, Yebin Liu, Ding Liang, and Yan-Pei Cao. Geosam2: Unleashing the power of sam2 for 3d part segmentation, 2025. 3

[5] Lihe Ding, Shaocong Dong, Yaokun Li, Chenjian Gao, Xiao Chen, Rui Han, Yihao Kuang, Hong Zhang, Bo Huang, Zhanpeng Huang, Zibin Wang, Dan Xu, and Tianfan Xue. Fullpart: Generating each 3d part at full resolution. *ICLR*, 2026. 3

[6] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024. 5, 7

[7] Rui Huang, Songyou Peng, Ayca Takmaz, Federico Tombari, Marc Pollefeys, Shiji Song, Gao Huang, and Francis Engelmann. Segment3d: Learning fine-grained class-agnostic 3d segmentation without manual labels. *European Conference on Computer Vision (ECCV)*, 2024. 3

[8] Team Hunyuan3D, Bowen Zhang, Chunchao Guo, Dongyuan Guo, Haolin Liu, Hongyu Yan, Huiwen Shi, Jiaao Yu, Jiachen Xu, Jingwei Huang, Kunhong Li, Lifu Wang, Linus, Penghao Wang, Qingxiang Lin, Ruining Tang, Xianghui Yang, Yang Li, Yirui Guan, Yunfei Zhao, Yunhan Yang, Zeqiang Lai, Zhihao Liang, and Zibo Zhao. Hy3d-bench: Generation of 3d assets, 2026. 2, 5

[9] Tanghui Jia, Dongyu Yan, Dehao Hao, Yang Li, Kaiyi Zhang, Xianyi He, Lanjiong Li, Jinnan Chen, Lutao Jiang, Qishen Yin, Long Quan, Ying-Cong Chen, and Li Yuan. Ultrashape 1.0: High-fidelity 3d shape generation via scalable geometric refinement. *arXiv preprint arXiv:2512.21185*, 2025. 3

[10] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023. 2, 3

[11] Zeqiang Lai, Yunfei Zhao, Zibo Zhao, Haolin Liu, Qingxiang Lin, Jingwei Huang, Chunchao Guo, and Xiangyu Yue. Lattice: Democratize high-fidelity 3d generation at scale. *CVPR*, 2026. 3, 5, 6

[12] Zeqiang Lai, Yunfei Zhao, Zibo Zhao, Xin Yang, Xin Huang, Jingwei Huang, Xiangyu Yue, and Chunchao Guo. Natex: Seamless texture generation as latent color diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18619–18629, 2026. 2, 3, 5, 6

[13] Lin Li, Haoran Feng, Zehuan Huang, Haohua Chen, Wenbo Nie, Shaohua Hou, Keqing Fan, Pan Hu, Sheng Wang, Buyu Li, and Lu Sheng. Segvigen: Repurposing 3d generative model for part segmentation. *arXiv preprint arXiv:2603.16869*, 2026. 2, 3, 4, 5, 7

[14] Manwen Liao, Xinyu Lian, Jian Mao, Kaixu Chen, Li Luo, Jinghao Yan, Wanshui Gan, Qiao Yu, Weitian Zhang, Chunhua Shen, Guang Chen, Bo Dai, Xudong Xu, and Zhaoyang Lyu. Megaparts: Scaling part-aware 3d object generation to 300 parts via token-efficient autoregressive modeling, 2026. 3

[15] Yuchen Lin, Chenguo Lin, Panwang Pan, Honglei Yan, Yiqiang Feng, Yadong Mu, and Katerina Fragkiadaki. Partcrafter: Structured 3d mesh generation via compositional latent diffusion transformers. *NeuRIPS*, 2025. 3

[16] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code, 2024. 3, 5

[17] Minghua Liu, Yinhao Zhu, Hong Cai, Shizhong Han, Zhan Ling, Fatih Porikli, and Hao Su. Partslip: Low-shot part segmentation for 3d point clouds via pretrained image-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21736–21746, 2023. 2, 3

[18] Minghua Liu, Mikaela Angelina Uy, Donglai Xiang, Hao Su, Sanja Fidler, Nicholas Sharp, and Jun Gao. Partfield: Learning 3d feature fields for part segmentation and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. 2, 3, 5, 7

[19] Changfeng Ma, Yang Li, Xinhao Yan, Jiachen Xu, Yunhan Yang, Chunshi Wang, Zibo Zhao, Yanwen Guo, Zhuo Chen, and Chunchao Guo. P3-sam: Native 3d part segmentation. *arXiv preprint arXiv:2509.06784*, 2025. 3, 5, 7, 14

[20] Ziqi Ma, Yisong Yue, and Georgia Gkioxari. Find any part in 3d, 2024. 2, 3, 5, 7

[21] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023. 3

[22] William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022. 2

[23] Ava Pun, Kangle Deng, Yiheng Zhu, Jun-Yan Zhu, Maneesh Agrawala, and Tinghui Zhou. Multicube: Compositional 3d generation with part-level semantic and spatial control. *arXiv preprint arXiv:2608.20448*, 2026. 3, 6

[24] Qwen Team. Qwen3 technical report, 2025. 5, 7, 13

[25] Qwen Team. Qwen3.8-max: A new bar for coding and cowork, 2026. 5, 14

[26] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment any-

thing in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. 3

[27] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021. 3

[28] Wanhu Sun, Zhongjin Luo, Heliang Zheng, Jiahao Chang, Chongjie Ye, Huiang He, Shengchu Zhao, Rongfei Jia, and Xiaoguang Han. Ei-part: Explode for completion and implode for refinement, 2026. 3

[29] George Tang, William Zhao, Logan Ford, David Benhaim, and Paul Zhang. Segment any mesh, 2024. 2, 3

[30] Jiaxiang Tang, Ruijie Lu, Zhaoshuo Li, Zekun Hao, Xuan Li, Fangyin Wei, Shuran Song, Gang Zeng, Ming-Yu Liu, and Tsung-Yi Lin. Efficient part-level 3d object generation via dual volume packing. *NeuRIPS*, 2025. 3

[31] Tencent Hunyuan3D Team. Hunyuan3d 2.1: From images to high-fidelity 3d assets with production-ready pbr material, 2025. 6

[32] Ardian Umam, Cheng-Kun Yang, Min-Hung Chen, Jen-Hui Chuang, and Yen-Yu Lin. Partdistill: 3d shape part segmentation by vision-language model distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 2, 3

[33] Zehan Wang, Ziang Zhang, Tianyu Pang, Chao Du, Hengshuang Zhao, and Zhou Zhao. Orient anything: Learning robust object orientation estimation from rendering 3d models. *arXiv preprint arXiv:2412.18605*, 2024. 12

[34] Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025. 5

[35] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 3

[36] Mutian Xu, Xingyilang Yin, Lingteng Qiu, Yang Liu, Xin Tong, and Xiaoguang Han. Sampro3d: Locating sam prompts in 3d for zero-shot instance segmentation. In *International Conference on 3D Vision (3DV)*, 2025. 3

[37] Xinhao Yan, Jiachen Xu, Yang Li, Changfeng Ma, Yunhan Yang, Chunshi Wang, Zibo Zhao, Zeqiang Lai, Yunfei Zhao, Zhuo Chen, et al. X-part: High fidelity and structure coherent shape decomposition and completion. *CVPR*, 2026. 3

[38] Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Edmund Y Lam, Yan-Pei Cao, and Xihui Liu. Sampart3d: Segment any part in 3d objects. *arXiv preprint arXiv:2411.07184*, 2024. 2, 3, 5, 7

[39] Yunhan Yang, Yuan-Chen Guo, Yukun Huang, Zi-Xin Zou, Zhipeng Yu, Yangguang Li, Yan-Pei Cao, and Xihui Liu. Holopart: Generative 3d part amodal segmentation. *arXiv preprint arXiv:2504.07943*, 2025. 3

[40] Yunhan Yang, Yufan Zhou, Yuan-Chen Guo, Zi-Xin Zou, Yukun Huang, Ying-Tian Liu, Hao Xu, Ding Liang, Yan-Pei Cao, and Xihui Liu. Omnipart: Part-aware 3d generation with semantic decoupling and structural cohesion. *SIGGRAPH Asia*, 2025. 3

[41] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Trans. Graph.*, 42(4), 2023. 3

[42] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics (TOG)*, 43 (4):1–20, 2024. 3

[43] Longwen Zhang, Qixuan Zhang, Haoran Jiang, Yinuo Bai, Wei Yang, Lan Xu, and Jingyi Yu. Bang: Dividing 3d assets via generative exploded dynamics. *ACM Transactions on Graphics (TOG)*, 44(4):1–21, 2025. 3

[44] Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, BIN FU, Tao Chen, Gang YU, and Shenghua Gao. Michelangelo: Conditional 3d shape generation based on shape-image-text aligned latent representation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 3

[45] Zibo Zhao, Zeqiang Lai, Qingxiang Lin, Yunfei Zhao, Haolin Liu, Shuhui Yang, Yifei Feng, Mingxin Yang, Sheng Zhang, Xianghui Yang, et al. Hunyuan3d 2.0: Scaling diffusion models for high resolution textured 3d assets generation. *arXiv preprint arXiv:2501.12202*, 2025. 3

[46] Ziming Zhong, Yanyu Xu, Jing Li, Jiale Xu, Zhengxin Li, Chaohui Yu, and Shenghua Gao. Meshsegmenter: Zero-shot mesh semantic segmentation via texture synthesis. In *European Conference on Computer Vision*, pages 182–199. Springer, 2024. 2, 3

[47] Yuchen Zhou, Jiayuan Gu, Xuanlin Li, Minghua Liu, Yunhao Fang, and Hao Su. Partslip++: Enhancing low-shot 3d part segmentation via multi-view instance segmentation and maximum likelihood estimation. *arXiv preprint arXiv:2312.03015*, 2023. 2, 3

[48] Yuchen Zhou, Jiayuan Gu, Tung Yen Chiang, Fanbo Xiang, and Hao Su. Point-SAM: Promptable 3d segmentation model for point clouds. In *The Thirteenth International Conference on Learning Representations*, 2025. 3

[49] Yiheng Zhu, Kangle Deng, Jean-Philippe Fauconnier, Inaki Navarro, Daiqing Li, Ava Pun, Yinan Zhang, Peiye Zhuang, Xiaoxia Sun, Maneesh Agrawala, Kiran Bhat, and Tinghui Zhou. Cubepart: An open-vocabulary part-controllable 3d generator. *arXiv preprint arXiv:2605.28763*, 2026. 2, 3, 5, 6, 16

[50] Zhe Zhu, Le Wan, Rui Xu, Yiheng Zhang, Honghua Chen, Zhiyang Dou, Cheng Lin, Yuan Liu, and Mingqiang Wei. Partsam: A scalable promptable part segmentation model trained on native 3d data. *arXiv preprint arXiv:2509.21965*, 2025. 3

FlexPart: Text-Controlled Semantics-Aware 3D Part Segmentation at Flexible Granularity

Supplementary Material

A. Additional Details

A.1. Annotation pipeline

This appendix describes the pipeline introduced in Sec. 3.3, in the same three stages: the partitions, the text annotation of each part, and the planner that replaces the annotation at inference.

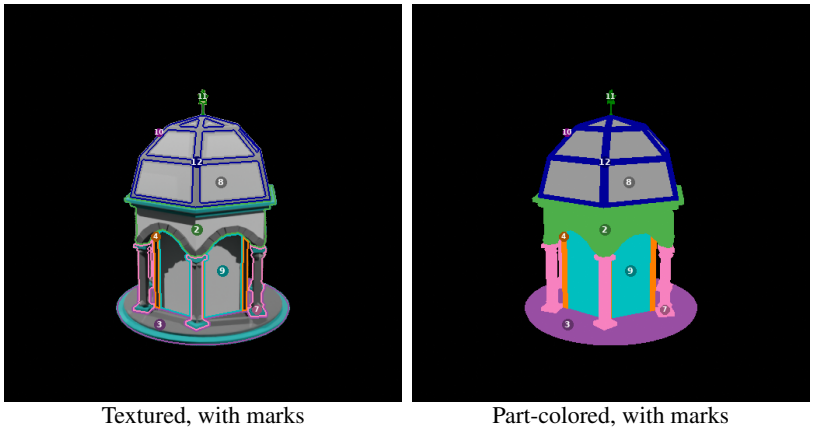
Stage 1: partitions. We take the artist-authored components of each asset as atomic units, since an artist has already split the mesh where a part boundary belongs, and the stage decides only which components belong together. Every component is marked across the sixteen views (Fig. 9) and one vision-language call returns all three levels, with the component IDs as its vocabulary:

```
The 3D asset has {n} parts (numbered 0 to {n-1}).
Produce THREE levels of semantic clustering in one pass:

COARSE -- top-level functional assemblies: each group is a complete,
  independently meaningful sub-system (e.g. "wheels", "left arm").
  Reference range: ~3-7 groups; let the object's structure decide.
MIDDLE -- natural subdivisions within each coarse group. Separate
  left/right symmetric pairs and functionally distinct sub-parts.
  If a coarse group is a single atomic part, its medium entry may be
  identical. Reference range: ~7-15 groups.
FINE   -- every semantically distinct part, with precise names.
  Reference range: ~12-20 groups.

Every part ID must appear in exactly one group at each level.
Return ONLY this JSON:
{
  "object_type": "brief description of the asset",
  "coarse": [{"name": "name", "part_ids": [0, 1]}, ...],
  "medium": [{"name": "name", "part_ids": [0]},   ...],
  "fine":   [{"name": "name", "part_ids": [0]},   ...]
}
```

The three levels are requested independently, so a coarse group is the functional grouping an artist would ask for and need not be a union of the fine groups. A level identical to another level is dropped, so an object yields up to three partitions. Each surviving level is one segmentation target: the `part_ids` of its k-th group become label k on the faces of those components.



Textured, with marks | Part-colored, with marks

Figure 9. **What the grouping call sees, per view.** Each component carries a numbered mark in both renders, so the returned groups are lists of IDs. The two renders disagree about where a boundary is only where a texture hides one, which is why both are given. Sixteen such pairs cover the object, and one call returns all three levels.

Stage 2: text annotation. This stage turns one partition into one prompt, one group per call. Fig. 10 shows a complete request. Each group is rendered from four viewpoints using three rendering modes: its boundary outlined on the whole object, which places it in context; the group on its own, which shows its shape without occlusion; and the full scene, which is the object it belongs to. Four viewpoints cover what a single view hides. The object-level caption, produced once per asset, is supplied as read-only context.

The call returns one JSON object per group, with the group ID echoed back so the answer and its geometry target stay paired:

```
{
  "SourceGroupId": "Group_01",          // echoed, must match the request
  "PartCategory":  "arm",               // high-level category of the group
  "PartName":      "right arm",         // unique within this object
  "PartPhrase":    "right arm with pauldron and gauntlet",
  "SpatialRelation": "upper right side of the torso, extending downward",
  "Shape": "a tapering cylindrical limb with a rounded shoulder plate"
}
```

The category and name must cover every recognisable component of the group instead of its most prominent one: where an upper arm, forearm, gauntlet and hand together form one limb, the category is “arm”. A second, text-only call then aggregates the per-group answers into the prompt the model trains on, mapping `PartName` to the item name, `SpatialRelation` to the location clause and `Shape` to the shape clause, in group-ID order and with the object caption as the preamble. This aggregation call receives only the per-group responses:

```
Below are per-part descriptions of one object, with their ids.
Write a single prompt: one sentence naming the object and its part count,
then the items in the given id order, each as "name: location, geometry".
Keep the given ids and their order. Introduce no detail that is not in
the input.
```




```
Object: a walker mech. It has 5 parts.
Annotate ONLY group Group_01.

For four viewpoints you are given:
  overlay    - the group outlined
               on the whole object
  solo       - the group on its own
  full scene - the object, for
               reference

Return JSON:
{
  "SourceGroupId": "Group_01",
  "PartCategory": "...",
  "PartName": "...",
  "PartPhrase": "...",
  "SpatialRelation": "...",
  "Shape": "..."
}
Cover every recognisable component
of the group, not only its most
prominent one. Echo SourceGroupId.
```

Figure 10. **One annotation call.** The images and the prompt a single Stage 2 request receives, for the `left_arm` group of a five-group coarse partition. The overlay places the group in the whole object, the solo render shows its shape without occlusion, and the full scene is the object it belongs to; four viewpoints of each cover the parts a single view hides. The call returns one JSON object, and the aggregation step maps `PartName`, `SpatialRelation` and `Shape` to the item’s three clauses.

Consistent location clauses. A clause that says “left” is only meaningful if the object has a front to be left of, and if the annotator agrees with the reader about where that front is. Turntable renderings make this easy to get wrong: the same part is on the left in one view and on the right in another. We therefore estimate, once per object, whether it has a well-defined front, using Orient-Anything [33] over the same multi-view rig the planner sees at inference. That decision fixes which vocabulary the `SpatialRelation` field may use:

```
directional    : You may use global directions -- left, right, front, back --
                 which are fixed for this object across all views.
non-directional: Use only relative terms: above, below, between, or adjacent
                 to a named part.
```

A directional object thus keeps one handedness across all sixteen views, and a symmetric one, a vase for instance, is described by relations among its own parts.

A.2. Additional ablation studies

Multi-granularity training. To evaluate the importance of multi-granularity training for semantics-aware control, we compare our approach against a baseline trained solely on fine-level partitions (Tab. 5). As observed, training with only a single segmentation pattern per geometry limits the model's flexibility. While it maintains adequate fine-level understanding, it performs poorly on coarse and medium granularities. By contrast, multi-granularity training significantly improves performance across all semantic levels—raising the average mIoU from 0.708 to 0.845. This clearly indicates that multi-granularity data is essential for empowering the model with robust, intent-driven segmentation capabilities.

Table 5. **Multi-granularity training.**

Training	Coarse ↑	Medium ↑	Fine ↑	Avg. ↑
w/o multi-granularity training	0.606	0.753	0.766	0.708
w/ multi-granularity training	0.891	0.852	0.792	0.845

Prompt–label order binding. Tab. 6 compares unordered and ordered training on the same multi-granularity data. Unordered training randomly assigns target IDs and shuffles prompt lists, capturing only global prompt-partition correlations. In contrast, ordered training aligns target IDs with their corresponding prompt positions, modeling both global partition structures and local part-to-label correlations. The results clearly show that unordered training fails to establish direct semantic-to-channel mapping, resulting in a near-zero Direct mIoU (0.011). Ordered training not only restores direct prediction capabilities (Direct mIoU of 0.804) but also achieves a higher Matched mIoU (0.845 vs. 0.767). This indicates that explicitly enforcing local semantic-latent binding intrinsically improves the model's overall understanding ability.

Table 6. **Ordered and unordered training.**

Training	Matched mIoU ↑	Direct mIoU ↑
Unordered	0.767	0.011
Ordered	0.845	0.804

A.3. Implementation details

Architecture. The denoiser is an MM-DiT with 12 double-stream and 24 single-stream blocks, hidden size 1536 and 12 heads. Text is encoded by a frozen Qwen3-VL-4B-Instruct [24], whose last hidden state is projected from 2560 to 1536 by a trainable RMSNorm and linear layer under a 1024-token budget. The part-ID field is encoded by the part-aware VAE of Sec. 3.2 at a resolution of 128, and geometry and prompt tokens are indexed in one sequence by the rotary encoding of Sec. 3.2. Text is dropped for 10% of training samples, which provides the unconditional branch for classifier-free guidance.

Training. AdamW, base learning rate 10^{-4} with $\beta = (0.9, 0.99)$, $\epsilon = 10^{-6}$ and weight decay 10^{-2}, a cosine schedule with $1{,}000$ warmup steps, gradient clipping at 0.5, in fp32. The text projection trains at 5×10^{-5} while the denoiser trains at 10^{-4}. We train for 125K steps. At inference we use the flow-matching Euler sampler with 30 steps and guidance scale 6.0.

Prompt perturbations. Sec. 3.3 introduces drop-to-background; a second perturbation is applied alongside it, and both act on the prompt. *Drop-to-background* is applied to every training sample, dropping a fraction of the items drawn uniformly up to 0.65 and reassigning their faces to bg. *Part-list shuffling* permutes the items and relabels the target accordingly, so an index denotes the item's position in the prompt list.

What the planner sees. The planner is Qwen3.8 [25], held frozen; it is a separate model from the Qwen3-VL-4B-Instruct that encodes the prompt inside FLEXPART. Its visual input is the geometry alone, sixteen turntable renderings at 512^2 of the bare mesh under a neutral matcap shading (Fig. 11).

It is used in two regimes, differing in what the user supplies. When the user states the part names, as in the FLEXPART-Auto row of Tab. 1 and the planner row of Tab. 3, the planner completes each name into a full item by writing its location and shape clauses, in the field order the annotation stage of App. A.1 would have produced, which FLEXPART consumes unchanged; the number of parts is then whatever the user listed. For a selective request it also appends the background items covering the remainder of the surface, which the user does not enumerate. When no part names are specified, as for the generated assets of Fig. 7 where no ground-truth decomposition exists, the planner receives only the renderings and a target granularity and decides the inventory itself, so it is not told how many parts to return.

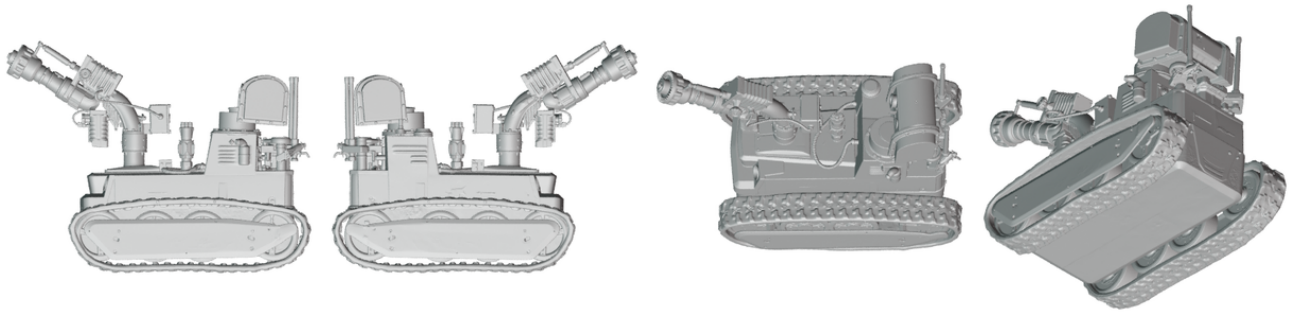


Figure 11. **What the visual planner is given.** Four of the sixteen turntable views, 90° apart, of the bare mesh; these are its only visual input, alongside the user's command. This object is also the first row of Fig. 7.

A.4. Additional experiments

Benchmark construction. Sec. 4.1 names the three evaluation sets; this paragraph says how each was built. For **PartObjaverse-Tiny** we follow the watertight protocol of P3-SAM [19], which introduces a watertight version of this benchmark and reports baselines on it: every asset is converted once, each new face taking the label of its nearest original face, and that mesh is the only geometry any method sees. It is the stricter setting, since the original connected components correlate with the parts and can be exploited whereas a watertight surface has neither connectivity nor color. On this benchmark our pipeline supplies text only: the part decomposition released with PartObjaverse-Tiny is used without modification, and every ground-truth region in Tab. 1 and Tab. 8 is a set of its original parts, so no part boundary is of our making. What our pipeline adds is the object caption and the name, location and shape clauses each region is referred to by, which the original release does not provide and which text-conditioned segmentation requires. **Selective-190** is built by hand from 28 FLEXPARTBENCH objects: for each case we choose a subset of the annotated parts as the named items and assign the remainder to background, so the ground truth is the benchmark's own annotation restricted to that subset rather than a new labelling. The number of named parts per case runs from two to eighteen, concentrated on the even values between two and ten. FLEXPARTBENCH itself is described below.

Constructing FLEXPARTBENCH. FLEXPARTBENCH is assembled in four stages. We first shortlist candidates from PartNeXt and PartVerse-XL, scoring each asset on whether its artist-authored components support three distinct levels, and sample 400 objects in the category proportions of the source libraries. Sampling targets the fine part count as well: most objects are asked to land between 20 and 28 fine parts, with a capped tail above that, so the benchmark is not dominated by either trivial or extreme decompositions. Second, every asset passes the grouping and annotation pipeline of App. A.1, and each of its three levels is checked automatically against a granularity contract: the three counts must strictly increase, the three partitions must be pairwise distinct, and levels with excessive singleton groups, a dominant catch-all group, or micro-fragmented fine partitions are rejected. Because the contract requires all three levels to survive, every object kept in FLEXPARTBENCH carries exactly three partitions, whereas a training object yields up to three once duplicate levels are dropped (Sec. 3.3). No asset carried a hard error at this stage; the automatic checks instead flagged 58 of the 400 as borderline for a human to read.

Third, and this is what the benchmark rests on, the authors reviewed the annotations by hand — every borderline case, and the accepted ones by sampling — against the rendered partitions rather than the text alone. We removed 23 objects outright: 21 whose source geometry or component labelling was itself defective, which no annotation could repair, and 2 more the reviewer

Table 7. **Category distribution of FLEXPARTBENCH** over its 364 objects, following the proportions of the source libraries.

Category	#	Category	#
Human	91	Building	16
Weapon and tool	66	Furniture and home	15
Vehicle	58	Plant	10
Daily container	36	Instrument, other	5
Electronics	32	Food	5
Animal	30	**Total**	**364**

judged mislabelled. A further 13 were dropped for having more fine parts than the sampling policy allows, leaving 364 objects and 1,092 annotated partitions (152 from PartNeXt, 212 from PartVerse-XL). Tab. 7 reports the category distribution; the fine level carries between 10 and 28 parts with a median of 23. Fourth, the set was frozen and every UID checked against the training pool.

The filtering and the freeze both predate any model run on this benchmark, so no selection here could have been informed by which cases our method gets right.

Baseline protocols. Each baseline is run with its released weights in the regime it was designed for; Sec. 4.1 says what each is conditioned on and this paragraph says how. **SegviGen** releases three checkpoints, one interactive and two for full segmentation with and without 2D guidance. Both of its Tab. 1 arms supply a guidance map, so both use the 2D-guidance checkpoint; Tab. 3 uses the *interactive* checkpoint, since a click on each named part is the closest equivalent it has to a selective input. Its predicted-map arm queries SAM 3 on a single front view with an English translation of the annotated part names, index-locked to the prompt. **SAMPart3D** requires one test-time optimization per object, 5,000 steps, and emits all of its scales from that single run; the scale-to-granularity correspondence was fixed on a separate calibration set and held there for both benchmarks. **Find3D** resamples the mesh to 5,000 points internally, which bounds the boundary precision it can reach, and decodes a multi-query panoptic argmax. **PartField** clusters its feature field to a part count we supply from the ground truth. **P3-SAM** is run at its native part count. Every text-conditioned method, Find3D and both of our rows included, receives the same English part names.

Background leakage on Selective-190. Tab. 3 reports Query IoU and BG IoU. A third quantity, the fraction of ground-truth background area a method predicts as some named part, measures the same failure from the other side and is what makes the BG IoU column interpretable. FLEXPART leaks 0.024 of the background into its named parts, and Find3D 0.171. The comparison worth making is against a model trained without drop-to-background, which leaks 0.976: it partitions the whole surface as usual and its bg label carries no meaning, which is the ablation Tab. 4 marks $\times$.

A.5. Does the model use semantics beyond part count?

A potential shortcut is that the model might infer the desired granularity solely from the number of prompt items, K, while ignoring their actual semantic content. To investigate this, we evaluate on the PartObjaverse-Tiny benchmark (Sec. 4.1), strictly controlling for input geometry, target partition, K, model checkpoint, sampler, and random seed. Using the FLEXPART-GT results from Tab. 1 as our full-prompt baseline, we compare the original control prompt for each object against two alternatives: (1) a list of K generic placeholders, and (2) a mismatched prompt sampled from a fixed derangement of genuine annotations with the same K. For the mismatched prompts, we replace the entire text—including the object caption, part names, location clauses, and shape clauses—while keeping K unchanged. This ensures the prompt retains a natural linguistic structure and annotation format, confirming that any observed effects are not merely artifacts of an unusual placeholder format. For instance, a three-item prompt for an orc figurine is replaced by a genuine three-item sailboat prompt. For pools containing only a single prompt of length K, we borrow genuine same-K text from another benchmark partition where possible; for the single global-maximum K, we apply a fixed cyclic permutation to its item descriptions. Excluding this specific edge case does not alter our overall conclusions. Finally, we aggregate the evaluation metrics per object and report the average across all objects.

Removing semantic content causes a large drop even though K is unchanged. More importantly, a genuine but mismatched same-K prompt also reduces mIoU substantially. The predicted partition therefore depends on the requested semantic content rather than acting only as a geometrically plausible K-way decomposition. This test establishes sensitivity to semantic conditioning; it does not evaluate a second, independently valid decomposition of the same object.

Table 8. **Semantic conditioning beyond part count.** Geometry and the requested number of regions K are fixed. The mismatched condition uses genuine same-K annotation text with semantics that do not match the target grouping. We report raw area-weighted mIoU without post-processing, following Sec. 4.1.

Prompt condition	mIoU ↑	Δ
Full semantic prompt	0.691	–
K-only placeholders	0.373	−0.318
Mismatched semantics, same K	0.493	−0.198

A.6. Comparison with CubePart

CubePart [49] conditions a generative model on an open-vocabulary part schema. Its second stage can decompose an existing mesh, which makes it the closest schema-conditioned method to our setting, and its code is released. A quantitative comparison under our metric is nonetheless not available: CubePart synthesizes a separate mesh for each requested part rather than labelling the faces of the input, so there is no per-face prediction to score without first projecting its output back onto the input surface, a step of our choosing rather than theirs. Fig. 12 therefore compares the two visually, on the same watertight mesh and with semantically matched part names. Each method receives prompts in its required format, at both coarse and fine granularities.

CubePart represents each requested item with its own latent and generates its geometry separately. The number of generated latent groups therefore grows with the schema, and the resulting part meshes must jointly reconstruct the input object. In these examples, that reconstruction becomes less stable as the requested decomposition becomes more detailed. On the second object the coarse schema names *both arms* and *both legs* as separate parts, yet CubePart's generated leg surface absorbs the arms, and at the fine schema the right hand comes back as a fragment. The third object shows the same trend at the scale our benchmark asks for: the coarse request returns a clean decomposition, while the detailed request produces heavily fragmented part geometry despite passing CubePart's own validity check.

We treat these three objects as case studies rather than an estimate of CubePart's overall failure rate. The displayed outputs show missing surface regions, geometry extending beyond the input shape, and parts breaking into disconnected fragments. Our formulation avoids this geometry-generation failure mode by retaining the input surface and predicting only its face labels.

A.7. Broader impact

Positive impact. An explicit part decomposition is what part-aware workflows need: localized editing, rigging and retopology, and semantic reuse of an asset's components. Because the control signal is text, a user obtains one by naming the parts, and a frozen vision-language model can write the prompt from renderings, which puts the capability within reach for education, indie development and rapid prototyping. We label the faces of the input mesh to obtain a part decomposition of the user's asset.

Negative impact and mitigation. Like any generative 3D model, FLEXPART can be misused, for instance to decompose and repurpose a copyrighted asset without attribution. It also inherits the biases of its training data, which over-represents common object categories and the design conventions of the artists who built them, so both its part vocabulary and its notion of what counts as one part are skewed towards those categories. We train only on license-compliant data, we recommend that downstream users attribute reused assets and check outputs for intellectual-property concerns before deployment, and we do not intend the method for identity-related or surveillance applications.

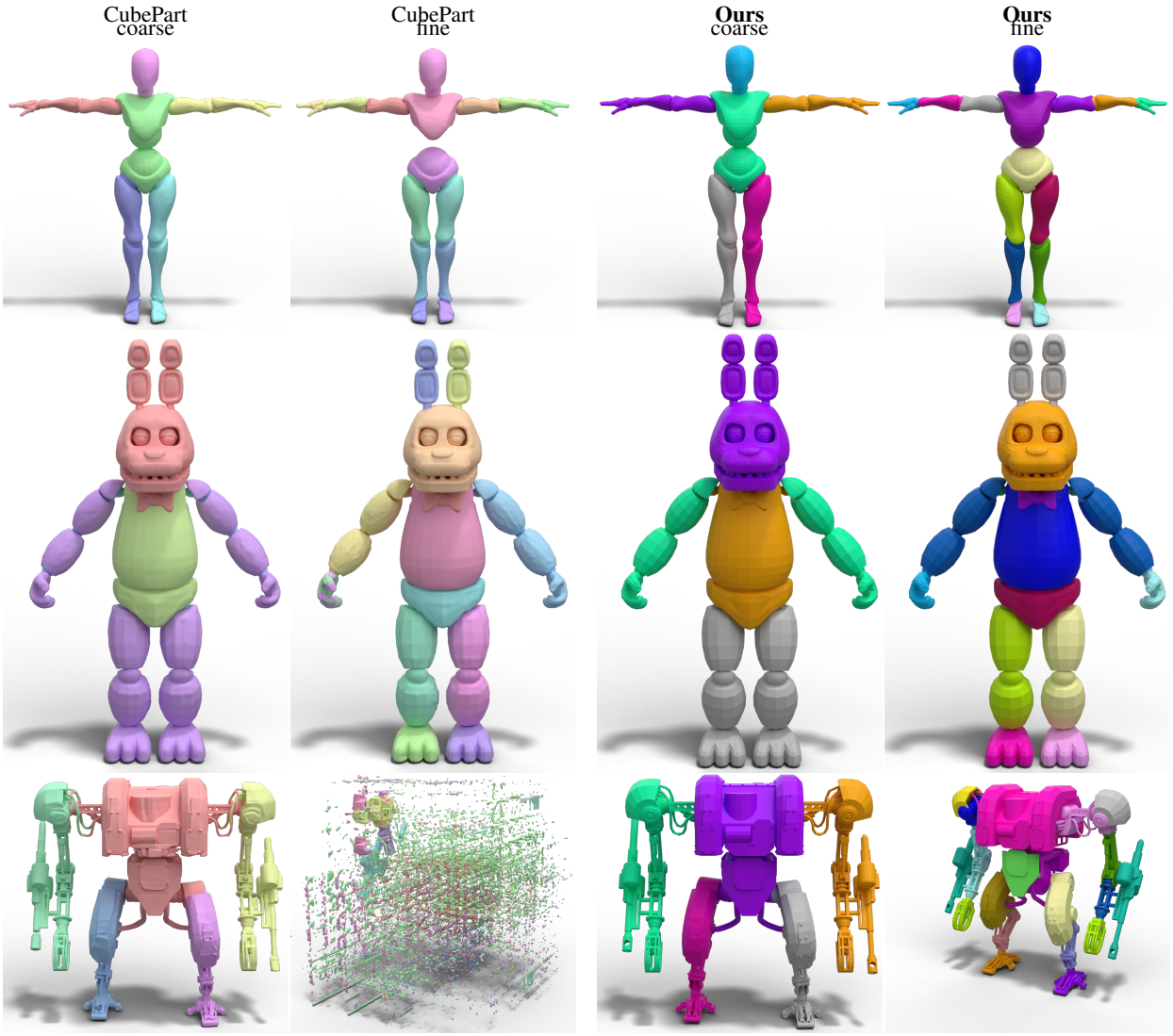


Figure 12. **Following the prompt on an existing mesh.** Both methods receive the same watertight meshes and part names for coarse and fine decompositions. CubePart allocates and generates an independent latent geometry for each requested item, whereas our method retains the input surface and predicts its face labels. The displayed cases illustrate the resulting differences in partition and surface preservation. *Top:* At the fine level, CubePart fails to generate the waist, leaving the chest and pelvis disconnected; in contrast, our method continuously labels the original surface. *Middle:* The coarse prompt requests *both arms* and *both legs* as separate items, but CubePart erroneously merges them (**purple**). Our method correctly separates the arms (**green**) and legs (**grey**). At the fine level, CubePart generates the right hand as fragmented specks. *Bottom:* CubePart handles the coarse request successfully but its fine-grained output (armour plates, brackets, hoses) becomes a cloud of fragments. Our method instead assigns labels on the unchanged input surface, yielding a complete partition without post-processing.