LLMSEG: TRAINING-FREE TEXT-PROMPTED IMAGE SEGMENTATION WITH B-SPLINES

Anonymous authors
Paper under double-blind review

ABSTRACT

Promptable segmentation models such as SAM 3 achieve remarkable open-vocabulary performance but require dedicated architectures, large-scale annotation engines, and task-specific training. We ask a different question: can a general-purpose multimodal large language model (MLLM) segment images *directly*, without an added segmentation head, mask decoder, or training? We present LLMSeg, a training-free pipeline that turns a vision-capable LLM into a text-prompted image segmenter. LLMSeg decomposes segmentation into three LLM-driven stages. First, *detect* localizes all instances matching a free-form text query and describes their coarse morphology. Next, *segment* traces each instance using closed periodic B-splines confined to its detected box. These splines define an outer boundary and optional hole loops. Finally, *refine* iteratively proposes focus boxes around residual errors and retraces confined splines. The process ends when the mask stabilizes or the pass budget is exhausted. The output is a binary mask with inspectable intermediate geometry. We evaluate LLMSeg across ten referring and reasoning segmentation splits (ReasonSeg and RefCOCO/+/g). We also evaluate open-vocabulary instance segmentation (LVIS, COCO, and SA-Co), semantic segmentation, and zero-shot detection (ODinW13 and RF-100VL). With an off-the-shelf Gemini-3.7-Flash backend, LLMSeg is competitive with and often surpasses the specialist SAM 3 model on identical items. It reaches 69.6 gIoU on ReasonSeg val without additional segmentation-specific training. Backend model updates can improve accuracy without retraining the pipeline.

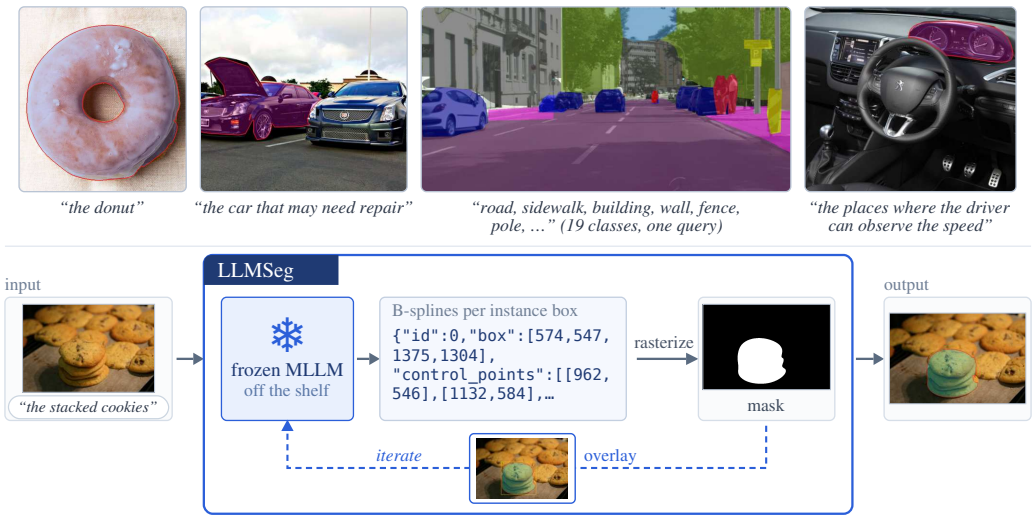


Figure 1: **Text-prompted segmentation from a frozen MLLM.** LLMSeg segments an image from a free-form text query alone, with a frozen multimodal LLM as its only vision component: no segmentation head, no mask decoder, no training. *Top:* four results, each under its query. *Bottom:* the model emits geometry rather than pixels; deterministic code rasterizes it, and the rendered overlay is fed back so the model can inspect and correct its own boundary.

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.

1 INTRODUCTION

Text-prompted segmentation (producing a pixel mask for every instance that matches a language query) is a foundational capability for multimodal AI. The dominant approach builds ever-stronger *specialist* models: the SAM series (Kirillov et al., 2023; Ravi et al., 2024) introduced promptable visual segmentation. SAM 3 (Carion et al., 2026) extends this capability to promptable *concept* segmentation from noun phrases. It relies on a dedicated detector-tracker architecture and a massive human- and model-in-the-loop data engine.

In parallel, multimodal large language models (MLLMs) have acquired remarkable general visual competence. Their capabilities include grounding, counting, and coarse localization (Comanici et al., 2025; OpenAI, 2023; Bai et al., 2025; Deitke et al., 2025). However, they are widely assumed to require a trained segmentation decoder, such as SAM, to produce dense pixel masks (Lai et al., 2024; Rasheed et al., 2024; Carion et al., 2026). We challenge this assumption. We present LLMSeg (Figure 1), a fully training-free pipeline. A general-purpose MLLM is its *only* vision component. The model never outputs pixels; instead, it outputs *geometry*. LLMSeg decomposes segmentation into three stages. First, *detect* localizes instances matching the query as boxes. Second, *segment* traces each instance with closed periodic B-spline loops confined to its box. Third, *refine* inspects the rendered mask overlay and re-traces residual errors within signed *focus boxes*. Deterministic code rasterizes splines and enforces box confinement.

This design provides an explicit *parametric* intermediate representation: cubic traces encode smooth curves, while degree-one traces encode straight edges. Rasterization and refinement need not preserve a smooth final boundary. With topology enabled, holes are specified as subtractive loops. Reasoning queries ("the food that the bird likes to eat") work off the shelf because each query is interpreted by the same LLM that performs the segmentation. Every stage emits human-readable JSON, making geometric decisions inspectable without claiming a faithful explanation of the model's reasoning. Since LLMSeg adds no trained parameters, backend upgrades require no pipeline retraining, though accuracy gains are model-dependent.

Empirically (Section 4.2), LLMSeg with an off-the-shelf Gemini-3.7-Flash backend is competitive with the specialist SAM 3 (Carion et al., 2026) when evaluated on identical items. We compare the methods across four task families: referring and reasoning segmentation (ReasonSeg, RefCOCO/+/g), open-vocabulary instance segmentation (LVIS, COCO, SA-Co/Gold), semantic segmentation (Cityscapes, ADE20K), and zero-shot in-the-wild detection (ODinW13, RF-100VL). On ReasonSeg val, LLMSeg reaches 69.6 gIoU zero-shot. This score exceeds that of every published trained specialist in our comparison. Unlike SAM 3, LLMSeg uses *only* a text query. It uses no image exemplars, box prompts, or video. This asymmetry makes the comparison conservative for LLMSeg.

Contributions: **(i)** We show for the first time at benchmark scale that a general-purpose MLLM can perform dense text-prompted segmentation without any *trained segmentation component*. Our approach uses a detect-segment-refine loop over parametric B-spline geometry. **(ii)** We introduce the technical machinery needed for reliable operation. It includes box-confined control points, one batched call per stage over all instances, signed focus-box refinement, and box-relative convergence. **(iii)** We evaluate LLMSeg against the SAM 3 and SAM 3 Agent baselines from Carion et al. (2026). Our evaluation covers ten referring/reasoning splits and five additional benchmarks. We analyze where the LLM route wins (reasoning, rare domains, presence judgment, and boundary quality) and where it loses (thin structures and amorphous classes).

2 RELATED WORK

Segmentation models and MLLM-driven segmentation. SAM (Kirillov et al., 2023) introduced promptable segmentation. SAM 2 (Ravi et al., 2024) extended it to video. SAM 3 (Carion et al., 2026) extended it to open-vocabulary concept prompts. GLIP (Li et al., 2022b), GroundingDINO (Liu et al., 2024a), OWLv2 (Minderer et al., 2024), and DINO-X (Ren et al., 2025) are trained specialists. MLLM-based methods add language reasoning but retain a trained mask component. LISA (Lai et al., 2024) decodes an LLM's `<SEG>` token with SAM. GLaMM (Rasheed et al., 2024), PSALM (Zhang et al., 2024c), and X-SAM (Wang et al., 2025a) also use trained mask components. SAM4MLLM (Chen et al., 2024) and Seg-Zero (Liu et al., 2025) train MLLMs to prompt SAM.

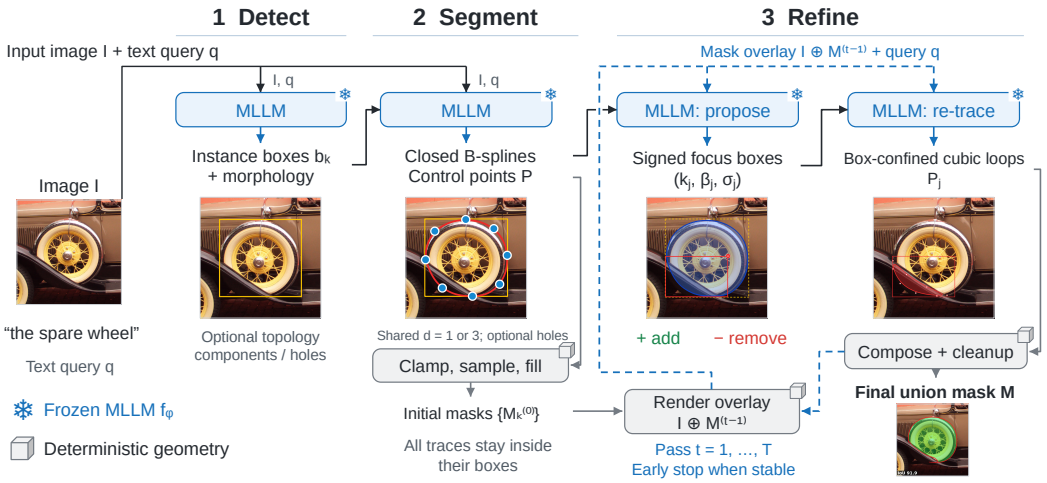


Figure 2: **LLMSeg pipeline overview.** Detection localizes instances, segmentation traces box-confined B-splines, and refinement corrects the mask within signed focus boxes (+ adds pixels, − removes them). Each MLLM call serves all instances; one is shown.

Training-free alternatives include RSVP (Lu et al., 2025) and attention-based grounding (Kang et al., 2025). Agentic systems include SAM 3 Agent (Carion et al., 2026) and Colon-Bench (Hamdi et al., 2026). These approaches still delegate mask production to a segmentation model. In contrast, LLMSeg uses only text queries to a frozen, general-purpose MLLM. It requires no exemplars, input boxes, or video prompts. The MLLM emits the final geometry, which our loop revises directly.

Geometry as text and contour representations. Representing geometry as text is not new. Vision-LLM (Wang et al., 2023) represents boundaries as short polygons in a trained multitask pipeline. VisionLLM v2 (Wu et al., 2024a) uses task-specific trained decoders. PixelLLM (Xu et al., 2024) learns localization heads and obtains masks through SAM. Text4Seg (Lan et al., 2025) generates segmentation tokens but requires fine-tuning. General models such as Molmo (Deitke et al., 2025) and Gemini (Comanici et al., 2025; Voigtlaender et al., 2025) also provide pointing and coarse localization. Classical boundary representations include spline snakes (Brigger et al., 2000). Trained vertex predictors include Polygon-RNN++ (Acuna et al., 2018) and Curve-GCN (Ling et al., 2019). LLMSeg uses this output space without training a contour predictor. A frozen, general-purpose backend emits control points that define the mask prediction. Only deterministic geometric processing follows. Confined signed regions then allow the refinement process to correct local parts of the mask.

3 Method

Overview. Given an image I and a free-form text query q, LLMSeg predicts binary masks for N retained instances. We index these instances by $k = 1, \ldots, N$. Stage 2 initializes each mask as $M_k^{(0)}$. Refinement pass t produces $M_k^{(t)}$. Their union is $M^{(t)} = \bigcup_{k=1}^{N} M_k^{(t)}$. The final output M is the union at termination. A frozen multimodal LLM f_ϕ is the only vision component. No parameters are updated, and no segmentation head is attached. The pipeline accesses f_ϕ only through fixed text prompts, with one prompt per stage call. Each call is stateless and uses no tools. Deterministic code controls the sequence. It also turns the returned geometry into masks. Figure 2 shows the three stages. The prompt templates are given in Section F.

3.1 Stage 1: Detecting instances and their structure

A single detection call to f_ϕ takes I and q as input. For every instance matching q, it returns an axis-aligned box $b_k = [x_{k,1}, y_{k,1}, x_{k,2}, y_{k,2}]$ and a confidence score. It also returns a short *morphology* description of the coarse shape and aspect. Optionally, it provides a *topology* descriptor that specifies the component and hole counts. Neither is used for localization; both condition Stages 2 and 3.



input LLMSeg input LLMSeg

"In an educational setting, children often use different materials to learn about letters, numbers, and words. What object in the picture could be used as a visual aid for learning about letters and words?"

"The general steps for brewing tea are to put tea leaves into a teacup and then pour hot water over them. What container in the picture is most likely to be used next for pouring hot water to make tea?"

Figure 3: **Reasoning queries work off the shelf.** Two ReasonSeg items segmented by LLMSeg with Gemini-3.7-Flash. The same model reads the query and segments, with no agent wrapper or query rewriting.

Topology is a switch rather than a load-bearing part of the method. It enables Stage 2 to carve holes explicitly. Disabling it leaves split-level quality within the run-to-run band (Section 5). Near-duplicate boxes are merged at IoU ≥ 0.9. Instances are then ordered in reading order. At most $N_{\max}$ are kept, so $0 \leq N \leq N_{\max}$. Coordinates are given in absolute pixels. The origin is at the top left. In set operations, b_k denotes the pixel region inside that box. If $N = 0$, the output is empty.

3.2 Stage 2: From Language to Geometry

One batched call traces every instance at once. It returns an outer loop for each instance. With topology conditioning, it also returns one loop per predicted hole, each a list of control points.

Representation. A loop is a *closed periodic B-spline* of degree d. It is defined over a control polygon $P = (p_1, \ldots, p_n)$ containing n two-dimensional control points indexed by i. The count n can vary between loops. The polygon is wrapped so that $p_{i+n} = p_i$:

$$C_{P,d}(u) \;=\; \sum_{i=1}^{n} B_{i,d}(u)\, p_i, \qquad u \in [0, n], \quad C_{P,d}(n) = C_{P,d}(0), \tag{1}$$

where u is the curve parameter and $B_{i,d}$ are the uniform periodic B-spline basis functions of degree d (Piegl & Tiller, 1997). Eq. (1) is a *piecewise* polynomial consisting of n degree-d segments. Each $B_{i,d}$ is supported on only $d{+}1$ knot spans. Thus, each control point influences only $d{+}1$ segments, and a misplaced point changes a bounded arc rather than the whole curve. Stage 3 does not edit these original control points. Its locality comes from adding or subtracting focus-box regions.

We also let the model choose the degree. The same call that places the control points first chooses $d \in \{1, 3\}$ based on the object's form. It uses $d = 3$ for rounded and organic silhouettes and $d = 1$ for angular, straight-edged ones. It then places the points as the selected degree requires. Cubic control points are pushed outward toward the edge, whereas degree-1 vertices lie directly on it. One degree is shared by all loops of all instances in the stage-2 call.

Rasterization by sampling. Masks are produced by sampling the curve and filling the resulting polygon, not by evaluating an implicit function. Let clip_b denote the per-coordinate clamp onto a box b. Let Fill denote even-odd polygon fill (Foley et al., 1996). One loop then becomes the region

$$\mathcal{R}_{b,d}(P) \;=\; \mathrm{Fill}\Big(\big(\mathrm{clip}_b(C_{\mathrm{clip}_b(P),d}(u_r))\big)_{r=1}^{m}\Big), \qquad u_r = \tfrac{r-1}{m-1}\, n, \tag{2}$$

with m samples indexed by r, uniformly spaced in the curve parameter and including the repeated endpoint. For cubic loops, m is proportional to the perimeter of b, so small and large objects are rasterized at comparable boundary resolution. For $d = 1$, the control polygon is filled directly. Eq. (2) applies Fill to an ordered sequence of boundary samples. Controls are clamped before tracing. As a defensive geometry check, the sampled boundary is also clamped before filling. A trace can never leave its own detected box, so a hallucinated control point cannot paint across the image. Overlapping instance boxes can still produce overlapping masks. Consequently, detection quality upper-bounds segmentation quality. We quantify this effect in Section 5.

With topology conditioning off, the initial instance mask is $M_k^{(0)} = \mathcal{R}_{b_k,d}(P_k^{\mathrm{out}})$, where P_k^{out} is instance k's outer control polygon. When topology conditioning is enabled, regions rasterized from

Method	MLLM/Backend	ReasonSeg		RefCOCO			RefCOCO+			RefCOCOg	
		val	test	val	testA	testB	val	testA	testB	val (U)	test (U)
Seg-Zero	Qwen2.5-VL 7B	62.0	52.0	–	–	–	–	–	–	–	–
LISA	LLaVA 7B	–	–	74.9	79.1	72.3	65.1	70.8	58.1	67.9	70.6
LISA	LLaVA1.5 13B	72.9	62.2	–	–	–	–	–	–	–	–
GLaMM	Vicuna 7B	47.2	–	79.5	83.2	76.9	72.6	78.7	64.6	74.2	74.9
PSALM	Phi1.5 1.3B	–	–	83.6	84.7	81.6	72.9	75.5	70.1	73.8	74.4
RICE	Qwen2.5-7B	–	–	83.5	85.3	81.7	79.4	82.8	75.4	79.8	80.4
HyperSeg	Phi2 2.7B	56.7	–	84.8	85.7	83.4	79.0	83.5	75.2	79.4	78.9
X-SAM	Phi3 3.8B	32.9	41.0	85.1	87.1	83.4	78.0	81.0	74.4	83.8	83.9
Grounded SAM	–	14.5	16.4	–	–	–	–	–	–	–	–
Gemini Seg	Gemini2.5 Flash	13.3	9.2	–	–	–	–	–	–	–	–
SAM 3*	–	9.9	11.3	42.0	42.8	44.2	33.6	37.0	32.9	40.0	39.7
LISA	LLaVA1.5 13B	60.3	50.8	–	–	–	–	–	–	–	–
Kang et al.	LLaVA1.5 13B	60.5	49.9	**76.1**	**78.9**	72.8	64.1	67.1	57.3	67.7	69.0
RSVP	GPT-4o	63.1	60.0	–	–	–	–	–	–	–	–
SAM 3 Agent	Qwen2.5-VL 7B	49.1	53.5	59.4	64.3	55.0	51.4	57.0	44.9	57.2	58.8
SAM 3 Agent	Qwen2.5-VL 72B	65.1	64.0	71.6	74.9	66.0	64.9	70.8	57.9	68.8	70.2
SAM 3 Agent	Llama4 Maverick	61.5	60.9	71.7	76.5	66.5	65.1	71.2	57.6	67.7	68.7
SAM 3 Agent	Qwen3-VL 8B Think	57.4	67.3	68.6	72.3	63.9	59.3	64.5	55.6	66.4	66.9
SAM 3 Agent	Qwen3-VL 235B Think	67.0	71.8	73.9	77.0	69.7	66.9	70.9	62.2	72.1	72.9
SAM 3 Agent	Gemini2.5 Pro	66.0	**72.1**	75.5	77.6	71.0	67.3	71.1	63.4	73.4	74.0
SAM 3 Agent*	Gemma-4-31B	63.9	66.7	73.2	75.6	75.0	67.2	**72.0**	61.9	72.1	74.7
SAM 3 Agent*	Gemini-3.7-Flash	62.0	63.2	73.2	76.7	75.5	**68.4**	**72.0**	**68.5**	**76.5**	**76.9**
LLMSeg (w/o refine)	Gemini-3.7-Flash	64.6	58.2	65.5	65.7	72.9	59.7	61.4	61.1	64.8	69.3
LLMSeg (full)	Gemini-3.7-Flash	**71.5**	65.8	71.2	73.8	**76.0**	67.0	69.9	64.9	70.9	74.5

Table 1: **Reasoning and referring expression segmentation.** Results on ReasonSeg (Lai et al., 2024) and RefCOCO/+/g (Kazemzadeh et al., 2014; Mao et al., 2016). All values are cIoU. The baselines are Seg-Zero (Liu et al., 2025), LISA (Lai et al., 2024), GLaMM (Rasheed et al., 2024), PSALM (Zhang et al., 2024c), RICE (Xie et al., 2025), HyperSeg (Wei et al., 2024), X-SAM (Wang et al., 2025a), Grounded SAM (Ren et al., 2024), Gemini Seg (Voigtlaender et al., 2025), Kang et al. (Kang et al., 2025), RSVP (Lu et al., 2025), and both SAM 3 and SAM 3 Agent (Carion et al., 2026). The *top block* contains methods fine-tuned for referring or reasoning segmentation. Their results are never bolded. Unmarked rows are published full-split numbers quoted from Carion et al. (2026). * Our baseline evaluation uses the complete ReasonSeg splits. On RefCOCO/+/g, it uses the same 500-item random subset per split (seed 0) as our shaded rows. Elsewhere, **bold** is the best and underline the second best per column.

predicted hole loops are subtracted from the outer region to obtain $M_k^{(0)}$. The predicted hole count is not guaranteed to survive loop overlap, rasterization, refinement, or cleanup.

An instance with an empty trace is kept at zero coverage, and Stage 3 can still grow it from its box.

3.3 Stage 3: Iterative Refinement

The initial outline often identifies the right object but misses boundary detail. Each refinement pass asks the model to locate errors on the current mask overlay. It then retraces those regions within smaller boxes. In pass $t = 1, \ldots, T$, both calls receive $I \oplus M^{(t-1)}$, the previous union mask rendered over the image. Leaving the fixed prompts and structural context implicit, we write the loop as

$$\begin{aligned} \{(k_j, \beta_j, \sigma_j)\}_j &= f_\phi(I \oplus M^{(t-1)}, q), \\ M_k^{(t)} &= \mathrm{Compose}\Big(M_k^{(t-1)}, \{(\sigma_j, \mathcal{R}_{\beta_j,3}(P_j))\}_{j:\, k_j = k}\Big). \end{aligned} \tag{3}$$

Here j indexes the focus boxes in the current pass. The value k_j identifies the instance corrected by the jth focus box. The box β_j is confined to that instance's box. The sign $\sigma_j \in \{+1, -1\}$ requests addition or removal. A second call uses the same overlay, query, and box geometry. It returns cubic control polygons P_j for all retained boxes in one batch. Before this call, an effect gate discards boxes with negligible potential to change the mask. If none remain, retracing is skipped.

Composition applies each correction to instance k_j and cleans up small artifacts. It carries a per-instance commitment field that is implicit in Eq. (3). This field resists opposing edits and reduces oscillation. Boolean composition and cleanup need not leave a globally smooth spline boundary. The updated union $M^{(t)}$ feeds the next pass. Both model calls batch all instances, so each pass uses at most two MLLM calls regardless of N. The gating and cleanup settings, the per-pixel composition rule, and a worked example of each call are given in Section B. Refinement ends after at most

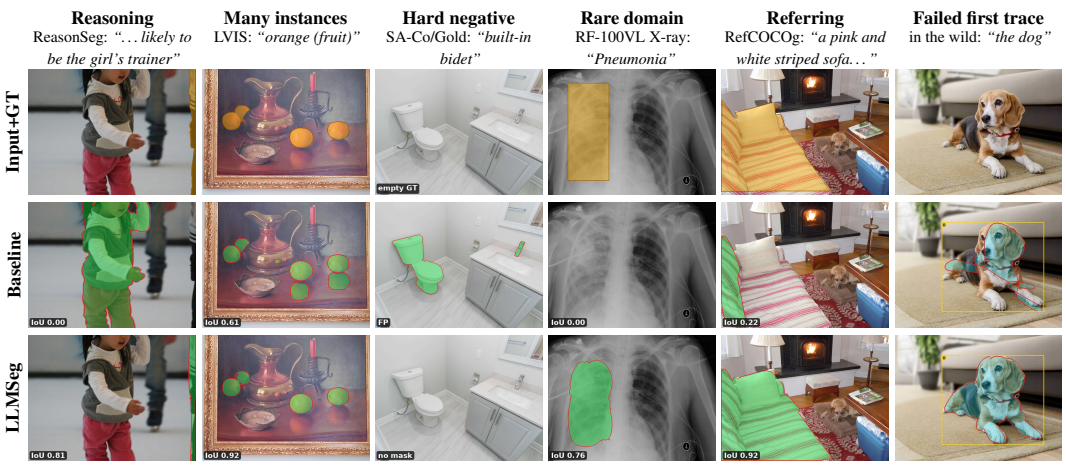


Figure 4: **LLMSeg in six settings.** Top: input with ground truth in amber. Middle: the baseline. This is SAM 3 in the first four columns and LLMSeg (w/o refine) in the last two. Bottom: LLMSeg. Chips give IoU. For the absent concept, they instead indicate whether a system fired.

T passes, or once the union mask changes only negligibly for s consecutive passes (the patience criterion). Section 4 gives the constants; Section 5 examines convergence.

4 Experiments

4.1 Experimental setup

Benchmarks and metrics. We follow the image evaluation suite of SAM 3 (Carion et al., 2026). We use each benchmark's standard metrics. **(i)** *Referring and reasoning segmentation*: We evaluate on ReasonSeg (Lai et al., 2024) (val/test) and all splits of RefCOCO, RefCOCO+, and RefCOCOg (Kazemzadeh et al., 2014; Mao et al., 2016). We report gIoU (mean per-item IoU) and cIoU (cumulative $\sum \mathrm{I} / \sum \mathrm{U}$). **(ii)** *Open-vocabulary instance segmentation*: We evaluate on LVIS (Gupta et al., 2019), COCO (Lin et al., 2014), and SA-Co/Gold (Carion et al., 2026). We report presence-gated cgF1 and pmF1 (with hard negative prompts), positive-set IoU, and COCO-style box/mask AP. **(iii)** *Semantic segmentation*: We report 19-class mIoU on Cityscapes (Cordts et al., 2016), using deterministic mask compositing. We use ADE20K (Zhou et al., 2019) as a class-prompted oracle upper bound. **(iv)** *Zero-shot detection*: We evaluate on ODinW13 (Li et al., 2022a) and RF-100VL (Robicheaux et al., 2025). We report AP_0, the mean per-subset box AP computed from masks.

Protocol: complete splits and sampled subsets. We run each split *in full* whenever end-to-end evaluation is practical. This includes ReasonSeg val (200) and test (779), Cityscapes val (500), and every ODinW13 and RF-100VL subset (6,237 and 1,987 items). In total, we fully evaluate 9,703 of the 20,203 scored items. Four of the eleven benchmarks are run in full. The remaining annotation files contain 10^4 to 10^5 items, making full MLLM inference impractical. We instead draw uniform random samples with seed 0. We do not take the first items because the annotation files are strongly ordered. We sample 2,500 items from SA-Co/Gold, covering all seven domains. We also sample 1,500 items each from LVIS and COCO, 1,000 from ADE20K, and 500 from each RefCOCO/+/g split. On these sampled splits, we run every baseline once on the *full* split. We then recompute its metrics on exactly the item set seen by LLMSeg, using a shared-key intersection. Negative sampling is also identical: 3 negatives per positive for LVIS and COCO, with seed 0.

LLMSeg configuration. The main results use Gemini-3.7-Flash (Comanici et al., 2025) as the backend for all three stages. The backend selects the Stage-2 curve degree $d \in \{1, 3\}$ (Section 3). This degree is shared across all instances in a call. Refinement curves are always cubic. Refinement uses a budget of $T{=}8$ passes, a cap of $N_{\max}{=}200$ instances per image (matching SAM 3's cap), and a focus-box budget of 6 per instance per pass. The stopping criterion is $|M^{(t)} \triangle M^{(t-1)}| < \varepsilon \, |\bigcup_{k=1}^{N} b_k|$, where $\varepsilon{=}0.005$. The loop stops when this criterion holds for $s{=}3$ consecutive passes with $t \geq 1$. Here, s is the stopping patience, $|\cdot|$ counts pixels, and $\triangle$ denotes symmetric difference. The denominator is the union of the instance boxes. By Eq. (2), this is the only region that a pass can alter.

	Semantic segmentation			Zero-shot detection							
	ADE20K-150		Cityscapes	ODinW13				RF-100VL			
Method	gIoU	cIoU	mIoU	AP_0	gIoU	cIoU	pmF1	AP_0	gIoU	cIoU	pmF1
APE-D (LVIS-trained)	–	–	44.2	–	–	–	–	–	–	–	–
Gemini 2.5 Pro	–	–	–	33.7	–	–	–	11.6	–	–	–
gDino-T	–	–	–	49.7	–	–	–	15.7	–	–	–
gDino1.5-Pro	–	–	–	58.7	–	–	–	–	–	–	–
SAM 3	–	–	**65.2**	**61.0**	–	–	–	15.2	–	–	–
SAM 3*	57.9	**67.3**	63.7	51.5	48.3	51.5	39.6	14.2	13.8	21.7	11.4
LLMSeg (w/o refine)	56.6	60.5	48.8	52.0	48.7	48.7	50.0	30.4	42.1	50.8	58.7
LLMSeg (full)	**59.2**	64.7	53.3	57.2	**54.2**	**55.7**	**57.5**	**31.4**	**43.6**	**53.2**	**61.7**

Table 2: **Semantic segmentation and zero-shot detection.** The baselines are APE-D (Shen et al., 2024), Gemini 2.5 Pro (Comanici et al., 2025), and gDino-T (Liu et al., 2024a). LLMSeg uses Gemini-3.7-Flash. Unmarked baselines are published full-split numbers. * Our evaluation uses complete splits, except on ADE20K, where we use the same 1,000 random items as the shaded rows. ADE20K reports oracle-class union-IoU and Cityscapes reports 19-class mIoU. Detection reports mean per-subset box AP_0 and pooled mask metrics. **Bold** is the best and underline the second best per column.

Normalizing by the full image would make the same threshold behave very differently for small and large objects (Section 5).

A full run uses 2 MLLM calls for detection and segmentation. Each refinement pass adds at most 2 calls. Thus, the maximum is $2T+2 = 18$ calls, although early stopping keeps the measured mean well below this limit. The stage-1 topology descriptor (Section 3) is optional. The main results disable it for all three stages (Section 5). We use the same settings on every benchmark and tune nothing per dataset.

Baselines. Our two main baselines are from the SAM 3 paper (Carion et al., 2026). *SAM 3*: We use the public checkpoint through Hugging Face Transformers in fp32. We provide one text query per item and allow up to 200 instances. We score its output with the same metric code on the same items. SAM 3 also accepts image-exemplar and box prompts and supports video. We use none of these features, and LLMSeg is text-only by design. *SAM 3 Agent*: This agent baseline was introduced in the same paper. An MLLM drives SAM 3 as a tool. We reproduce its loop faithfully. The MLLM proposes simple noun phrases for SAM 3, inspects the returned numbered masks, and selects those matching the query. We use Gemma-4-31B-it and Gemini-3.7-Flash rather than the paper's Qwen and Gemini 2.5 Pro backends. The selected union is scored identically. *Published specialists*: We also quote full-split numbers from Carion et al. (2026).

	LVIS			COCO		
Method	cgF1	box AP	mask AP	cgF1	box AP	mask AP
OWLv2	20.1	35.2	–	–	38.2	–
OWLv2 ^{L}	29.3	45.5	43.4	–	46.1	–
gDino-T	14.7	20.5	–	–	45.7	–
LLMDet-L	35.1	42.0	36.3	–	55.6	–
APE-D ^{L}	–	59.6	53.0	–	58.3	–
DINO-X	–	52.4	38.5	–	56.0	–
Gemini 2.5 Flash	13.4	–	–	–	–	–
SAM 3	37.2	53.6	48.5	–	56.4	–
SAM 3*	50.1	**69.1**	**62.2**	52.4	**68.0**	**59.2**
SAM 3 Agent*	51.0	56.2	50.9	49.9	48.9	42.0
LLMSeg (w/o refine)	50.4	57.1	36.9	50.4	59.6	38.0
LLMSeg (full)	**53.9**	64.2	47.2	**56.1**	66.6	50.5

Table 3: **Open-vocabulary instance segmentation.** The baselines are OWLv2 (Minderer et al., 2024), gDino-T (Liu et al., 2024a), LLMDet-L (Fu et al., 2025), APE-D (Shen et al., 2024), DINO-X (Ren et al., 2025), and Gemini 2.5 Flash (Comanici et al., 2025). * Matched subset. Others are published. ^{L} LVIS-trained. cgF1 is presence-gated F1. Gray rows are ours.

4.2 Main Results

Tables 1 to 3 summarize eleven benchmarks. Figure 4 shows one example per setting. Additional examples appear in Figure 12. **Reasoning segmentation.** On ReasonSeg (Table 1), LLMSeg reaches 71.5 cIoU on validation. This is the best result among methods not fine-tuned for the task. Refinement adds 6.9 cIoU over the segment stage alone. The gIoU scores and complete published results for both tasks appear in Section D.1. This is reasoning segmentation without a mask decoder (Figure 3).

Referring expression segmentation. On RefCOCO/+/g (Table 1), LLMSeg substantially outperforms SAM 3, while SAM 3 Agent generally remains stronger. Refinement improves cIoU across all

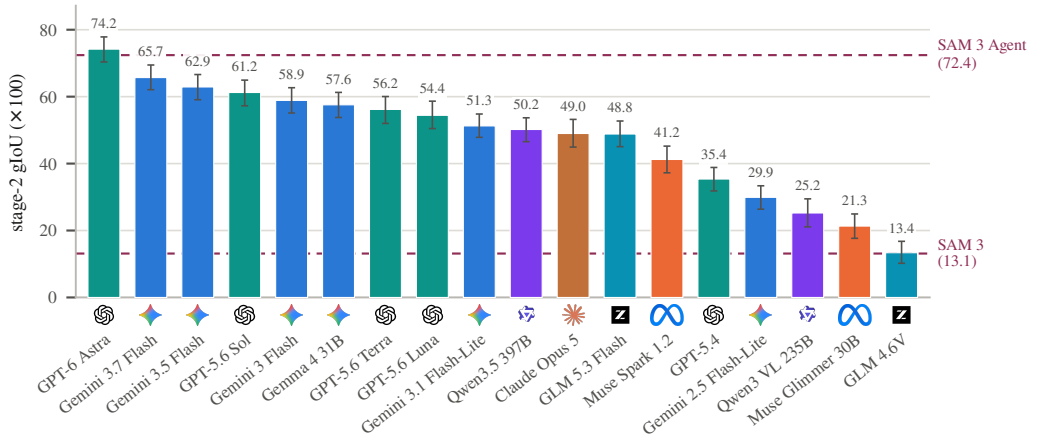


Figure 5: **LLMSeg quality is the backend's quality.** Stage-2 gIoU on all 200 ReasonSeg val items. We show 18 of the 21 tested backends for readability. Whiskers show 95% item-bootstrap intervals. Dashed lines mark SAM 3 and SAM 3 Agent (Gemini-3.7-Flash).

eight splits. On RefCOCO testB, LLMSeg reaches 76.0 cIoU, the best zero-shot result on that split. This result shows that refinement narrows the gap to SAM 3 Agent.

Open-vocabulary instance segmentation. LLMSeg reaches 53.9 and 56.1 cgF1 on LVIS and COCO (Table 3). It outperforms both SAM 3 baselines in cgF1 but trails SAM 3 in mask AP. On SA-Co/Gold, a lower cgF1 (36.4 vs. 45.4) despite similar pmF1 (55.4 vs. 55.2) points to weaker presence decisions.

Semantic segmentation and zero-shot in-the-wild detection. LLMSeg achieves 59.2 gIoU on ADE20K and 53.3 mIoU on Cityscapes (Table 2). On Cityscapes, it trails SAM 3 by 10.4 points. Dense scene labeling therefore remains a limitation. On zero-shot detection, LLMSeg reaches 57.2 AP_0 on ODinW13 and 31.4 on RF-100VL (Table 2). It exceeds SAM 3 on all four metrics in both benchmarks. Its AP_0 on RF-100VL is 2.2× that of SAM 3, showing strong cross-domain transfer.

5 Analysis

5.1 Backend progress transfers without segmentation training

We hold the pipeline fixed (T=12, topology on) and test 21 backends from six developers. Stage-2 gIoU ranges from 13.4 to 74.2 on ReasonSeg val (Figure 5). GPT-6 Astra reaches 74.2 without a segmentation model, compared with 72.4 for SAM 3 Agent driven by Gemini-3.7-Flash. However, the item-bootstrap interval for GPT-6 Astra includes that baseline score. This result therefore does not establish superiority. More directly, swapping generations raises final gIoU from 28.8 to 44.5 within Gemini Flash-Lite. Within GPT, it raises gIoU from 27.1 to 55.9. Neither comparison changes the pipeline or training. The benefit is that LLMSeg can use better backends without retraining, not that every new release is guaranteed to improve performance. Two qualifications matter. First, general benchmark capability predicts segmentation poorly (ρ=0.22, n=17, p=0.41). Second, refinement lowers gIoU on 11 of 21 backends. Its gains therefore depend on the model's ability to inspect and correct its own masks. The full ranking and generation comparisons are in Section E.2. The appendix also reports the stage-2 and final results and the study protocol.

5.2 Refinement improves the boundary and usually stops early

With Gemini-3.7-Flash, refinement improves every benchmark in Tables 1 to 3. Focus boxes let the model correct a local boundary without redrawing the object. In the topology-enabled ablation, removing refinement reduces cIoU by 5.4 on ReasonSeg and 12.7 on RefCOCO (Table 4). This is the clearest component effect. However, refinement is also the largest source of inference cost. The loop has no convergence guarantee, so we measure both mask stability and quality. In the T=12 ReasonSeg study, 79% of items stop before the cap (Figure 19a). The median item uses six passes,

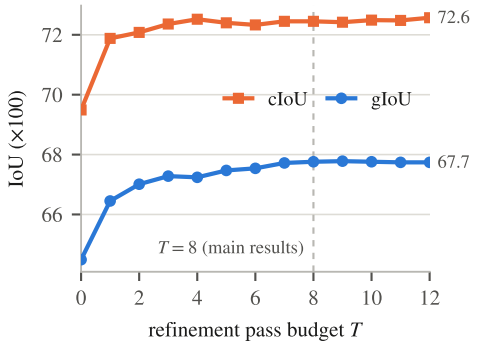


Figure 6: **Refinement budget.** We score the mask after the first T passes at every budget on 200 ReasonSeg val items with Gemini-3.7-Flash. Gains saturate before T=8, the budget of the main results.

Configuration	ReasonSeg val		RefCOCO val	
	cIoU ↑	calls ↓	cIoU ↑	calls ↓
w/o refinement	67.2 ±2.72	2.0	64.6	2.0
w/o detection	72.1 ±0.87	8.8	78.8	10.4
w/ topology	72.6 ±0.23	14.3	77.3	16.8
Polygon loops	73.1 ±3.16	14.7	79.1	16.6
Per-instance refinement	**74.9** ±0.91	21.3	77.7	18.8
Main conditioning	74.7 ±1.62	14.6	**79.8**	17.5

Table 4: **Design alternatives.** Final cIoU and mean MLLM calls per image. ReasonSeg values are mean ± run SD over three runs. RefCOCO values come from one run. Gray marks the main conditioning. Bold marks the best cIoU.

while 18.5% reach the cap. We score the mask after each pass budget (Figure 6). Four passes recover 2.7 of the eventual 3.2 gIoU gain. Raising T from 8 to 12 changes gIoU by under 0.1. The main results therefore use T=8. At T=12, early stopping cuts mean calls from 26 to 14.7.

5.3 What do confinement, batching, and curves contribute?

Table 4 compares the core design alternatives in final-mask quality and inference calls with Gemini-3.7-Flash on 200 items per split. The ablation variants enable topology conditioning. The shaded row uses the main method's conditioning without topology. The refinement budget is T=12, rather than the main results' T=8. The exception is the no-refinement configuration (T=0). ReasonSeg reports three-run means and standard deviations. RefCOCO has one run per configuration. Disabling topology raises observed cIoU on both splits, so we omit it by default (holes: Table 15).

Detection helps the initial trace. Removing detection lowers initial cIoU from 68.3 to 60.3 on ReasonSeg. On RefCOCO, it lowers initial cIoU from 66.0 to 63.5. After refinement, the no-detection configuration closes these gaps with fewer total calls. Detection's initial advantage is larger on the 63 ReasonSeg targets with disconnected ground-truth masks (+11.3 cIoU). After refinement, the advantage reverses (−4.0). These subsets are defined by connected components, not annotated instances. We keep detection for instance identities and box confinement (Eq. (2)), not for final-mask accuracy.

Batching bounds the cost as instance count grows. Relative to the configuration with topology enabled, per-instance cropped refinement increases ReasonSeg calls from 14.3 to 21.3 (49%). This yields a 2.3-point cIoU increase. On RefCOCO, it adds calls for only a 0.4-point increase. Batched tracing uses one call. Batched refinement uses at most two calls per pass, regardless of the instance count. In contrast, crop-based calls grow with the number of instances.

Curves provide a compact representation. Polygon loops are competitive in final cIoU. On the items traced with cubic curves, the topology-enabled configuration emits 28.9 coordinates per image versus 48.5 for polygons, a 1.7× reduction. This result supports compact geometry rather than uniform accuracy superiority. The method can also choose degree one, and polygon prompts allow more vertices. Thus, this is not an equal-budget comparison of primitives.

6 Conclusion

We presented LLMSeg, a training-free pipeline that transforms a multimodal LLM into a text-prompted segmenter using box-confined B-spline geometry and iterative focus-box refinement. Across ten benchmarks spanning five task families, LLMSeg is competitive with SAM 3 and performs better on reasoning queries, rare domains, and presence-gated metrics. Its limitations include detection errors, imprecise pixel-level boundaries, amorphous classes, and inference cost. We will release the code, prompts, and evaluation harness.

AI USE STATEMENT

In this work, we used generative AI tools (LLM coding assistants: Claude Code and Cursor AI) to help write the evaluation-harness code and to edit portions of this manuscript (experimental details and introduction). The segmentation method itself *uses* multimodal LLMs as its vision backend, which is the subject of the paper and is fully described in Section 3. We have not used generative AI tools to generate research ideas, experimental results, or citations. All AI-assisted code was verified and tested for correctness by the authors, and all AI-drafted text was reviewed and revised by the authors. We take responsibility for the final content of this work.

ETHICS STATEMENT

Two aspects merit explicit mention. First, the vision backend is an API-served multimodal LLM, so any image processed by the pipeline is sent to a third-party provider. Deployments on private, medical, or otherwise sensitive imagery should therefore use a locally hosted backend or obtain the consent and data-handling agreements such imagery requires. Second, the method inherits whatever recognition biases its backend carries: because segmentation is driven entirely by the language model's reading of the query and the image, systematic disparities in how a backend recognizes people, objects, or scenes propagate directly into the produced masks. We do not audit backend bias here, and we caution against deploying the pipeline in settings where such disparities carry material consequences without a domain-specific evaluation first. More broadly, open-vocabulary segmentation is dual-use and can support surveillance applications; our released code and prompts target research on reasoning-driven segmentation, and we provide no component specialized for identifying individuals.

REPRODUCIBILITY STATEMENT

LLMSeg is training-free, so there are no training scripts, fine-tuned checkpoints, or generated datasets to release: the method consists of prompts and deterministic geometry code. All code developed for this work, including the three-stage pipeline and the complete evaluation harness used to produce every number reported here, is included in the supplementary material and will be released in a public GitHub repository upon publication. The release contains the per-stage prompts verbatim, dataset loaders and item-key construction code, metric and AP scoring code for all benchmarks, and the structured JSON records generated by every run reported in the paper. These records allow rasterization, composition, and scoring to be re-executed offline without re-querying any model. Every benchmark used in this work is publicly available and was obtained from its official release. The SAM 3 and SAM 3 Agent baselines use the public checkpoint from the Hugging Face Hub through Transformers without modification.

We report detailed configurations to enable full replication of the experiments: pipeline hyperparameters, decoding settings, and the stopping criterion in Section 4 and Section C.2; the prompt templates verbatim in Section F; coordinate conventions and parsing rules in Section B; and the evaluation protocol, including negative-sampling counts and seeds, matched-subset recomputation, and the treatment of empty-ground-truth items, in Section C. The repository also pins the API model identifiers and call parameters for the MLLM backends, the baseline inference settings, and the exact library versions and environment lock file needed to rerun both the proposed method and the baselines. Because providers may update API-served models, we pin model versions and release all per-item records so that the reported results remain verifiable even if a backend changes.

REFERENCES

David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 859–868. IEEE, 2018.

Xiang An, Kaicheng Yang, Xiangzi Dai, Ziyong Feng, and Jiankang Deng. Multi-label cluster discrimination for visual representation learning. In *European Conference on Computer Vision*, pp. 428–444. Springer, 2024.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Patrick Brigger, Jeff Hoeg, and Michael Unser. B-spline snakes: a flexible tool for parametric contour detection. *IEEE Transactions on Image Processing*, 9(9):1484–1496, 2000.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pp. 213–229. Springer, 2020.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris Coll-Vinent, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, et al. Sam 3: Segment anything with concepts. In *International Conference on Learning Representations*, volume 2026, pp. 138846–138923, 2026.

Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a polygon-rnn. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4485–4493. IEEE, 2017.

Yi-Chia Chen, Wei-Hua Li, Cheng Sun, Yu-Chiang Frank Wang, and Chu-Song Chen. SAM4MLLM: Enhance multi-modal large language model for referring expression segmentation. In *European Conference on Computer Vision*, pp. 323–340. Springer, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

Matt Deitke, Christopher Clark, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Zheng Ding, Jieke Wang, and Zhuowen Tu. Open-vocabulary universal image segmentation with MaskCLIP. *arXiv preprint arXiv:2208.08984*, 2022.

Yue Fan, Xiaojian Ma, Rujie Wu, Yuntao Du, Jiaqi Li, Zhi Gao, and Qing Li. Videoagent: A memory-augmented multimodal agent for video understanding. In *European Conference on Computer Vision*, pp. 75–92. Springer, 2024.

James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2 edition, 1996.

Shenghao Fu, Qize Yang, Qijie Mo, Junkai Yan, Xihan Wei, Jingke Meng, Xiaohua Xie, and Wei-Shi Zheng. LLMDet: Learning strong open-vocabulary object detectors under the supervision of large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Yuying Ge, Yixiao Zhao, Ziyun Zhu, et al. Planting a seed of vision in large language model. *arXiv preprint arXiv:2307.08041*, 2023.

Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv preprint arXiv:2104.13921*, 2021.

Agrim Gupta, Piotr Dollar, and Ross Girshick. LVIS: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5356–5364, 2019.

Abdullah Hamdi, Changchun Yang, and Xin Gao. Colon-Bench: An Agentic Workflow for Scalable Dense Lesion Annotation in Full-Procedure Colonoscopy Videos . In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2026*. Springer Nature Switzerland, 2026.

Jiaqi Huang, Zunnan Xu, Ting Liu, Yong Liu, Haonan Han, Kehong Yuan, and Xiu Li. Densely connected parameter-efficient tuning for referring image segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3653–3661, 2025.

Qing Jiang, Feng Li, Zhaoyang Zeng, Tianhe Ren, Shilong Liu, and Lei Zhang. T-Rex2: Towards generic object detection via text-visual prompt synergy. In *European Conference on Computer Vision*, pp. 38–57. Springer, 2024.

Qing Jiang, Junan Huo, Xingyu Chen, Yuda Xiong, Zhaoyang Zeng, Yihao Chen, Tianhe Ren, Junzhi Yu, and Lei Zhang. Detect anything via next point prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 25472–25483, 2026.

Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. MDETR: Modulated detection for end-to-end multi-modal understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1780–1790, 2021.

Seil Kang, Jinyeong Kim, Junhyeok Kim, and Seong Jae Hwang. Your large vision-language model only needs a few attention heads for visual grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9339–9350, 2025.

Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. ReferItGame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 787–798, 2014.

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, 2023. URL https://arxiv.org/abs/2304.02643.

Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. LISA: Reasoning segmentation via large language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9579–9589, 2024.

Mengcheng Lan, Chaofeng Chen, Yue Zhou, Jiaxing Xu, Yiping Ke, Xinjiang Wang, Litong Feng, and Wei Zhang. Text4seg: Reimagining image segmentation as text generation. In *International Conference on Learning Representations*, volume 2025, pp. 1634–1661, 2025.

Justin Lazarow, Weijian Xu, and Zhuowen Tu. Instance segmentation with mask-supervised polygonal boundary transformers. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4372–4381. IEEE, 2022.

Chunyuan Li, Haotian Liu, Liunian Li, Pengchuan Zhang, Jyoti Aneja, Jianwei Yang, Ping Jin, Houdong Hu, Zicheng Liu, Yong Jae Lee, et al. ELEVATER: A benchmark and toolkit for evaluating language-augmented visual models. *Advances in Neural Information Processing Systems*, 35:9287–9301, 2022a.

Jiachen Li, Qing Xie, Renshu Gu, Jinyu Xu, Yongjian Liu, and Xiaohan Yu. Lgd: Leveraging generative descriptions for zero-shot referring image segmentation. *arXiv preprint arXiv:2504.14467*, 2025.

Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10965–10975, 2022b.

Feng Liang, Bichen Wu, Xiaoliang Dai, Kunpeng Li, Yinan Zhao, Hang Zhang, Peizhao Zhang, Peter Vajda, and Diana Marculescu. Open-vocabulary semantic segmentation with mask-adapted CLIP. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7061–7070, 2023.

Justin Liang, Namdar Homayounfar, Wei-Chiu Ma, Yuwen Xiong, Rui Hu, and Raquel Urtasun. Polytransform: Deep polygon transformer for instance segmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9128–9137. IEEE, 2020.

Wenqi Liang, Gan Sun, Yao He, Jiahua Dong, Suyan Dai, Ivan Laptev, Salman Khan, and Yang Cong. Pixelvla: Advancing pixel-level understanding in vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 110464–110481, 2026.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, Zürich, 2014. URL /se3/wp-content/uploads/2014/09/coco_eccv.pdf,http://mscoco.org. Oral.

Weifeng Lin, Xinyu Wei, Ruichuan An, Tianhe Ren, Tingwei Chen, Renrui Zhang, Ziyu Guo, Wentao Zhang, Lei Zhang, and Hongsheng Li. Perceive anything: Recognize, explain, caption, and segment anything in images and videos. *Advances in Neural Information Processing Systems*, 38: 96231–96256, 2026.

Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. Fast interactive object annotation with curve-gcn. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5252–5261. IEEE, 2019.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, pp. 38–55. Springer, 2024a.

Ting Liu and Siyuan Li. Hybrid global-local representation with augmented spatial guidance for zero-shot referring image segmentation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 29634–29643, 2025.

Yong Liu, Cairong Zhang, Yitong Wang, Jiahao Wang, Yujiu Yang, and Yansong Tang. Universal segmentation at arbitrary granularity with language instruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3459–3469, 2024b.

Yuliang Liu, Hao Chen, Chunhua Shen, Tong He, Lianwen Jin, and Liangwei Wang. Abcnet: Real-time scene text spotting with adaptive bezier-curve network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9806–9815. IEEE, 2020.

Yuqi Liu, Bohao Peng, Zhisheng Zhong, Zihao Yue, Fanbin Lu, Bei Yu, and Jiaya Jia. Seg-Zero: Reasoning-chain guided segmentation via cognitive reinforcement. *arXiv preprint arXiv:2503.06520*, 2025.

Yi Lu, Jiawang Cao, Yongliang Wu, Bozheng Li, Licheng Tang, Yangguang Ji, Chong Wu, Jay Wu, and Wenbo Zhu. RSVP: Reasoning segmentation via visual prompting and multi-modal chain-of-thought. *arXiv preprint arXiv:2506.04277*, 2025.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11–20, 2016.

Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, et al. Simple open-vocabulary object detection. In *European Conference on Computer Vision*, pp. 728–755. Springer, 2022.

Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Scaling open-vocabulary object detection, 2024. URL https://arxiv.org/abs/2306.09683.

Minheng Ni, Yabo Zhang, Kailai Feng, Xiaoming Li, Yiwen Guo, and Wangmeng Zuo. Ref-diff: Zero-shot referring image segmentation with generative models. *arXiv preprint arXiv:2308.16777*, 2023.

OpenAI. Gpt-4 technical report, 2023.

Sida Peng, Wen Jiang, Huaijin Pi, Xiuli Li, Hujun Bao, and Xiaowei Zhou. Deep snake for real-time instance segmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8530–8539. IEEE, 2020.

Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, 2 edition, 1997.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, pp. 8748–8763. PMLR, 2021.

Hanoona Rasheed, Muhammad Maaz, Sahal Shaji, Abdelrahman Shaker, Salman Khan, Hisham Cholakkal, Rao M Anwer, Eric Xing, Ming-Hsuan Yang, and Fahad S Khan. GLaMM: Pixel grounding large multimodal model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13009–13018, 2024.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024. URL https://arxiv.org/abs/2408.00714.

Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded SAM: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024.

Tianhe Ren, Yihao Chen, Qing Jiang, Zhaoyang Zeng, Yuda Xiong, Wenlong Liu, Zhengyu Ma, Junyi Shen, Yuan Gao, Xiaoke Jiang, Xingyu Chen, Zhuheng Song, Yuhong Zhang, Hongjie Huang, Han Gao, Shilong Liu, Hao Zhang, Feng Li, Kent Yu, and Lei Zhang. DINO-X: A unified vision model for open-world object detection and understanding, 2025. URL https://arxiv.org/abs/2411.14347.

Peter Robicheaux, Matvei Popov, Anish Madan, Isaac Robinson, Joseph Nelson, Deva Ramanan, and Neehar Peri. Roboflow100-VL: A multi-domain object detection benchmark for vision-language models. *arXiv preprint arXiv:2505.20612*, 2025.

Yunhang Shen, Chaoyou Fu, Peixian Chen, Mengdan Zhang, Ke Li, Xing Sun, Yunsheng Wu, Shaohui Lin, and Rongrong Ji. Aligning and prompting everything all at once for universal visual perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13193–13203, 2024.

Shuyang Sun, Runjia Li, Philip Torr, Xiuye Gu, and Siyang Li. Clip as rnn: Segment countless visual concepts without training endeavor. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13171–13182, 2024.

Yucheng Suo, Linchao Zhu, and Yi Yang. Text augmented spatial-aware zero-shot referring image segmentation. *arXiv preprint arXiv:2310.18049*, 2023.

Paul Voigtlaender, Valentin Gabeur, and Rohan Doshi. Conversational image segmentation with Gemini 2.5. https://developers.googleblog.com/en/conversational-image-segmentation-gemini-2-5/, 2025.

Hao Wang, Limeng Qiao, Zequn Jie, Zhijian Huang, Chengjian Feng, Qingfang Zheng, Lin Ma, Xiangyuan Lan, and Xiaodan Liang. X-SAM: From segment anything to any segmentation. *arXiv preprint arXiv:2508.04655*, 2025a.

Shijie Wang, Dahun Kim, Ali Taalimi, Chen Sun, and Weicheng Kuo. Learning visual grounding from generative vision and language model. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 8057–8067. IEEE, 2025b.

Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, et al. Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *Advances in Neural Information Processing Systems*, 36:61501–61513, 2023.

Yuji Wang, Jingchen Ni, Yong Liu, Chun Yuan, and Yansong Tang. Iterprime: Zero-shot referring image segmentation with iterative grad-cam refinement and primary word emphasis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 8159–8168, 2025c.

Cong Wei, Yujie Zhong, Haoxian Tan, Yong Liu, Zheng Zhao, Jie Hu, and Yujiu Yang. HyperSeg: Towards universal visual segmentation with large language model. *arXiv preprint arXiv:2411.17606*, 2024.

Jiannan Wu, Muyan Zhong, Sen Xing, Zeqiang Lai, Zhaoyang Liu, Zhe Chen, Wenhai Wang, Xizhou Zhu, Lewei Lu, Tong Lu, et al. Visionllm v2: An end-to-end generalist multimodal large language model for hundreds of vision-language tasks. *Advances in Neural Information Processing Systems*, 37:69925–69975, 2024a.

Junfeng Wu, Yi Jiang, Qihao Liu, Zehuan Yuan, Xiang Bai, and Song Bai. General object foundation model for images and videos at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3783–3795, 2024b.

Zhuofan Xia, Dongchen Han, Yizeng Han, Xuran Pan, Shiji Song, and Gao Huang. GSVA: Generalized segmentation via multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3858–3869, 2024.

Enze Xie, Peize Sun, Xiaoge Song, Wenhai Wang, Xuebo Liu, Ding Liang, Chunhua Shen, and Ping Luo. Polarmask: Single shot instance segmentation with polar representation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12190–12199. IEEE, 2020.

Yin Xie, Kaicheng Yang, Xiang An, Kun Wu, Yongle Zhao, Weimo Deng, Zimin Ran, Yumeng Wang, Ziyong Feng, Roy Miles, et al. Region-based cluster discrimination for visual representation learning. *arXiv preprint arXiv:2507.20025*, 2025.

Jiarui Xu, Xingyi Zhou, Shen Yan, Xiuye Gu, Anurag Arnab, Chen Sun, Xiaolong Wang, and Cordelia Schmid. Pixel aligned language models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13030–13039. IEEE, 2024.

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.

Seonghoon Yu, Paul Hongsuck Seo, and Jeany Son. Zero-shot referring image segmentation with global-local context features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19456–19465, 2023.

Seonghoon Yu, Paul Hongsuck Seo, and Jeany Son. Pseudo-ris: Distinctive pseudo-supervision generation for referring image segmentation. In *European Conference on Computer Vision*, pp. 18–36. Springer, 2024.

Tao Zhang, Shiqing Wei, and Shunping Ji. E2ec: An end-to-end contour-based method for high-quality high-speed instance segmentation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4433–4442. IEEE, 2022.

Tao Zhang, Xiangtai Li, Hao Fei, Haobo Yuan, Shengqiong Wu, Shunping Ji, Chen Change Loy, and Shuicheng Yan. OMG-LLaVA: Bridging image-level, object-level, pixel-level reasoning and understanding. *Advances in Neural Information Processing Systems*, 37:71737–71767, 2024a.

Yuxuan Zhang, Tianheng Cheng, Lianghui Zhu, Rui Hu, Lei Liu, Heng Liu, Longjin Ran, Xiaoxin Chen, Wenyu Liu, and Xinggang Wang. EVF-SAM: Early vision-language fusion for text-prompted segment anything model. *arXiv preprint arXiv:2406.20076*, 2024b.

Zheng Zhang, Yeyao Ma, Enming Zhang, and Xiang Bai. PSALM: Pixelwise segmentation with large multi-modal model. In *European Conference on Computer Vision*, pp. 74–91. Springer, 2024c.

Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ADE20K dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

Chong Zhou, Chenchen Zhu, Yunyang Xiong, Saksham Suri, Fanyi Xiao, Lemeng Wu, Raghuraman Krishnamoorthi, Bo Dai, Chen Change Loy, Vikas Chandra, et al. Edgetam: On-device track anything model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 13832–13842, 2025.

Chenming Zhu, Xuanye Zhang, Yanran Li, Liangdong Qiu, Kai Han, and Xiaoguang Han. Sharpcontour: A contour-based boundary refinement approach for efficient and accurate instance segmentation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4382–4391. IEEE, 2022.

Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. Segment everything everywhere all at once. *Advances in Neural Information Processing Systems*, 36:19769–19782, 2023.

APPENDIX

The appendix follows the order of the paper. Section A extends the related work. Section B gives the method details that the main text states only in words, and Section C the experimental setup: benchmarks, configuration, scoring and the protocols of the backend and cost studies. Section D holds the complete result tables and the qualitative grids, Section E the extended analysis (refinement, backends, ablations, topology, localization, cost and limitations), and Section F the prompt templates verbatim.

A EXTENDED RELATED WORK

Promptable and open-vocabulary segmentation. SAM (Kirillov et al., 2023) introduced promptable segmentation with points, boxes, and masks; SAM 2 (Ravi et al., 2024) extended it to video, and SAM 3 (Carion et al., 2026) to open-vocabulary *concept* prompts, training a DETR-based detector-tracker (Carion et al., 2020; Kamath et al., 2021) on a purpose-built data engine. Open-vocabulary detection and segmentation methods (Gu et al., 2021; Minderer et al., 2022; Ding et al., 2022; Liang et al., 2023) leverage vision-language encoders such as CLIP (Radford et al., 2021); GLIP (Li et al., 2022b) and GroundingDINO (Liu et al., 2024a) cast detection as phrase grounding, with OWLv2 (Minderer et al., 2024), LLMDet (Fu et al., 2025), APE (Shen et al., 2024), DINO-X (Ren et al., 2025), GLEE (Wu et al., 2024b), and T-Rex2 (Jiang et al., 2024) pushing scale and prompt modalities. All of these are *trained specialists*. LLMSeg instead treats segmentation as an emergent capability of a general MLLM, requiring no segmentation training at all, and uses only text prompts (no exemplars, boxes, or video, unlike SAM 3).

MLLMs for grounding and segmentation. A large family attaches a trained mask decoder to an (M)LLM: LISA (Lai et al., 2024) introduced reasoning segmentation by decoding a <SEG> token with SAM; GLaMM (Rasheed et al., 2024), GSVA (Xia et al., 2024), OMG-LLaVA (Zhang et al., 2024a), PSALM (Zhang et al., 2024c), EVF-SAM (Zhang et al., 2024b), HyperSeg (Wei et al., 2024), and X-SAM (Wang et al., 2025a) refine this recipe; SAM4MLLM (Chen et al., 2024) and Seg-Zero (Liu et al., 2025) train the MLLM to emit prompts for SAM. Training-free alternatives still delegate the pixels to a segmentation model: RSVP (Lu et al., 2025) and attention-based grounding (Kang et al., 2025) use the MLLM for localization only, and SAM 3 Agent (Carion et al., 2026) lets an MLLM drive SAM 3 as a tool, similar in spirit to set-of-mark prompting (Yang et al., 2023). Native MLLM segmentation has appeared only as coarse per-token masks in Gemini 2.5 (Comanici et al., 2025; Voigtlaender et al., 2025), which SAM 3 reports as far behind specialists. LLMSeg is, to our knowledge, the first pipeline in which a *frozen, general-purpose* MLLM produces the final mask geometry itself, as parametric splines rather than pixels and with no trained component anywhere downstream, at accuracy competitive with trained systems. The prior work closest to that output space is discussed next.

MLLMs that emit coordinates as text. Writing geometry down as tokens rather than decoding it is not itself new; what is new here is that the tokens are the final answer. In prior work the emitted text is an intermediate consumed by a trained head, and the pipeline around it is trained. VisionLLM (Wang et al., 2023) is the closest precedent: it represents an instance boundary as a short polygon, typically 16 points, emitted by the language model, but keeps a trained decoding head per task, which both ties generalization to the task set seen in training and caps mask resolution at the vertex budget. VisionLLM v2 (Wu et al., 2024a) scales that recipe to hundreds of vision-language tasks by routing a special per-task token to a decoder finetuned for that task, finetuning the MLLM alongside each one. PixelLLM (Xu et al., 2024) adds an MLP head regressing locations and an encoder embedding an input box to give an MLLM localization and grounding, trains the pipeline end to end, and still obtains masks by prompting SAM with a box. Text4Seg (Lan et al., 2025) pushes furthest into text as the output space, defining segmentation tokens over a row-ordered grid that is encoded and decoded so that masks are produced by generating text, yet the MLLM must be finetuned to generate them. Beyond segmentation, Jiang et al. (2026) cast detection as next-point prediction over the same kind of structured coordinate text, training the full pipeline with supervised finetuning followed by reinforcement learning, and report detection and grounding but no masks; PixelVLA (Liang et al., 2026) carries pixel-level grounding into vision-language-action models with trained pixel modules. Perceive Anything (Lin et al., 2026) instead couples SAM 2 with a modern LLM in a single jointly trained pipeline for fine-grained region understanding, and its gains are in recognition, explanation and captioning: the masks come from the SAM 2 branch, not from the language model. Frontier

general-purpose models are acquiring coarse localization on their own, with Molmo (Deitke et al., 2025) able to point at queried objects and the Gemini series grounding boxes well (Comanici et al., 2025; Voigtlaender et al., 2025), but neither comes close to a specialized segmentation network or SAM 3 at the pixel level. LLMSeg departs from all of the above in one consistent way: the backend is frozen and general-purpose, nothing downstream of it is trained, and the control points it writes out are the prediction itself rather than a prompt, a latent, or a seed for a decoder.

Agentic MLLM pipelines. Beyond single-call inference, MLLMs increasingly act as agents that decompose visual tasks, call tools, and verify intermediate results (Fan et al., 2024; Ge et al., 2023; Yang et al., 2023), including for large-scale annotation (Carion et al., 2026). Colon-Bench (Hamdi et al., 2026) shows such a workflow producing dense lesion masks in a niche clinical domain where SAM 3 fails, but the masks are still drawn by EdgeTAM (Zhou et al., 2025), a compact SAM 2 tracker that the MLLM calls as a tool, so the pixel-level competence remains the tool's. LLMSeg follows this philosophy with a fixed, budgeted loop: every stage is a structured JSON exchange, and deterministic geometry code (rasterization, clamping, composition) provides the reliability that free-form agent loops lack. The iterative focus-box refinement can be seen as the model *self-verifying* its own rendered prediction, an ability that improves with backend reasoning strength (Section 5).

Contour and curve parameterizations of masks. Describing a mask by its boundary rather than by a pixel grid has a thirty-year history. Active contour models (Kass et al., 1988) evolve a spline under internal smoothness and external image forces, and Brigger et al. (2000) showed that the optimal solution of the curvature-constrained snake energy is a cubic spline, the classical argument for the representation we adopt. Deep learning recast the same idea as vertex prediction: Polygon-RNN (Castrejon et al., 2017) and Polygon-RNN++ (Acuna et al., 2018) emit polygon vertices autoregressively for interactive annotation, Curve-GCN (Ling et al., 2019) predicts all vertices at once with a graph network and supports spline as well as polygon outputs, Deep Snake (Peng et al., 2020) deforms an initial contour with circular convolutions, PolarMask (Xie et al., 2020) regresses boundary rays in polar coordinates, PolyTransform (Liang et al., 2020) and SharpContour (Zhu et al., 2022) deform the polygon of an existing mask to sharpen it, E2EC (Zhang et al., 2022) learns the contour initialization end to end, and BoundaryFormer (Lazarow et al., 2022) regresses polygons with a transformer supervised only through a differentiable rasterizer. Parametric curves are standard in text spotting as well, where ABCNet (Liu et al., 2020) localizes curved text with Bézier control points. LLMSeg shares its output space with this literature and little else: each method above is a network trained to regress vertices from image features over a fixed label set, whereas LLMSeg obtains control points as text from a frozen general-purpose MLLM, with no added contour or mask supervision and no closed vocabulary. The representation is what makes that possible, since a handful of control points is short enough to be a language model's output, and focus-box confinement lets the refinement loop add or subtract local regions without retracing the whole outline.

B METHOD DETAILS

B.1 STRUCTURED REPLIES AND COORDINATE CONVENTIONS

Prompts and output schemas. Each stage is a single prompt with a strict structured output schema. Detect returns a list of detections, each with a box, a label, a confidence, and the morphology description and, when topology conditioning is enabled, the component and hole counts. Segment returns, for every instance id, its list of spline loops, each with a role (outer boundary or hole), a degree, and its control points. The degree is the model's own choice of 1 or 3 and is shared by every loop of every instance in the call; the refinement schema has no degree field, since focus loops are always cubic. The two refinement calls return, respectively, typed focus boxes tagged with the instance they belong to, and the confined spline loops for every surviving focus box. Each instance line in the segment prompt restates its box as an explicit numeric inequality: the points are clamped in any case, but a clamped-in point is a worse outline than a correctly placed one. §F gives the templates themselves.

Coordinate conventions across backends. Everything passed between stages is in absolute pixels with origin top-left, boxes as $[x_1, y_1, x_2, y_2]$. Gemini-family models natively emit boxes as $[y_{\min}, x_{\min}, y_{\max}, x_{\max}]$ on a 0 to 1000 grid. Asking them for x-first order instead does not remove this: they then transpose their own output on some multi-instance images, and no downstream check can catch it, because a swapped box is still a well-formed box. We therefore request each family's *native* order and convert afterwards, transposing before rescaling. Control points are $[x, y]$ for every family; a loop that comes back transposed is detected, because the swapped reading sits almost entirely inside its box while the raw reading does not, and repaired against that box. Unlike a box, points confined to a box can be checked this way, since the box is ground truth for them.

Recovering truncated replies. Batched replies covering several instances are long enough to hit output limits. Rather than discard such a reply, we keep every instance that was completed before the cut and treat the rest as missing, which the refinement stage can still grow from their boxes. The prompts fix the emission order by ascending instance id, so a truncation costs the tail rather than an arbitrary subset. Detect replies are recovered the same way. An instance that survives with no loops is scored as an empty mask, never dropped from the item.

B.2 RASTERIZATION AND COMPOSITION

Rasterization. A cubic loop is rasterized with $m = \min\{2000, \max\{64, \mathrm{round}(2(w_b + h_b))\}\}$ samples for a rasterization box b of width w_b and height h_b. Before clipping, this is roughly one sample per pixel of box perimeter. Samples include both endpoints as in Eq. (2); degree-1 loops use the closed control polygon directly. The box is b_k for an initial instance loop and β_j for a refinement loop.

Composing a refinement pass. Compose in Eq. (3) is deterministic. Naively re-applying the signed regions each pass makes the mask oscillate: a boundary pixel added by one pass is removed by the next and added again. We therefore carry a signed *commitment* field alongside each instance mask, marking every pixel as committed to foreground, committed to background, or uncommitted, with a magnitude that records how often passes have agreed on it. A refinement writes freely into uncommitted pixels; one that agrees with an existing commitment reinforces it; one that opposes it must first cancel one unit of commitment, and only flips the pixel once the accumulated resistance is spent. Repeated agreement therefore makes a pixel progressively harder to overturn. The unit an opposing edit cancels is the only constant here, and we leave it at the value that lets a single edit spend one unit of agreement; setting it to zero would freeze committed pixels permanently. Eq. (4) below gives the per-pixel rule.

Two deterministic steps bracket the composition. An *effect gate* drops, before the re-tracing call, any focus box whose largest achievable change to the current mask is negligible, which is what lets many passes cost one call rather than two. Afterwards, a light morphological cleanup sized relative to the instance box removes specks and spurious holes; where the stage-1 topology descriptor is in use it gates that cleanup, so an object predicted to have holes keeps them.

The soft lock, per pixel. This expands Compose in Eq. (3). Let $\Lambda_k^{(t)}$ be instance k's signed commitment field after pass t, initialized to zero before refinement. Fix a pass t, a focus box j that

corrects instance $k = k_j$, and a pixel x in its correction region $\mathcal{R}_{\beta_j,3}(P_j)$. Write a for the pixel's current binary mask value and ℓ for its current signed commitment, initially $a = \mathbf{1}[x \in M_k^{(t-1)}]$ and $\ell = \Lambda_k^{(t-1)}(x)$. These values carry forward after each correction within the pass. Let $\chi = \operatorname{sgn} \ell$ and $v = |\ell| - \alpha$, the resistance left after an opposing edit erodes it by α ($\alpha = 1$ throughout). With $\sigma = \sigma_j$, one correction updates the pixel to

$$(a', \ell') = \begin{cases} (\mathbf{1}[\sigma = +1],\ \sigma) & \ell = 0 \quad \text{(free: commit)}, \\ (\mathbf{1}[\sigma = +1],\ \ell + \sigma) & \chi = \sigma \quad \text{(agree: reinforce)}, \\ (a,\ \chi\, v) & \chi = -\sigma,\ v > 0 \quad \text{(oppose: erode)}, \\ (\mathbf{1}[\sigma = +1],\ \sigma) & \chi = -\sigma,\ v \le 0 \quad \text{(oppose: flip)}. \end{cases} \tag{4}$$

Here $\mathbf{1}[A]$ is 1 when statement A is true and 0 otherwise. Primes denote a single correction, not a complete refinement pass. After all corrections and the cleanup below, the resulting mask and field are $M_k^{(t)}$ and $\Lambda_k^{(t)}$; the union is $M^{(t)}$. Negative regions are applied before positive ones within a pass. After composition, positive islands disconnected from the main mask are bridged when close, and a deterministic cleanup applies a minor closing/opening sized as a percentage of the instance-box minimum dimension and removes specks and spurious holes scaled by object area, gated on the stage-1 component and hole counts so that an object predicted to have holes keeps them.

B.3 The residual as a stopping signal

The stopping criterion of Section 4 acts on the residual $\delta^{(t)} = |M^{(t)} \triangle M^{(t-1)}| \,/\, |\bigcup_{k=1}^{N} b_k|$, the fraction of the instance-box union that a pass repaints, and not on any measure of quality: the pipeline sees no ground truth at run time, so how much the mask still moves is the only observable it has. Figure 7 shows that quantity pass by pass on a 16-image development set (free text queries on images that belong to no evaluation benchmark), which is where ε and s were set before any benchmark was run. The 16 images and their queries are attached to the supplementary material with the code.

Three things read off it. First, the residual does decay. It is not monotone, since a pass is free to propose a large new edit at any point, but every trace ends below where it started, most of them by an order of magnitude or more, and several reach the axis floor, that is, passes that changed no pixel at all. Second, the rate is a property of the image, not a constant. The first pass repaints between 0.2% (*person*) and 27% (*toy*) of the box union, a $175\times$ spread (unsmoothed values; the figure plots the 3-pass average), and the starting value does not predict the decay: *toy* starts highest and stops itself at pass 10, while *taxis*, six crowded instances of the same class, starts at 6% and is still moving 2% of its box union at pass 12. What makes an item slow is boundary length, occlusion and instance count, none of which is known before the run, which is why the budget has to be spent per image rather than fixed globally. Third, ε=0.005 with s=3 sits where those two regimes separate: 11 of the 16 items stop on their own, the earliest at pass 3 (the patience floor, which no item can beat), the median at 7, and 5 run to the cap.

A small residual is a statement about stability, not about correctness, and a loop could in principle sit still on a wrong mask. What licenses using it as a surrogate is the aggregate behaviour at benchmark scale: on ReasonSeg val, quality saturates on the same scale over which these residuals collapse (Figure 19(b)), and the items that never stop are also the items that end furthest behind in gIoU, so mask stability and accuracy stop moving together rather than one hiding the other.

B.4 One item, step by step

Figure 8 follows a single ReasonSeg item through every call the pipeline makes, on the query *"the spare wheel"*. Stage 2 is already competent here, scoring 84.1 IoU, and the loop takes it to 91.9: this is the ordinary case, not a rescue. What makes the item instructive is *what* is wrong with the stage-2 trace. The wheel is mounted behind a black fender that cuts across its lower-left, and the outline runs straight past the occlusion, so the entire error is one wedge of car body inside an otherwise good boundary. Nothing in the figure is drawn for illustration. Every box, control point and loop is the geometry the model actually returned for that item, and every mask is rebuilt from those records by the pipeline's own composition code, which reproduces the stage-2 and final masks exactly.

Three things in the figure are worth reading closely. First, this is a cubic instance and it looks like one: stage 2 chose d=3 and placed 24 control points around a circle, none of them outside the box it

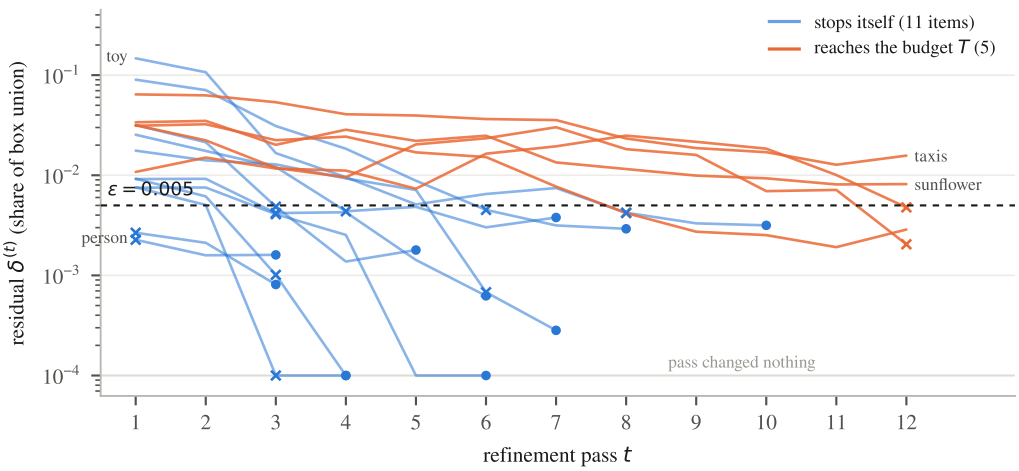


Figure 7: **Per-item refinement residuals on the development set.** One curve per development image, Gemini-3.7-Flash at budget T=12: the 3-pass moving average of the residual $\delta^{(t)}$ on a log axis, dashed at the threshold ε. Blue items stop themselves, orange ones exhaust the budget, and $\times$ marks the first smoothed pass below ε.

was given, and Eq. (1) turns them into a smooth closed curve rather than a 24-gon. Second, the loop is mostly subtractive. Pass 1 removes 15,118 pixels of fender, 9.4% of the object's area, and buys 7.3 IoU; the two later passes shave slivers off the same edge. Third, pass 3 is the interesting one. It proposes *adding* a wedge back along that lower-left edge, and the mask does not move: composing that proposal against the commitment field left by the first two passes changes a single pixel, where the same proposal against an empty field would have added 7,791 (a further 4.8% of the object). This is the signed resistance of Eq. (3) doing exactly what it is for, refusing an edit that contradicts what earlier passes committed to instead of letting the boundary oscillate. The loop then stopped itself after 4 of the 12 allowed passes.

B.5 Detect-box refinement: an option we excluded

Since segmentation quality is bounded by the detect box (Section 3), an obvious extension is to let the model *move the box* before tracing. We implemented and evaluated this as a single pre-pass placed between stages 1 and 2: one call returns, per instance, signed per-side deltas (x_1, x_2, y_1, y_2; positive expands the side outward, negative pulls it inward), each clamped to a fraction of that box's width or height so a single call cannot relocate an object, and the adjusted boxes are clipped to the image.

Figure 9 shows the case it is designed for: a backend whose detector under-covers the object, where widening the box before tracing lets the outline reach parts that box confinement would otherwise have clipped away.

The pre-pass nonetheless does not earn its place in LLMSeg. Table 5 compares the pipeline with and without it on ReasonSeg val. With Gemini-3.7-Flash the model simply declines to move the boxes: only 6.5% of instances, on 9.2% of images, come back with a non-zero delta on any side. The pre-pass therefore spends a call per image to change nothing on nine images in ten, which is the whole argument against it. The measured gIoU does rise, but that rise is present in equal measure on the images the pre-pass left untouched, so it is backend variance rather than an effect (see the caption); we make no claim either way about its size. With Gemini-3.1-Flash-Lite the boxes do move, on 18.9% of instances and 26.8% of images, and that run is 2.0 gIoU *worse* with the pre-pass than without: a call spent moving boxes is not a call that buys accuracy. The focus-box refinement of Section 3 already recovers most of the coverage a slightly tight box costs, since a positive focus box near a boundary has a similar effect to widening that side. We therefore exclude the pre-pass, and every number in Section 4.2 is produced without it.

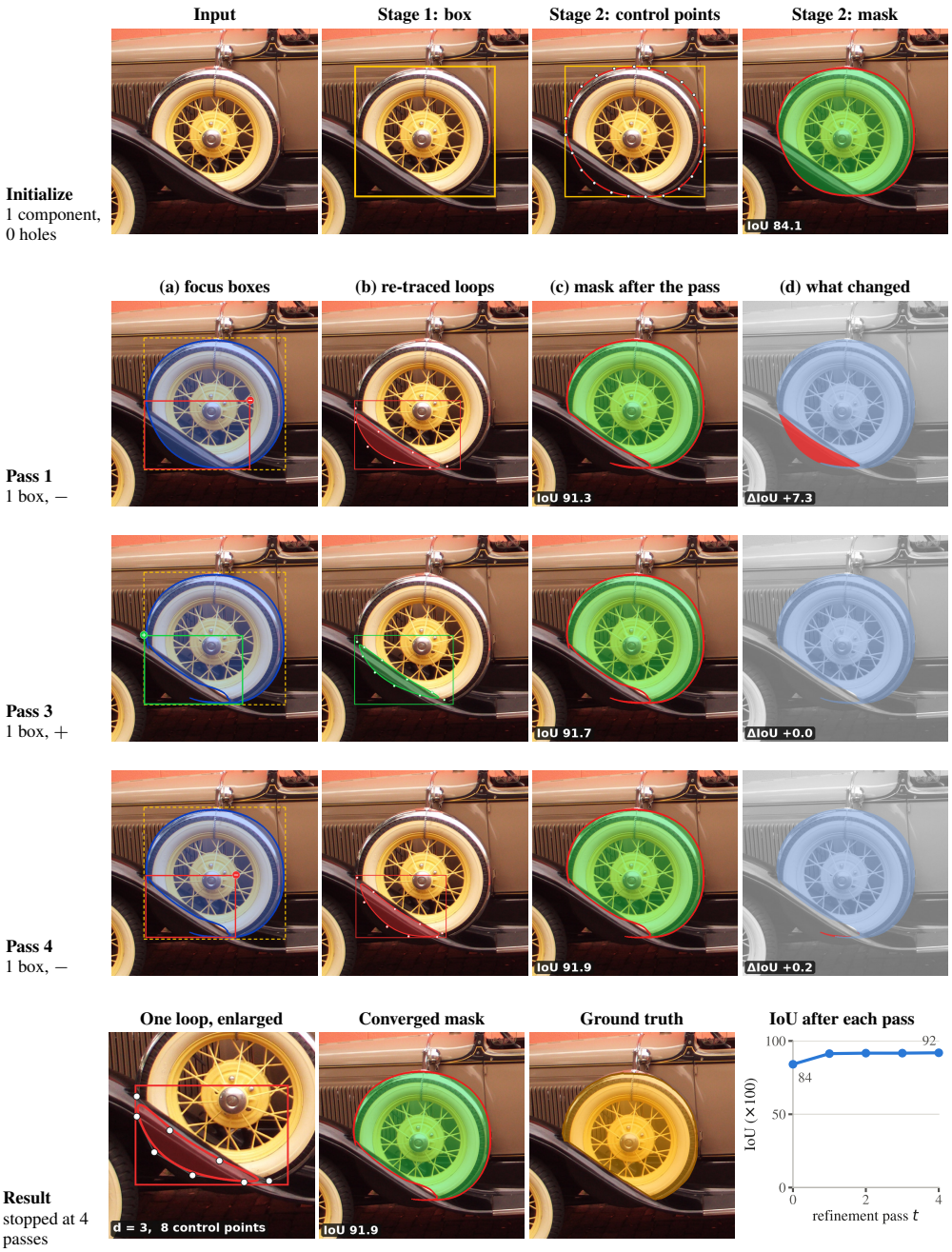
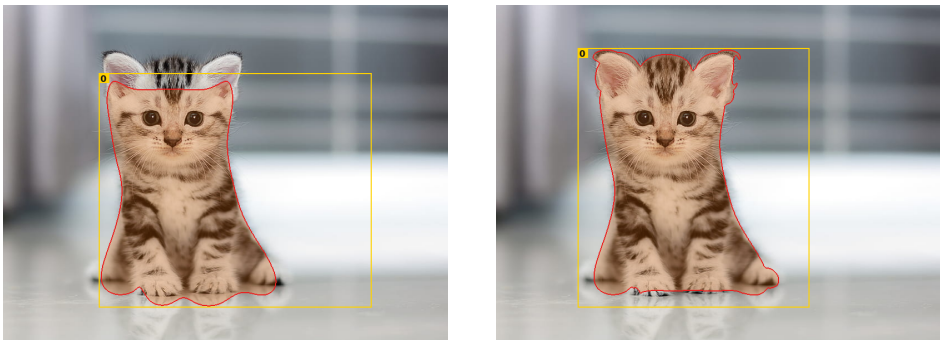


Figure 8: **Every step of LLMSeg on one ReasonSeg test item**, *"the spare wheel"*, with the Gemini-3.7-Flash backend. **Top:** stage 1's box, then stage 2's control points and the curve of Eq. (1) through them. **Middle:** three of the four passes, each proposing signed focus boxes over the current mask and composing the re-traced loops, with the net effect in green (added) and red (removed). **Bottom:** one loop enlarged, the converged mask, the ground truth, and the IoU after each pass.



without the pre-pass with the pre-pass

Figure 9: **When detect-box refinement helps.** With a backend whose stage-1 box under-covers the object, expanding the box before tracing lets the outline reach the clipped region. Backend Gemini-3.1-Flash-Lite, query *"the cat"*. Across a full benchmark the situation is rare enough that the pre-pass is not part of LLMSeg (Table 5).

Backend	**w/o pre-pass**	**w/ pre-pass**	Δ	**boxes moved (%)**
Gemini-3.7-Flash (used in LLMSeg)	69.2	69.7	$+0.5^{*}$	6.5
Gemini-3.1-Flash-Lite	47.1	45.1	-2.0	18.9

Table 5: **Effect of the detect-box adjustment pre-pass** on ReasonSeg val (all 200 items, T=12), per backend. Δ is the change in gIoU from adding the pre-pass, whose extra call is paid on every image whether or not a box moves. Gemini-3.7-Flash averages three runs, Gemini-3.1-Flash-Lite one. *Not attributable to the pre-pass: smaller than the run-to-run deviation, and it survives on the images where no box moved at all.

C Experimental Details

C.1 Benchmarks

We evaluate on five families of benchmarks, listed with their statistics in Table 6. An item is always one (image, query) pair, and the query is the only prompt any system receives. *Reasoning segmentation.* ReasonSeg (Lai et al., 2024) pairs images from OpenImages and ScanNetv2 with implicit queries that name the target by its role or function rather than its category, so the query has to be resolved before anything can be segmented; its 1,218 pairs are split into 239 train, 200 val and 779 test, and the ground truth is one binary mask per query. *Referring expression segmentation.* RefCOCO and RefCOCO+ (Kazemzadeh et al., 2014) were collected in a two-player game on COCO images, so their expressions are short and disambiguate one instance from same-class neighbours; RefCOCO+ forbids absolute location words, which forces appearance-based descriptions, and the testA/testB split separates images crowded with people from images crowded with other categories. RefCOCOg (Mao et al., 2016) was collected non-interactively and its descriptions are correspondingly longer. Ground truth is a single COCO instance per expression. *Open-vocabulary concept segmentation.* LVIS (Gupta et al., 2019) prompts with one of 1,203 long-tailed category names and is annotated federatedly, so absence is only established by its per-image verified-absent list, which is where our negative prompts are drawn from; COCO (Lin et al., 2014) has 80 exhaustively annotated categories, so any category without an annotation in an image is absent by construction. SA-Co/Gold (Carion et al., 2026) prompts with open noun phrases over seven domains, is triple-annotated (we use the first replica), and is dominated by negatives. In all three the ground truth is every instance of the concept, and the metric is the presence-gated cgF1 of Carion et al. (2026). *Semantic segmentation.* ADE20K-150 (Zhou et al., 2019) is class-prompted over 150 stuff-and-thing classes on 2,000 val images; Cityscapes (Cordts et al., 2016) contributes its 500 finely annotated val images and 19 evaluation classes. *Zero-shot detection transfer.* ODinW13 (Li et al., 2022a) and RF-100VL (Robicheaux et al., 2025) are collections of 13 and 100 small real-world detection datasets prompted with their own category names. They carry boxes and no masks, so the ground-truth region is the annotated box filled in, and the reported metric is box AP averaged over subsets.

Benchmark (splits)	Query (words)	Split	Neg.	Scored	Ground truth	Metric
ReasonSeg val / test	reasoning sentence (18.3 / 20.7)	200 / 779	2.0 / 0.3	all	one mask per query	gIoU, cIoU
RefCOCO val / testA / testB	referring expression (3.6)	10,834 / 5,657 / 5,095	0	500 each	one instance	gIoU, cIoU
RefCOCO+ val / testA / testB	no location words (3.6)	10,758 / 5,726 / 4,889	0	500 each	one instance	gIoU, cIoU
RefCOCOg val / test	written description (8.5)	4,896 / 9,602	0	500 each	one instance	gIoU, cIoU
LVIS val	category name (1.4)	129,547	45.9	1,500	every instance	cgF1, pmF1, AP
COCO val2017	category name (1.2)	29,631	50.6	1,500	every instance	cgF1, pmF1, AP
SA-Co/Gold	open noun phrase (2.6)	168,202	84.5	2,500	every instance	cgF1, pmF1
ADE20K-150 val	class name (1.0)	16,103	0	1,000	all pixels of class	gIoU, cIoU
Cityscapes val	19 class names in one query	500	0	500	19-class label map	mIoU
ODinW13 test	category name	6,237 (13 subsets)	0	all	box, filled	AP_0
RF-100VL test	category name	1,987 (100 subsets)	0	all	box, filled	AP_0

Negative shares are percentages. Unique images behind those items: 200 / 779 (ReasonSeg), 1,500 / 750 / 750 (the RefCOCO and RefCOCO+ splits), 1,300 / 2,600 (RefCOCOg), 19,809 (LVIS), 5,000 (COCO), 2,000 (ADE20K), 500 (Cityscapes), 4,812 (ODinW13) and 1,296 (RF-100VL). Negatives are intrinsic to SA-Co/Gold and to the empty-ground-truth items of ReasonSeg; for LVIS and COCO they are prompts we sample, so their share is a property of our protocol and not of the dataset.

Table 6: **Benchmarks.** Query type with mean query length in words, the size of each split as our loader builds it, the share of items whose correct answer is an empty mask, the number of items we score, and the reported metric. An item is one (image, query) pair.

C.2 Configuration of LLMSeg

The main results use Gemini-3.7-Flash (Comanici et al., 2025) as the backend for all three stages. Stage-2 curve degrees are the backend's own choice of $d \in \{1, 3\}$ shared across the instances in each call (Section 3) and refinement curves are always cubic; their rasterization is given in Section B.2. The budget constants and the stopping rule are given in Section 4. A full run costs 2 MLLM calls (detect and segment) plus at most 2 per refinement pass, so at most $2T+2 = 18$; measured over ReasonSeg val the mean is 13.4 calls per image (median 14), because the effect gate and early stopping end many loops well before T (Figure 19). Mean wall-clock time is 159 s per image. The stage-1 topology descriptor (Section 3) is an option rather than a fixed part of the configuration, and the main results

leave it *off* on all three stages; Section 5 and Section E.3 report both settings, and the difference between them sits inside the run-to-run band. These settings are used unchanged on every benchmark; nothing is tuned per dataset.

C.3 Evaluation protocol and matched-subset scoring

Evaluation protocol. Six ReasonSeg items, four of the 200 in val and two of the 779 in test, carry a legitimately *empty* ground-truth mask: the queried thing is absent from the image. We score these items rather than excluding them, so a system that correctly predicts nothing is credited and one that hallucinates a region is penalized. This is worth stating because dropping them raises the reported gIoU of any system that does not handle absence well, and would make our val numbers incomparable to the published ReasonSeg results we quote. For LVIS/COCO/ADE20K, presence-gated metrics use 3, 3, and 0 sampled negative prompts per positive respectively, seed 0, shared across all systems. Cityscapes mIoU is computed from corpus-level per-class intersections/unions over the 19 classes: SAM 3 is queried one class at a time; LLMSeg labels all classes of one image in a single query and groups spline instances by label. All backends use the same deterministic composition: stuff masks are painted before thing masks, with larger masks first within each group, so smaller masks overwrite larger ones. Uncovered pixels remain unassigned; no confidence argmax is used. ADE20K is class-prompted, item = (image, class): a union-IoU oracle upper bound, not comparable to published argmax mIoU.

Matched-subset scoring. Item keys are constructed in the data loader before the backend split, so every backend sees identical items. gIoU/cIoU/cgF1/pmF1/mIoU are recomputed over the key intersection from per-item components; AP is re-evaluated with COCOeval over the matched items only, with boxes derived from masks for all systems. Metrics for LLMSeg count pipeline failures (detect or segment errors) as zero.

Four benchmarks are run on their entire split, so for them the key intersection is the split itself and the numbers can be read against published full-split results: ReasonSeg val and test, Cityscapes val mIoU, and AP_0 on both detection collections, averaged over all 13 ODinW13 subsets and over the 99 of the 100 RF-100VL subsets on which AP is defined. Sampling costs us one metric, box and mask AP on LVIS and COCO. AP ranks detections across categories, so an image may enter the evaluation only when every ground-truth category it contains was queried. Our items are (image, category) pairs drawn uniformly, so a sampled image usually has just one of its categories queried and is excluded: 48 of 1,449 LVIS images and 62 of 1,304 COCO images survive, and the survivors skew towards images with few categories. Those two columns are therefore not comparable to published full-split federated AP, and we read them only as a comparison between systems on the same images. Every other metric we report is computed per item and is unaffected.

Table organization. Rows are methods (baselines and LLMSeg configurations) and columns are benchmarks, splits and metrics, so that adding a new baseline or a new LLMSeg configuration is one extra row rather than a new table. Shaded rows are LLMSeg. A dash marks a cell we do not report, either because the metric does not apply to that method or because the run is not part of this evaluation.

C.4 Reproducing the SAM 3 Agent baseline

SAM 3 Agent is the agent baseline of Carion et al. (2026), not part of LLMSeg. Our reproduction follows their reference agent implementation. The MLLM iterates over three tools: one runs SAM 3 on a simple noun phrase and returns the resulting masks as a numbered overlay, one selects the numbers matching the full query, and one reports that nothing in the image matches. The loop is capped at 8 rounds, and the union of the selected masks is scored through the same path as every other system. We run it with Gemma-4-31B and Gemini-3.7-Flash rather than the paper's Qwen2.5-VL and Gemini 2.5 Pro backends, so our SAM 3 Agent rows are somewhat below the published ones (Table 1).

C.5 Backend study protocol

Protocol. The backend study of Section 5 holds LLMSeg completely fixed and changes only which MLLM answers its three calls. All twenty-one backends run the full pipeline on all 200 ReasonSeg

val items with $T{=}12$ and every other setting fixed, including topology conditioning on. This differs from the main results ($T{=}8$, topology off). Metrics are recomputed from the per-item records rather than copied from any summary, so gIoU is the mean per-item IoU and cIoU is $\sum \mathrm{I}/\sum \mathrm{U}$ over all 200 items, with pipeline failures scored 0. Confidence intervals are percentile bootstraps over items (2000 resamples). Each call has a wall-clock budget; backends with high serving latency (marked † in Table 11) receive a longer one so that provider latency, not model capability, does not decide their score. The budget only bounds how long a call may take and does not alter any reply. Qwen3-VL-235B is reported but failed to return a usable reply on 49% of items, so its score is as much a reply-rate as a segmentation quality. Every row of Table 11 covers the same 200 items. The two reference lines of Figure 5 are SAM 3 and SAM 3 Agent scored through the same code on the same items, and are the ReasonSeg val entries of Table 7; SAM 3 Agent is shown with Gemini-3.7-Flash driving it, and moves by only 0.7 gIoU (72.4, 72.5, 73.1) across the three MLLMs we ran it with, against 10.8 for LLMSeg across one Gemini Flash product line.

Serving the backends through OpenRouter. The twenty-one backends reach the pipeline through two routes, and the route changes only how a request is transported, never the prompt, the parser or the geometry code. The eight Gemini and Gemma models are called through Google's native Gemini API. The other thirteen (the five OpenAI models, Claude Opus 5, the three Qwen models, the two GLM models and the two Muse models) are served through OpenRouter,[1] a hosted gateway that exposes models from many developers behind one OpenAI-compatible chat-completions interface, which we call through the open-source liteLLM client.[2] Each backend is addressed by a fixed OpenRouter model identifier, listed in the released code. Every request is a single user message holding the stage prompt as text and the image (the input photograph, or the rendered mask overlay during refinement) as a base64 data URL carrying the file's own bytes and MIME type. The image is never resized, because every coordinate the model returns is stated in that image's pixel frame; the one exception is an encoding above the 10 MB limit some upstream providers enforce, which is re-compressed at unchanged pixel dimensions. We set no sampling parameters: temperature, nucleus sampling, the output-token limit and the reasoning effort are left at each provider's defaults, so every backend is evaluated as its developer serves it, and OpenRouter's default routing chooses among the hosts that serve a model. Requests are rate-limited on the client side and retried up to five times with exponential backoff (1 s, doubling to a cap of 8 s, with jitter). A call that still fails, or that exceeds the per-call wall-clock budget above, is a pipeline failure, and the item is scored 0, the same rule as on the native route. OpenRouter's public model and endpoint listings are also the source of the prices and serving speeds used in Section E.6.

The capability index. Public benchmark coverage is ragged: no single published benchmark is reported for all twenty-one backends, and the benchmarks that do exist live on incomparable scales. We therefore build each capability axis by z-scoring every benchmark *within this set of backends* and averaging whichever of the axis's benchmarks a model publishes. Missing values are never imputed: a model with no published number for an axis is absent from that axis's panel, and Table 11 reports how many benchmarks back each model's index. The four axes are **software engineering** (AA Coding Index, SWE-bench Verified, SWE-bench Pro, LiveCodeBench), **reasoning and knowledge** (AA Intelligence Index, GPQA Diamond, HLE), **agentic and tool use** (AA Agentic Index, OSWorld, Terminal-Bench), and **vision and multimodal** (MMMU, MMMU-Pro, CharXiv); the general index is the same construction over all of them at once. Two consequences are worth stating plainly. The index is *relative*: adding a backend re-centres it, so its zero carries no meaning beyond this table. And the underlying numbers are vendor-reported, which is exactly why we compare rank correlations rather than reading a regression slope off the fit.

Backend model specifications. So that any accuracy number here can be read against price, speed and model size, we compiled one specification row per candidate MLLM backend (43 models, released with our code): identity and release date, size (total/active parameters, context window, modalities), cost per million tokens, measured serving throughput and time-to-first-token, and published programming (SWE-bench, Terminal-Bench, LiveCodeBench), reasoning (GPQA, HLE) and visual (MMMU, CharXiv, OSWorld) scores. Cost, context and speed are pulled programmatically from the OpenRouter model and endpoint APIs on one date; parameter counts from the Hugging Face weight indices; benchmark scores from vendor model cards, technical reports and launch posts,

[1] https://openrouter.ai
[2] https://github.com/BerriAI/litellm

with the few secondary-source or vendor-conflicting values marked in the cell. Empty means not published, never zero: closed-weight backends disclose no parameter count, and Terminal-Bench scores are only comparable within a benchmark version and agent scaffold, which we record inline.

C.6 COST ACCOUNTING

Protocol. Table 16 and Figure 29 price the same 200 ReasonSeg val items that Table 11 scores, for all twenty-one LLMSeg backends and for the SAM 3 and SAM 3 Agent baselines. Nothing here was re-run: every number is either measured from records those runs already wrote or quoted from the published model table. *Wall clock* is the median seconds per item over the items a backend answered, taken from per-item timers recorded during the runs; the median rather than the mean, because a run that hit its per-call budget on a handful of items has a mean that reports the timeout rather than the backend. *Calls* counts both kinds of model an image invokes. The MLLM figure is reconstructed from each item's recorded pass list — one detect and one segment call, plus one focus call per refinement pass and one batched spline call for every pass whose focus boxes survived the effect gate — and is the mean number of agent rounds for SAM 3 Agent. The SAM 3 figure is the number of forward passes of the 860 M-parameter model (840 M of which is the image detector; the tracker is idle on stills): one for SAM 3 itself, one per phrase the agent chooses to segment, and none for LLMSeg, which calls no segmentation model. Both were timed on one RTX-class GPU. Price, throughput and time-to-first-token are the providers' published figures.

The dollar column is an estimate, and only a ratio. The spend column is an estimate rather than a billed measurement. We price every row with one shared budget: 1900 input tokens per call (the stage prompt templates, which measure 522 to 953 tokens, plus one image at a full-resolution allowance of about 1.1k) and 175 output tokens per call (the mean structured payload recorded across all twenty-one runs). The column is a function of the published price and the measured call count and of nothing else. It therefore ranks backends correctly against each other while its absolute value is a lower bound: reasoning tokens are billed by several of these providers and are not counted here.

SAM 3 is not free either. We ran SAM 3 on a local GPU, so nothing was billed for it, but a forward pass has a market price and pricing it at zero would flatter LLMSeg. We charge every SAM 3 pass at \$0.0043 per image, the pay-as-you-go rate of a serverless endpoint serving the same checkpoint,[3] which puts SAM 3 at \$4.30 per thousand images for its one pass and adds 1.4 passes' worth to each SAM 3 Agent image on top of its MLLM calls. LLMSeg calls no segmentation model, so it carries no such term.

[3] Segmind serverless inference, pay-as-you-go credits, retrieved 2026-09.

D Additional Results

D.1 Complete reasoning and referring comparisons

Table 1 combines ReasonSeg and RefCOCO/+/g in cIoU with selected baselines. The complete tables, including gIoU and the training columns, are reproduced here.

Method	MLLM	Training		val		test	
		RES	ReasonSeg	gIoU	cIoU	gIoU	cIoU
Published, full splits (Carion et al., 2026)							
SEEM (Zou et al., 2023)	–	✗	✗	25.5	21.2	24.3	18.7
Grounded SAM (Ren et al., 2024)	–	✗	✗	26.0	14.5	21.3	16.4
OVSeg (Liang et al., 2023)	–	✗	✗	28.5	18.6	26.1	20.8
GLaMM (Rasheed et al., 2024)	Vicuna 7B	✓	✗	47.4	47.2	–	–
SAM4MLLM (Chen et al., 2024)	Qwen-VL 7B	✓	✗	46.7	48.1	–	–
SAM4MLLM (Chen et al., 2024)	LLaVA1.6 8B	✓	✗	58.4	60.4	–	–
Seg-Zero (Liu et al., 2025)	Qwen2.5-VL 3B	✓	✗	58.2	53.1	56.1	48.6
Seg-Zero (Liu et al., 2025)	Qwen2.5-VL 7B	✓	✗	62.6	62.0	57.5	52.0
X-SAM (Wang et al., 2025a)	Phi3 3.8B	✓	✓	56.6	32.9	57.8	41.0
HyperSeg (Wei et al., 2024)	Phi2 3B	✓	✓	59.2	56.7	–	–
Kang et al. (Kang et al., 2025)	LLaVA1.5 7B	✗	✗	–	52.4	–	48.7
Kang et al. (Kang et al., 2025)	LLaVA1.5 13B	✗	✗	–	60.5	–	49.9
LISA (Lai et al., 2024)	LLaVA 7B	✓	✗	44.4	46.0	36.8	34.1
LISA (Lai et al., 2024)	LLaVA 7B	✓	✓	52.9	54.0	47.3	34.1
LISA (Lai et al., 2024)	LLaVA 13B	✓	✗	48.9	46.9	44.8	45.8
LISA (Lai et al., 2024)	LLaVA 13B	✓	✓	56.2	62.9	51.7	51.1
LISA (Lai et al., 2024)	Llama2 13B	✓	✓	60.0	67.8	51.5	51.3
LISA (Lai et al., 2024)	LLaVA1.5 7B	✓	✗	53.6	52.3	48.8	47.1
LISA (Lai et al., 2024)	LLaVA1.5 7B	✓	✓	61.3	62.9	55.6	56.9
LISA (Lai et al., 2024)	LLaVA1.5 13B	✗	✗	57.7	60.3	53.8	50.8
LISA (Lai et al., 2024)	LLaVA1.5 13B	✓	✓	65.0	72.9	61.3	62.2
RSVP (Lu et al., 2025)	LLaVA1.6 7B	✗	✗	59.2	56.7	56.9	50.7
RSVP (Lu et al., 2025)	Qwen2-VL 7B	✗	✗	58.6	48.5	56.1	51.6
RSVP (Lu et al., 2025)	Gemini1.5-Flash	✗	✗	56.9	49.2	57.1	59.2
RSVP (Lu et al., 2025)	GPT-4o	✗	✗	64.7	63.1	60.3	60.0
Gemini Seg (Voigtlaender et al., 2025)	Gemini2.5 Flash	?	?	28.3	13.3	30.6	9.2
SAM 3 Agent (Carion et al., 2026)	Qwen2.5-VL 7B	✗	✗	62.2	49.1	63.0	53.5
SAM 3 Agent (Carion et al., 2026)	Qwen2.5-VL 72B	✗	✗	74.6	65.1	70.8	64.0
SAM 3 Agent (Carion et al., 2026)	Llama4 Maverick	✗	✗	68.5	61.5	67.1	60.9
SAM 3 Agent (Carion et al., 2026)	Qwen3-VL 8B Think	✗	✗	68.5	57.4	70.2	67.3
SAM 3 Agent (Carion et al., 2026)	Qwen3-VL 235B Think	✗	✗	76.6	**67.0**	73.7	71.8
SAM 3 Agent (Carion et al., 2026)	Gemini2.5 Pro	✗	✗	**77.0**	66.0	**74.0**	**72.1**
SAM 3 (text query) (Carion et al., 2026)	–	✗	✗	13.1	9.9	8.4	11.3
SAM 3 Agent (Carion et al., 2026)	Gemma-4-31B	✗	✗	**73.1**	63.9	**72.2**	**66.7**
SAM 3 Agent (Carion et al., 2026)	Gemini-3.7-Flash	✗	✗	72.4	62.0	**72.2**	63.2
LLMSeg (w/o refine)	Gemini-3.7-Flash	✗	✗	64.9	64.6	62.7	58.2
LLMSeg (full)	Gemini-3.7-Flash	✗	✗	69.6	**71.5**	68.3	65.8

Table 7: **Reasoning segmentation on ReasonSeg** (Lai et al., 2024). Top block: published numbers quoted from Carion et al. (2026) on the full splits, reproducing its complete baseline list. Bottom block (below the rule): our own evaluations (val 200 items, test 779; SAM 3 Agent with Gemma-4-31B is scored on 778 test items). Both blocks target the full splits. SAM 3 and SAM 3 Agent are both baselines from Carion et al. (2026); only the shaded rows are ours. The two training columns follow Carion et al. (2026): RES = fine-tuned for referring expression segmentation, ReasonSeg = fine-tuned on the ReasonSeg training split; a method is zero-shot when both are ✗, and "?" marks training the source does not report. LLMSeg (w/o refine) is our pipeline stopped after the segment stage. Best zero-shot per column in **bold**, within each block.

SAM 3 Agent baseline with a Gemini-3.7-Flash backend. Swapping the baseline's agent MLLM from Gemma-4-31B to Gemini-3.7-Flash gives 72.4/62.0 (val gIoU/cIoU) and 69.6/61.8 (test subset), essentially matching the Gemma run. Since LLMSeg uses Gemini-3.7-Flash as well, this isolates the comparison: with the same MLLM on both sides, the difference between LLMSeg and the SAM 3 Agent baseline is the mask producer (splines vs. SAM 3), not the language model.

D.2 Complete open-vocabulary, semantic, and detection comparisons

The original versions of Tables 3 and 2, including the MLLM column and all metrics, are reproduced below.

Method	MLLM / backbone	RES	RefCOCO			RefCOCO+			RefCOCOg	
			val	testA	testB	val	testA	testB	val (U)	test (U)
Published, full splits, cIoU (Carion et al., 2026)										
LISA (Lai et al., 2024)	LLaVA 7B	✓	74.9	79.1	72.3	65.1	70.8	58.1	67.9	70.6
GSVA (Xia et al., 2024)	13B	✓	79.2	81.7	77.1	70.3	73.8	63.6	75.7	77.0
GLaMM (Rasheed et al., 2024)	Vicuna 7B	✓	79.5	83.2	76.9	72.6	78.7	64.6	74.2	74.9
SAM4MLLM (Chen et al., 2024)	LLaVA1.6 7B	✓	79.6	82.8	76.1	73.5	77.8	65.8	74.5	75.6
SAM4MLLM (Chen et al., 2024)	LLaVA1.6 8B	✓	79.8	82.7	74.7	74.6	80.0	67.2	75.5	76.4
GLEE (Wu et al., 2024b)	Plus	✓	79.5	–	–	68.3	–	–	70.6	–
GLEE (Wu et al., 2024b)	Pro	✓	80.0	–	–	69.6	–	–	72.9	–
DETRIS (Huang et al., 2025)	DETRIS-L	✓	81.0	81.9	79.0	75.2	78.6	70.2	74.6	75.3
UniLSeg (Liu et al., 2024b)	UniLSeg-20	✓	80.5	81.8	78.4	72.7	77.0	67.0	78.4	79.5
UniLSeg (Liu et al., 2024b)	UniLSeg-100	✓	81.7	83.2	79.9	73.2	78.3	68.2	79.3	80.5
PSALM (Zhang et al., 2024c)	Phi1.5 1.3B	✓	83.6	84.7	81.6	72.9	75.5	70.1	73.8	74.4
EVF-SAM (Zhang et al., 2024b)	RC	✓	82.1	83.7	80.0	75.2	78.3	70.1	76.8	77.4
EVF-SAM (Zhang et al., 2024b)	Extra Data	✓	82.4	84.2	80.2	76.5	80.0	71.9	78.2	78.3
RICE (Xie et al., 2025)	Qwen2.5-7B	✓	83.5	85.3	81.7	**79.4**	82.8	75.4	79.8	80.4
MLCD-seg (An et al., 2024)	Qwen2.5-7B	✓	83.6	85.3	81.5	**79.4**	82.9	**75.6**	79.7	80.5
HyperSeg (Wei et al., 2024)	Phi2 2.7B	✓	84.8	85.7	**83.4**	79.0	**83.5**	75.2	79.4	78.9
X-SAM (Wang et al., 2025a)	Phi3 3.8B	✓	**85.1**	**87.1**	**83.4**	78.0	81.0	74.4	**83.8**	**83.9**
GL-CLIP (Yu et al., 2023)	ResNet-50	✗	32.7	35.3	30.1	37.7	40.7	34.9	41.6	42.9
GL-CLIP (Yu et al., 2023)	ViT-B/32	✗	32.9	34.9	30.1	38.4	42.1	32.7	42.0	42.0
CaR (Sun et al., 2024)	ViT-B/16	✗	33.6	35.4	30.5	34.2	36.0	31.0	36.7	36.6
Ref-Diff (Ni et al., 2023)	VAE	✗	37.2	38.4	37.2	37.3	40.5	33.0	44.0	44.5
TAS (Suo et al., 2023)	ResNet-50	✗	39.9	42.9	35.9	44.0	50.6	36.4	47.7	47.4
TAS (Suo et al., 2023)	ViT-B/32	✗	39.8	41.1	36.2	43.6	49.1	36.5	46.6	46.8
IteRPrimeE (Wang et al., 2025c)	–	✗	40.2	46.5	33.9	44.2	51.6	35.3	46.0	45.1
Pseudo-RIS (Yu et al., 2024)	CRIS	✗	39.8	44.8	33.0	42.2	46.3	34.5	43.7	43.4
Pseudo-RIS (Yu et al., 2024)	ETRIS	✗	41.1	48.2	33.5	44.3	51.4	35.1	46.0	46.7
LGD+DINO (Li et al., 2025)	ViT-B/32	✗	49.5	54.7	41.0	49.6	58.4	38.6	50.3	51.1
VLM-VG (Wang et al., 2025b)	ResNet-50	✗	47.7	51.8	44.7	41.2	45.9	34.7	46.6	47.1
VLM-VG (Wang et al., 2025b)	ResNet-101	✗	49.9	53.1	46.7	42.7	47.3	36.2	48.0	48.5
HybridGL (Liu & Li, 2025)	ViT-B/32	✗	49.5	53.4	45.2	43.4	49.1	37.2	51.3	51.6
Kang et al. (2025)	LLaVA-1.5 7B	✗	74.2	76.5	70.4	62.5	65.2	56.0	64.2	68.1
Kang et al. (2025)	LLaVA-1.5 13B	✗	76.1	78.9	72.8	64.1	67.1	57.3	67.7	69.0
SAM 3 Agent (Carion et al., 2026)	Qwen2.5-VL 7B	✗	59.4	64.3	55.0	51.4	57.0	44.9	57.2	58.8
SAM 3 Agent (Carion et al., 2026)	Qwen2.5-VL 72B	✗	71.6	74.9	66.0	64.9	70.8	57.9	68.8	70.2
SAM 3 Agent (Carion et al., 2026)	Llama4 Maverick	✗	71.7	76.5	66.5	65.1	71.2	57.6	67.7	68.7
SAM 3 Agent (Carion et al., 2026)	Qwen3-VL 8B Think	✗	68.6	72.3	63.9	59.3	64.5	55.6	66.4	66.9
SAM 3 Agent (Carion et al., 2026)	Qwen3-VL 235 Think	✗	73.9	77.0	69.7	66.9	70.9	62.2	72.1	72.9
SAM 3 Agent (Carion et al., 2026)	Gemini 2.5 Pro	✗	75.5	77.6	71.0	67.3	71.1	63.4	73.4	74.0
Our runs, matched 500-item subsets, gIoU										
SAM 3 (text query)	–	✗	38.7	40.9	40.9	32.1	35.0	28.0	40.0	38.7
SAM 3 Agent	Gemma-4-31B	✗	74.9	75.6	**76.7**	69.8	72.9	67.1	75.5	76.5
SAM 3 Agent	Gemini-3.7-Flash	✗	**75.3**	**76.9**	76.1	**71.2**	**74.2**	**71.2**	**78.4**	**77.3**
LLMSeg (w/o refine)	Gemini-3.7-Flash	✗	66.0	66.2	72.1	61.9	63.3	64.0	67.1	68.5
LLMSeg (full)	Gemini-3.7-Flash	✗	71.6	73.0	74.8	68.2	70.8	67.6	72.7	73.0
Our runs, matched 500-item subsets, cIoU										
SAM 3 (text query)	–	✗	42.0	42.8	44.2	33.6	37.0	32.9	40.0	39.7
SAM 3 Agent	Gemma-4-31B	✗	**73.2**	75.6	75.0	67.2	**72.0**	61.9	72.1	74.7
SAM 3 Agent	Gemini-3.7-Flash	✗	**73.2**	**76.7**	75.5	**68.4**	**72.0**	**68.5**	**76.5**	**76.9**
LLMSeg (w/o refine)	Gemini-3.7-Flash	✗	65.5	65.7	72.9	59.7	61.4	61.1	64.8	69.3
LLMSeg (full)	Gemini-3.7-Flash	✗	71.2	73.8	**76.0**	67.0	69.9	64.9	70.9	74.5

Table 8: **Referring expression segmentation** on RefCOCO/+/g (Kazemzadeh et al., 2014; Mao et al., 2016). *Top block*: published full-split cIoU quoted from Carion et al. (2026), separated into methods fine-tuned on the RES task (RES ✓) and zero-shot methods (RES ✗); note that nearly all of the MLLM-based rows were trained on the RefCOCO *box* annotations. *Bottom blocks*: our runs on matched 500-item random subsets per split (shuffled, seed 0), upper block gIoU and lower block cIoU. All rows are zero-shot for RES and use the same selected items (SAM 3 Agent with Gemma-4-31B is scored on 499 of the 500 RefCOCO+ testA items). SAM 3 and SAM 3 Agent are baselines from Carion et al. (2026), reproduced by us on these items; only the shaded rows are ours. Best within each block in **bold**, and in the published block the best zero-shot entry is underlined. RefCOCOg val (U)/test (U) are the splits we run.

Method	MLLM	LVIS				COCO				SA-Co/Gold	
		cgF1	pmF1	box AP	mask AP	cgF1	pmF1	box AP	mask AP	cgF1	pmF1
Human	–	–	–	–	–	–	–	–	–	72.8	–
OWLv2 (Minderer et al., 2024)	–	20.1	–	35.2	–	–	–	38.2	–	17.3	–
OWLv2 (LVIS-trained)	–	29.3	–	45.5	43.4	–	–	46.1	–	24.6	–
gDino-T (Liu et al., 2024a)	–	14.7	–	20.5	–	–	–	45.7	–	3.3	–
LLMDet-L (Fu et al., 2025)	–	35.1	–	42.0	36.3	–	–	55.6	–	6.5	–
APE-D (LVIS-trained) (Shen et al., 2024)	–	–	–	59.6	53.0	–	–	58.3	–	16.4	–
DINO-X (Ren et al., 2025)	–	–	–	52.4	38.5	–	–	56.0	–	21.3	–
Gemini 2.5 Flash (Comanici et al., 2025)	–	13.4	–	–	–	–	–	–	–	13.0	–
SAM 3 (Carion et al., 2026)	–	37.2	–	53.6	48.5	–	–	56.4	–	**54.1**	–
SAM 3*	–	50.1	58.0	**69.1**	**62.2**	52.4	57.6	**68.0**	**59.2**	45.4	55.2
SAM 3 Agent*	Gemma-4-31B	51.0	**62.4**	56.2	50.9	49.9	58.6	48.9	42.0	–	–
LLMSeg (w/o refine)	Gemini-3.7-Flash	50.4	53.0	57.1	36.9	50.4	54.0	59.6	38.0	31.9	48.6
LLMSeg (full)	Gemini-3.7-Flash	**53.9**	57.3	64.2	47.2	**56.1**	**60.6**	66.6	50.5	36.4	**55.4**

Table 9: **Open-vocabulary instance segmentation.** Unmarked rows are published full-split numbers quoted from Carion et al. (2026). * Our evaluation of the baseline on matched random subsets with hard negatives, the same items as the shaded rows, which are ours. cgF1 is presence-gated F1 and pmF1 its positive-only counterpart. **Best** and second best per column; the human reference is not ranked.

Method	Semantic segmentation			Zero-shot detection							
	ADE20K-150		Cityscapes	ODinW13				RF-100VL			
	gIoU	cIoU	mIoU	AP_0	gIoU	cIoU	pmF1	AP_0	gIoU	cIoU	pmF1
APE-D (LVIS-trained)	–	–	44.2	–	–	–	–	–	–	–	–
Gemini 2.5 Pro	–	–	–	33.7	–	–	–	11.6	–	–	–
gDino-T	–	–	–	49.7	–	–	–	15.7	–	–	–
gDino1.5-Pro	–	–	–	58.7	–	–	–	–	–	–	–
SAM 3	–	–	**65.2**	**61.0**	–	–	–	15.2	–	–	–
SAM 3*	57.9	**67.3**	63.7	51.5	48.3	51.5	39.6	14.2	13.8	21.7	11.4
LLMSeg (w/o refine)	56.6	60.5	48.8	52.0	48.7	48.7	50.0	30.4	42.1	50.8	58.7
LLMSeg (full)	**59.2**	64.7	53.3	57.2	**54.2**	**55.7**	**57.5**	**31.4**	**43.6**	**53.2**	**61.7**

Table 10: **Semantic segmentation and zero-shot detection.** Unmarked rows are published full-split numbers quoted from Carion et al. (2026). * Our evaluation of the baseline: complete Cityscapes, ODinW13 and RF-100VL splits, and on ADE20K the same 1,000 random items as the shaded rows, which are ours. ADE20K reports oracle-class union-IoU; Cityscapes reports 19-class mIoU. ODinW13 and RF-100VL report mean per-subset box AP_0 and pooled mask metrics. **Best** and second best per column.

D.3 Qualitative results on the main benchmarks

The grids below extend the qualitative evidence of Section 4.2; they use the same Gemini-3.7-Flash runs that back the tables, and each chip is the IoU the harness recorded for that panel. Figure 10 isolates what refinement does on every benchmark family, Figure 11 shows full Cityscapes label maps, Figure 12 sets LLMSeg beside both SAM 3 baselines, Figure 13 shows refinement repairing a trace, and Figure 14 collects the failure modes.

D.4 Crowded scenes, negatives, rare domains and single passes

Four grids extend the qualitative evidence of Section 4.2 to the cases the main paper has room to describe only in aggregate: images holding many objects of the queried class (Figure 15), SA-Co prompts whose concept is absent from the image (Figure 16), the rare visual domains of RF-100VL (Figure 17), and what an individual refinement pass does (Figure 18). Every panel comes from the same Gemini-3.7-Flash runs at $T{=}12$ that back the tables, every baseline panel from the SAM 3 and SAM 3 Agent runs scored on the identical item, and each chip is that system's recorded IoU, so a number here equals a number there.

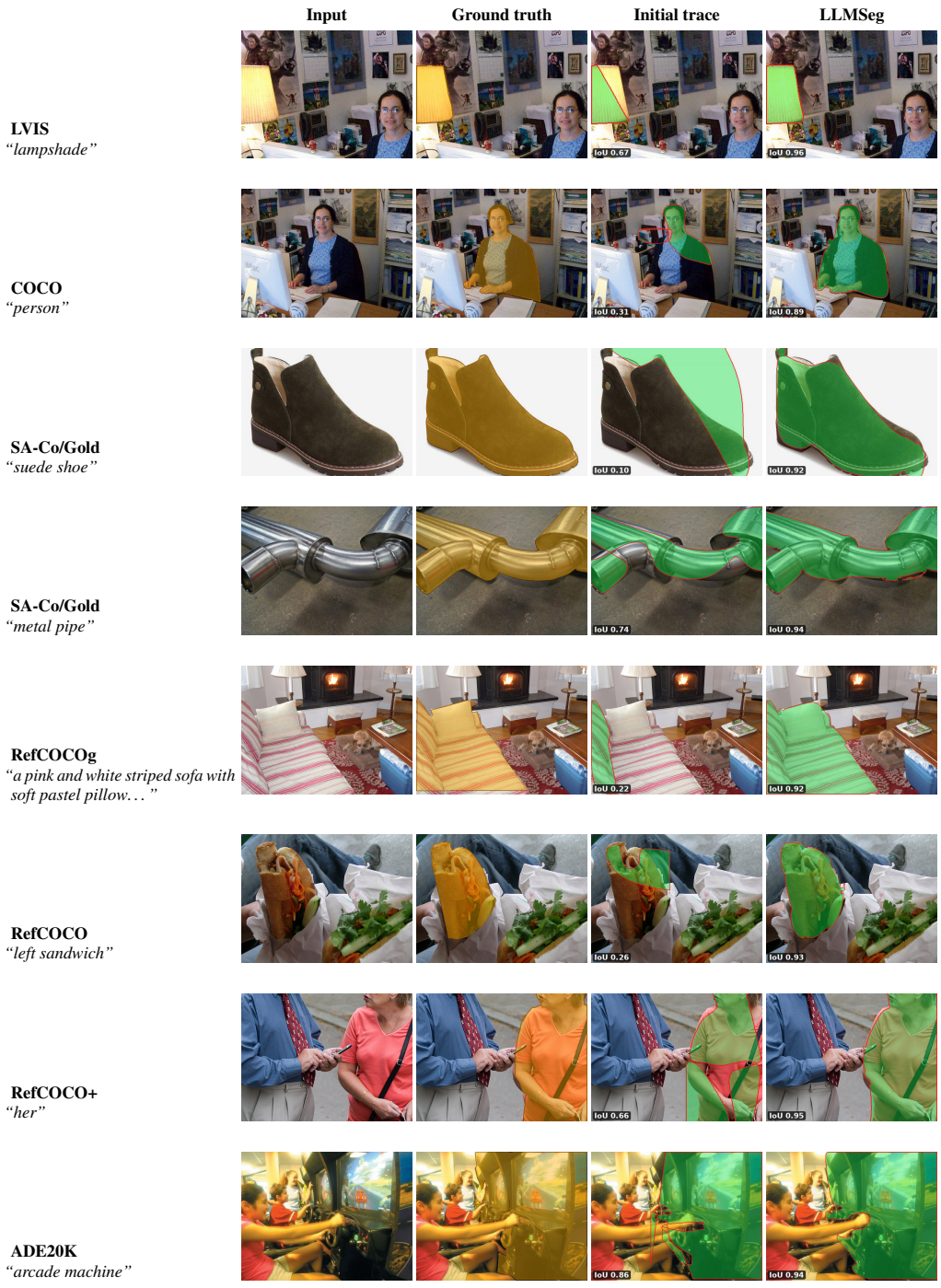


Figure 10: **LLMSeg across benchmarks and task formats**, same Gemini-3.7-Flash backend and settings throughout. The prompt is the text on the left, ground truth is shaded amber, predictions green with a red outline, and each chip is that panel's IoU. Column 3 is the stage-2 mask and column 4 the final one, so the pair isolates what refinement does.

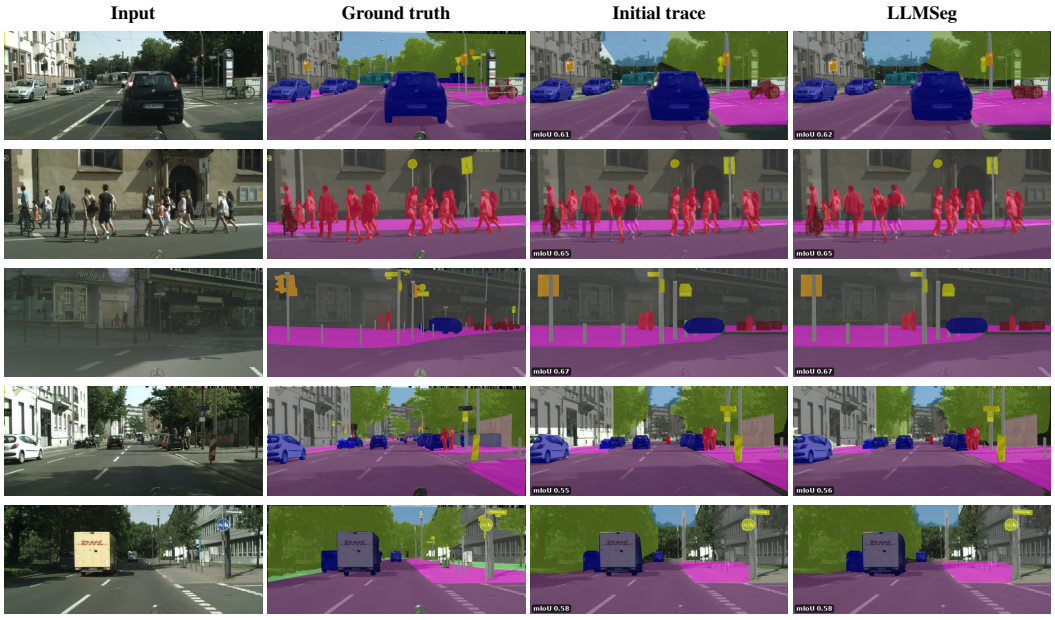


Figure 11: **Dense semantic labeling on Cityscapes** (Cordts et al., 2016), Gemini-3.7-Flash backend, official 19-class palette. LLMSeg labels every class of an image from a single query and resolves overlaps by deterministic mask compositing, so each panel is a full label map rather than one mask; the chip gives that image's mIoU. The losses are on the amorphous ground classes, whose extent has no visual boundary to trace.

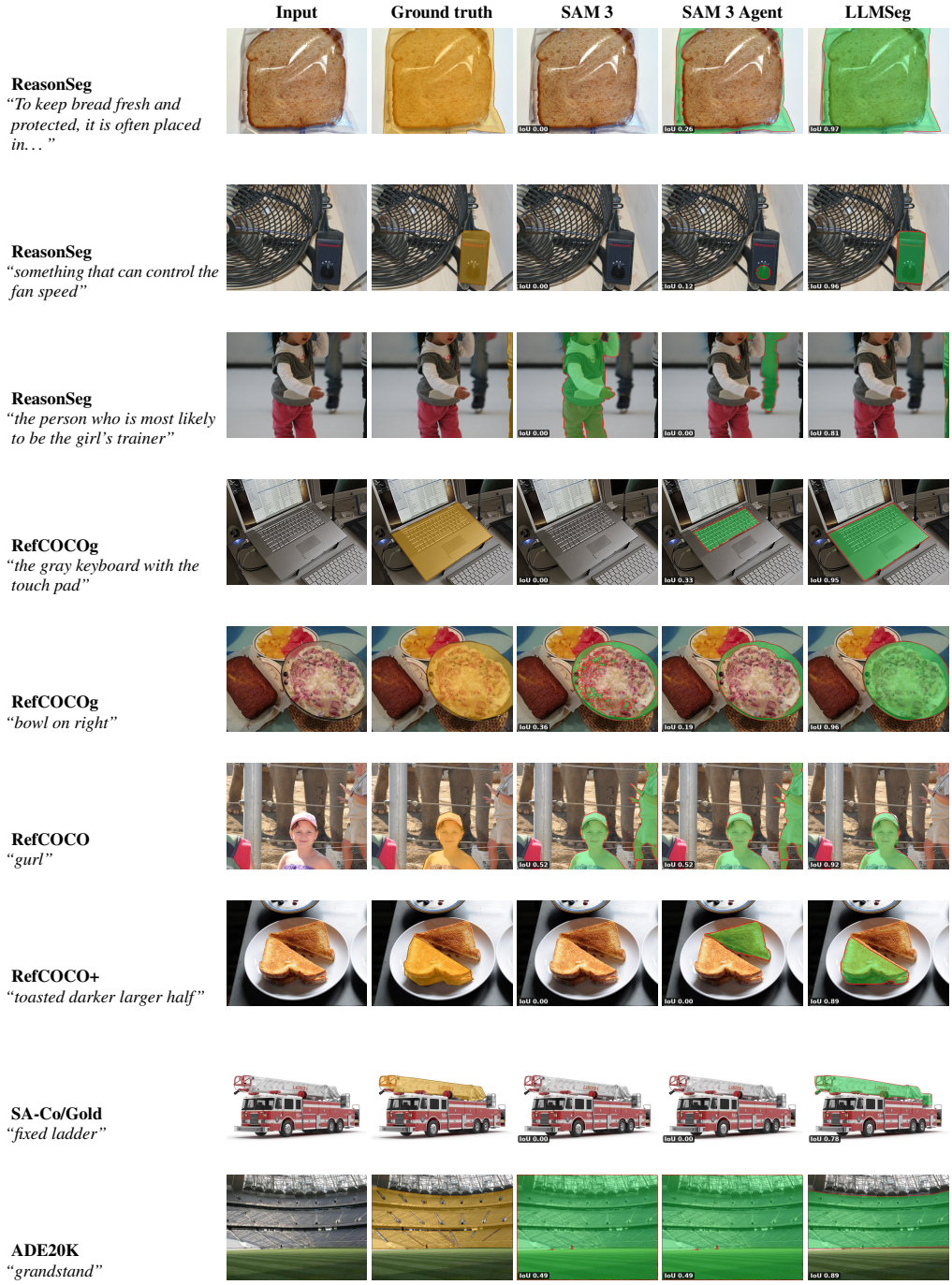
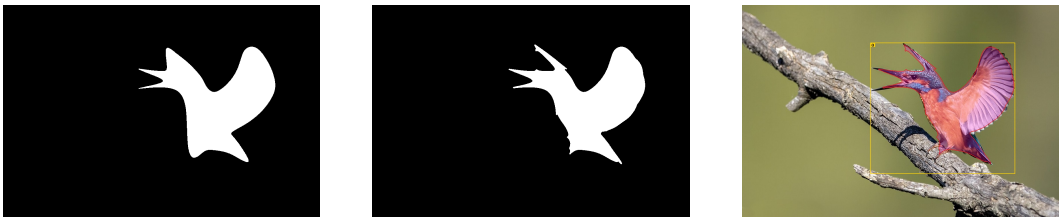
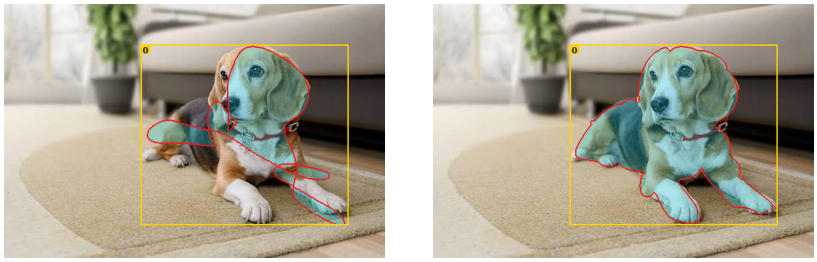


Figure 12: **Where LLMSeg separates from both SAM 3 baselines.** Selected items drawn from every benchmark for which all three systems were run; SAM 3 (Carion et al., 2026) receives the same text query as LLMSeg, and each chip is that system's recorded IoU. These are chosen wins: they show what the baselines' failure modes look like, not how often they occur.

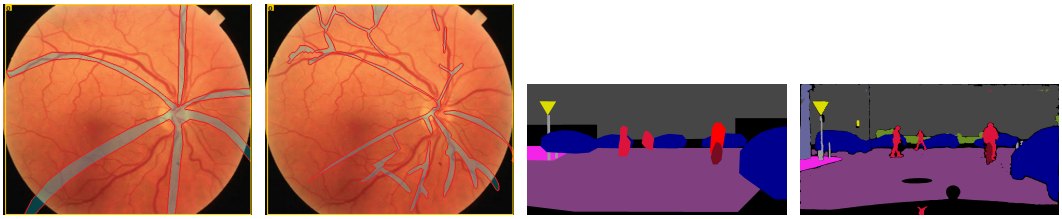


w/o refine full full (overlay)



w/o refine full

Figure 13: **What refinement fixes.** Top: kingfisher mask before (left) and after (middle) focus-box refinement, recovering the wing feathers and beak, with the final spline overlay on the right. Bottom: refinement also recovers from a *failed* first trace, because each pass re-traces loops confined to tight focus boxes rather than editing the broken outline.



w/o refine full (7 passes) LLMSeg label map SAM 3 label map



input ground truth LLMSeg SAM 3

Figure 14: **Failure modes.** Top left pair: thin structures (eyelashes) defeat spline outlines even after refinement, though SAM 3 struggles here too. Top right pair: on Cityscapes, amorphous and heavily-occluded background classes favour pixel decoders over compact parametric loops. Bottom row: the query is grounded correctly, but a closed loop cannot cut into the deep, narrow concavities of an articulated body.

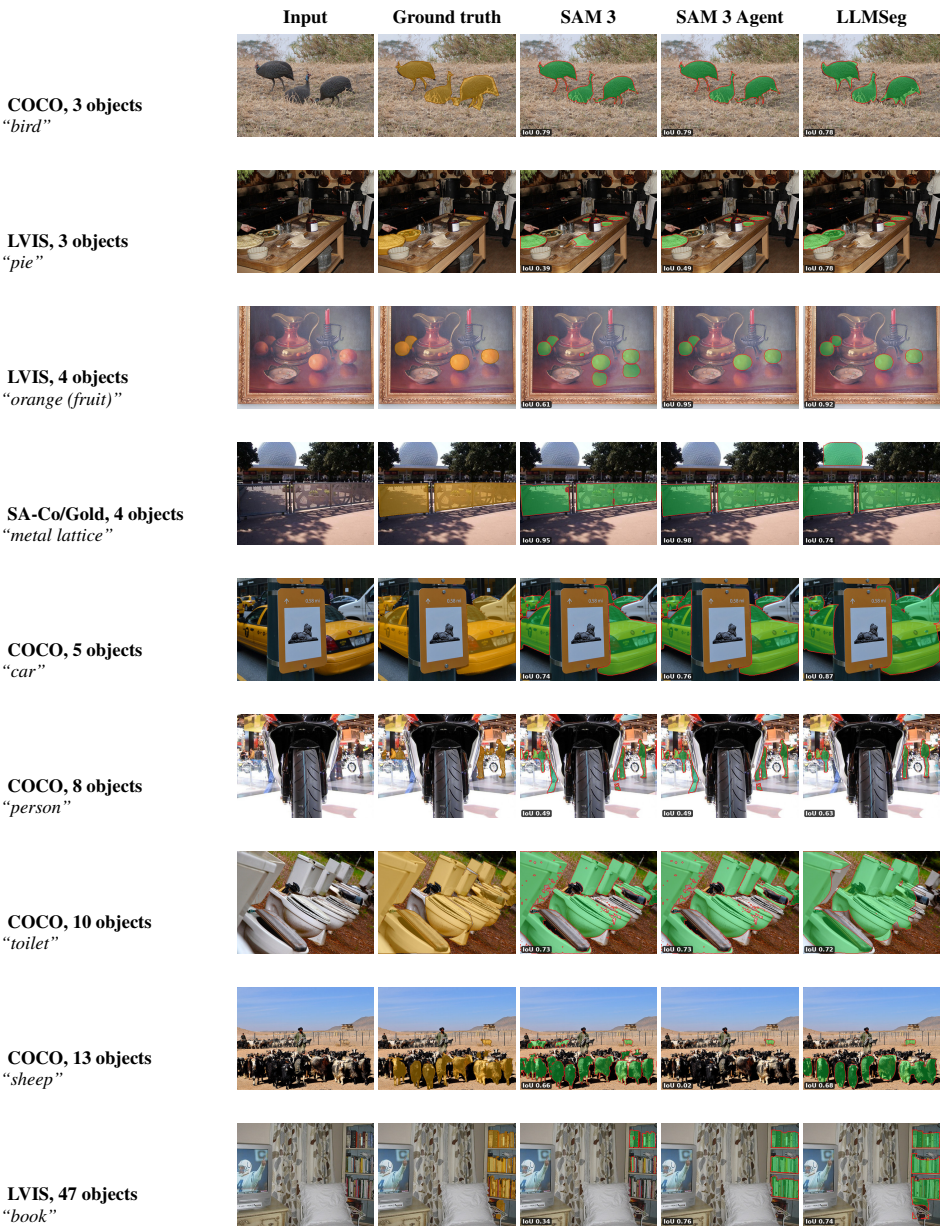


Figure 15: **Images holding several objects of the queried class.** LVIS, COCO and SA-Co/Gold items whose ground truth holds at least three objects, sampled to span the outcomes rather than to favour LLMSeg. The label gives the benchmark and the number of ground-truth objects, and the chip is that system's recorded union IoU. When the objects are many and touching, all three systems degrade into one connected region.

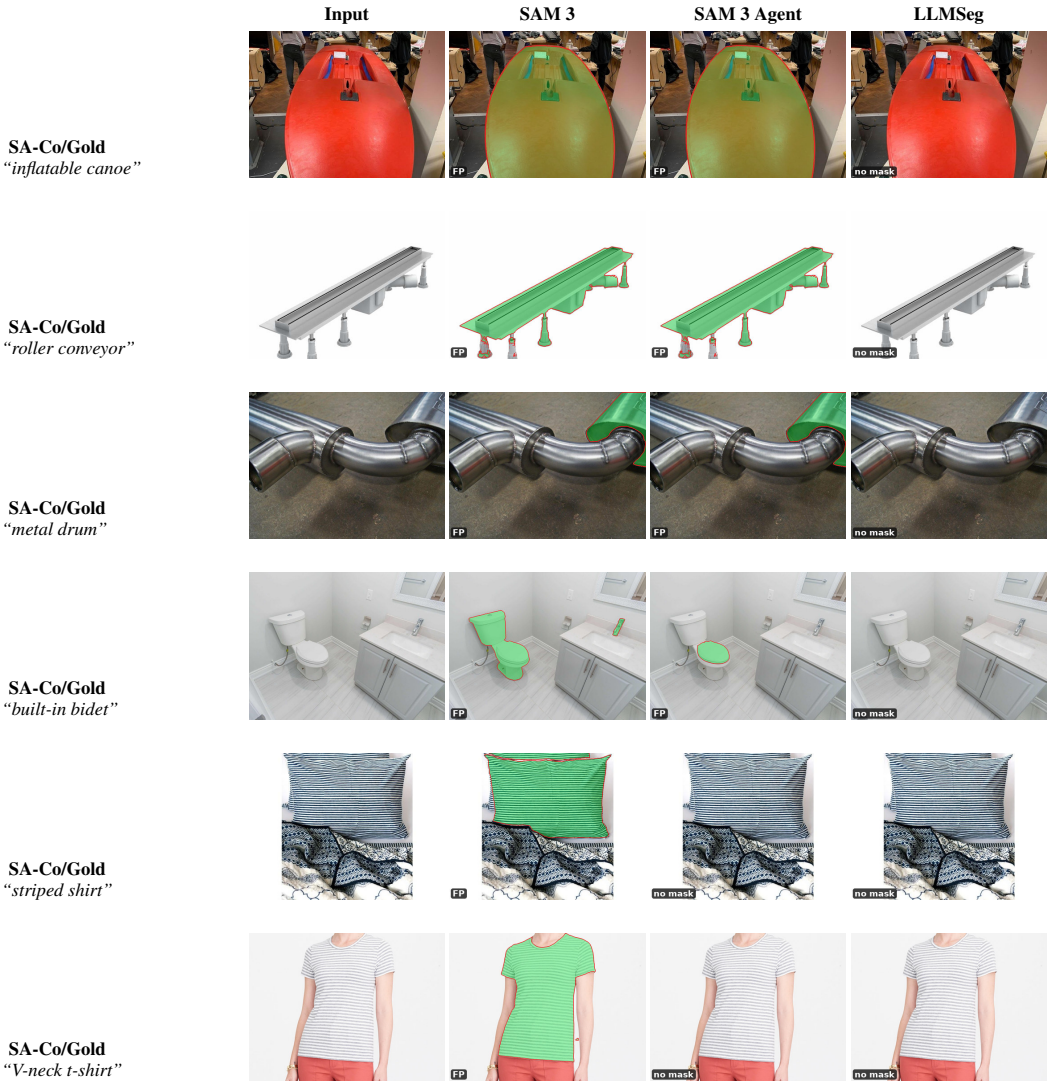


Figure 16: **Hard negatives: the prompt names an attribute the object does not have.** Six SA-Co/Gold items whose phrase has no annotation in the image. The ground truth is empty, so there is no ground-truth panel and the chip reports only whether a system fired. The scene always contains the *category*, never the qualified concept. LLMSeg still loses SA-Co/Gold overall (Table 3).

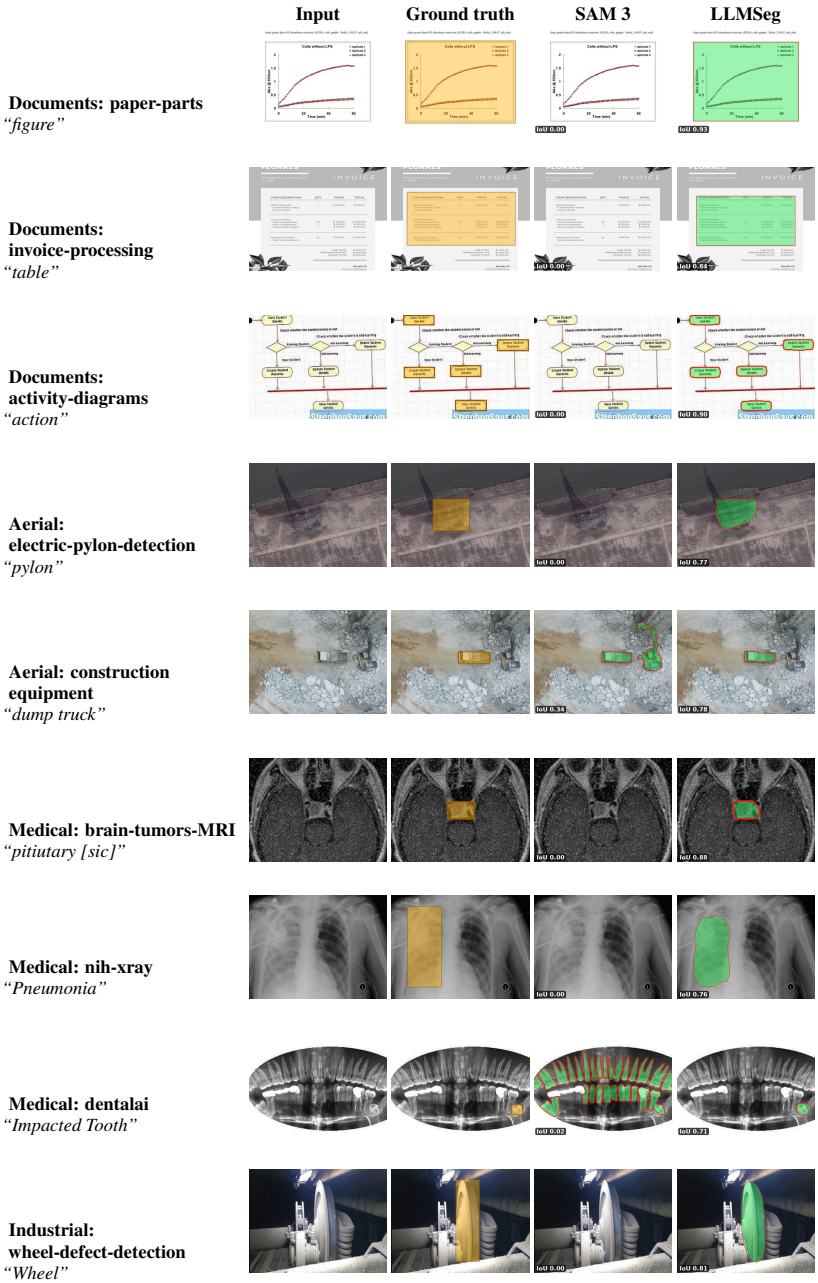


Figure 17: **Rare domains: nine RF-100VL subsets, four domain families.** Documents, aerial imagery, medical imaging and industrial inspection, one subset per row, prompted with the subset's own category name verbatim. Its ground truth is the annotated detection box filled in, and rows were chosen for legibility at print size. The failure shown is vocabulary, not localization.



Figure 18: **What one refinement pass does.** Four items on which the final mask is far above the stage-2 trace, one per task format, each shown at stage 2, after the first two passes, and after its last pass; the chip names the state and gives its IoU. The intermediate masks are replayed from the item’s refinement record with the pipeline’s own composition code.

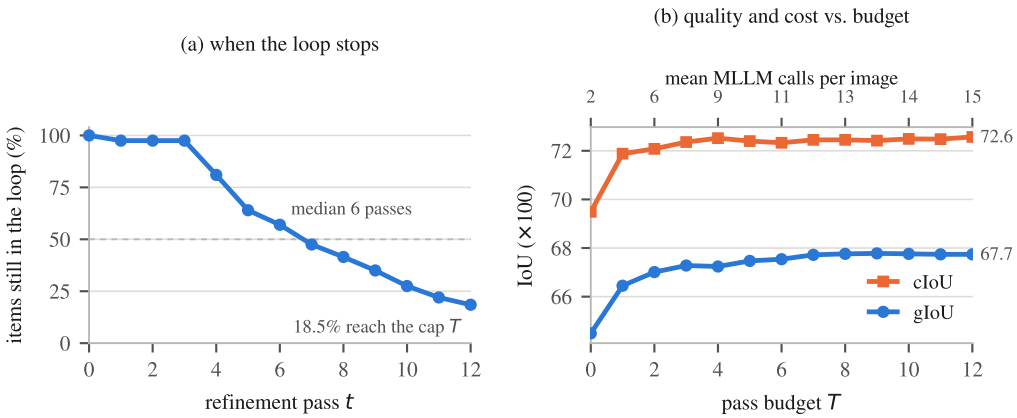


Figure 19: **The refinement loop stops on its own.** ReasonSeg val, all 200 items, Gemini-3.7-Flash. **(a)** Share of items still inside the loop at each pass under the stopping criterion of Section 4. **(b)** The same run's mask after the first T passes, scored at every budget (Figure 6), with the mean MLLM calls on the top axis.

E Extended Analysis

This section expands the findings of Section 5. The complete ablation results are in Table 13 and Table 14. The ablation suite uses T=12 and a topology-on reference; the main results use T=8 with topology off. Comparisons to the topology-off ablation row therefore describe configuration differences, rather than isolated effects of the other component being changed.

E.1 Refinement and convergence

Effect of iterative refinement. Refinement improves every benchmark in Table 1 to 2 (e.g. +8.6 cIoU on RefCOCO+ testA, +5.8 cgF1 on COCO), and Figure 13 shows why: the first confined trace captures the gestalt, and focus boxes then act as local zoom, where splines confined to a tight box come out substantially more accurate.

Does the loop actually converge?. This matters more than a typical hyperparameter question. Eq. (3) is an open loop driven by a stochastic model that is free to propose a fresh set of focus boxes every pass, and nothing in it is a contraction. Left alone, such a loop can cycle indefinitely, repainting a boundary one way and then the other, so a reader is right to ask whether the pass budget T is a genuine early stop or just a truncation that hides an unstable process.

Figure 19(a) answers it empirically on ReasonSeg val. The stopping criterion of Section 4 ends the loop for 79% of items before the budget is reached (81% of those that enter it at all); the median item uses 6 passes, 36% stop within 4, and only 18.5% are still running when T cuts them off. This behavior is consistent with the intended damping mechanism: once successive passes agree on a boundary the commitment field makes those pixels progressively harder to overturn, although the study does not isolate the soft lock's contribution. The minority that reaches the cap is a harder set to begin with, already 5.7 gIoU behind at the stage-2 mask (60.0 vs. 65.7) and still 7.0 behind at the end (62.3 vs. 69.3). Normalizing the criterion by box area rather than image area makes its threshold relative to the region the loop can alter. Per-item residual traces, on the development set where ε and s were chosen, are in Figure 7 of Section B.3.

Figure 19(b) ties this to accuracy rather than to mask stability alone. Quality saturates well inside the budget: the first pass buys +2.0 gIoU, four passes buy +2.7 of the eventual +3.2, and passes 8 to 12 move it by less than 0.1. Because most items have already stopped, the measured cost is far below the $2T+2 = 26$ call worst case: 14.7 calls per image on average at T=12, and about 9 at the T=4 knee. T is therefore a safety net rather than an operating point.

E.2 BACKEND STUDY

Because LLMSeg adds no trained parameters, backend progress is the only thing that has to improve for segmentation to improve. We test that directly: the same pipeline, at the study configuration (T=12, topology conditioning on), run over ReasonSeg val with **twenty-one** MLLM backends from six developers, changing nothing but which model answers the three calls. Figure 21 is the complete ranking, read on the stage-2 mask so that one bar carries one backend; Table 11 repeats every row with its final mask, both bootstrap intervals and the signed effect of refinement. The spread is the same story either way, 13.4 to 74.2 gIoU at stage 2 and 14.2 to 72.9 refined, a range far wider than any design choice in Table 13, and every backend is at or above SAM 3's 13.1 on this benchmark, though the weakest only matches it: GLM 4.6V scores 13.4 ($[10.2, 16.8]$) traced and 14.2 ($[11.0, 17.6]$) refined, so the claim that the floor of this set clears SAM 3 rests on the other twenty rows. The second dashed line is SAM 3 Agent at 72.4, driven by the same Gemini-3.7-Flash we use: with a segmentation specialist as its tool it is level with our best backend, GPT-6 Astra at 74.2 ($[70.4, 77.9]$) traced and 72.9 ($[69.2, 76.6]$) refined, a gap of 1.8 on the traced mask and 0.5 on the refined one, with the baseline score inside the corresponding GPT-6 Astra item-bootstrap intervals, while SAM 3 on its own is 59.8 behind. The two dashed lines also respond very differently to the MLLM. Swapping the agent's MLLM across Gemma-4-31B, Gemini-3.5-Flash and Gemini-3.7-Flash moves it by 0.7 gIoU (72.4 to 73.1), because its masks come from SAM 3 either way; swapping ours across the same Gemini generations moves it by 10.8. The backend, not the pipeline, is what our quality is made of, and it is the part of the stack that is still improving. Figure 24 shows what that spread looks like on single items: the same five ReasonSeg queries traced by seven of the backends, where the object being selected barely changes down a row and the outline drawn around it does.

General capability and segmentation quality. Figure 20(a) plots that quality against a general capability index built from public benchmark scores (§C.5), and (b) re-plots the same points against release date, joining models inside one product line. Two things follow. First, general capability is a *poor* predictor of segmentation quality: over the twenty-one backends the index correlates with final gIoU at only $\rho = 0.22$ (n=17, $p = 0.41$), and no axis reaches significance either — software engineering $\rho = 0.21$ ($p = 0.42$), reasoning and knowledge $\rho = 0.43$ ($p = 0.09$), multimodal VQA $\rho = 0.36$ (n=8), agentic $\rho = -0.05$ (n=14). The exceptions are instructive, and they run in both directions. The second and third highest capability indices in the set belong to Claude Opus 5 (+0.70, eight benchmarks, 96.0 on SWE-bench Verified) and GLM 5.3 Flash (+0.66), which segment at 46.6 and 49.6, fourteenth and twelfth of twenty-one; GPT-5.4 sits high on the index and low on the plot (27.1), and Muse Glimmer 30B likewise (23.5). All of them localize adequately and then place control points badly. The reverse holds as well: Gemma 4 31B and Gemma 4 26B carry two of the three lowest indices in the set (-0.95 and -1.29) and still segment seventh and eighth at 56.2 and 56.1. The practical reading is unchanged and rests on those exceptions rather than on the fit: no published headline score, vision or coding, is a reliable way to pick a backend for LLMSeg, and the only dependable test is to run the pipeline on the model.

Second, and more directly useful, every model line we can follow across generations improves over its span, with no pipeline change: Gemini Flash 58.6 → 66.2 → 69.4 → 66.7 over four releases in nine months, Gemini Flash-Lite 28.8 → 44.5, GPT 27.1 → 55.9, and Qwen 49.5 → 51.0. The trend is not monotone at the release level: Gemini 3.8 Flash scores 2.7 below Gemini 3.7 Flash, a reminder that a newer model is not automatically a better backend. The Flash-Lite line is the most economical evidence: a +15.7 gIoU gain bought by nobody retraining anything. Size within a generation buys little: Gemma 4 31B and its mixture-of-experts sibling Gemma 4 26B, released the same day, land 0.1 apart (56.2 and 56.1). The two Z.ai models are not one product line and the generation plot joins no arrow between them, but they bracket the same eight months as the Gemini Flash line: GLM 4.6V, of December 2025, is the weakest backend in the study at 14.2, and GLM 5.3 Flash, of August 2026, reaches 49.6. Price buys much less than generation does ($\rho = 0.25$ against input price, Figure 23); the second-best backend here costs 21× less than the best.

One caveat belongs with these numbers. Refinement is not uniformly positive across backends, and on the wider set it is negative on eleven of the twenty-one: it adds +3.7 gIoU on Gemini-3.7-Flash but *costs* 8.3 on GPT-5.4, 6.8 on Gemini-3.1-Flash-Lite, 2.4 on Claude Opus 5, 1.3 on Gemma 4 31B and 1.2 on GPT-6 Astra, our strongest backend. The loop can only re-trace what the backend proposes, so a backend that mislocates its focus boxes spends its passes moving the mask off the object. The refinement gains reported elsewhere in this paper are Gemini-3.7-Flash's, and the appropriate

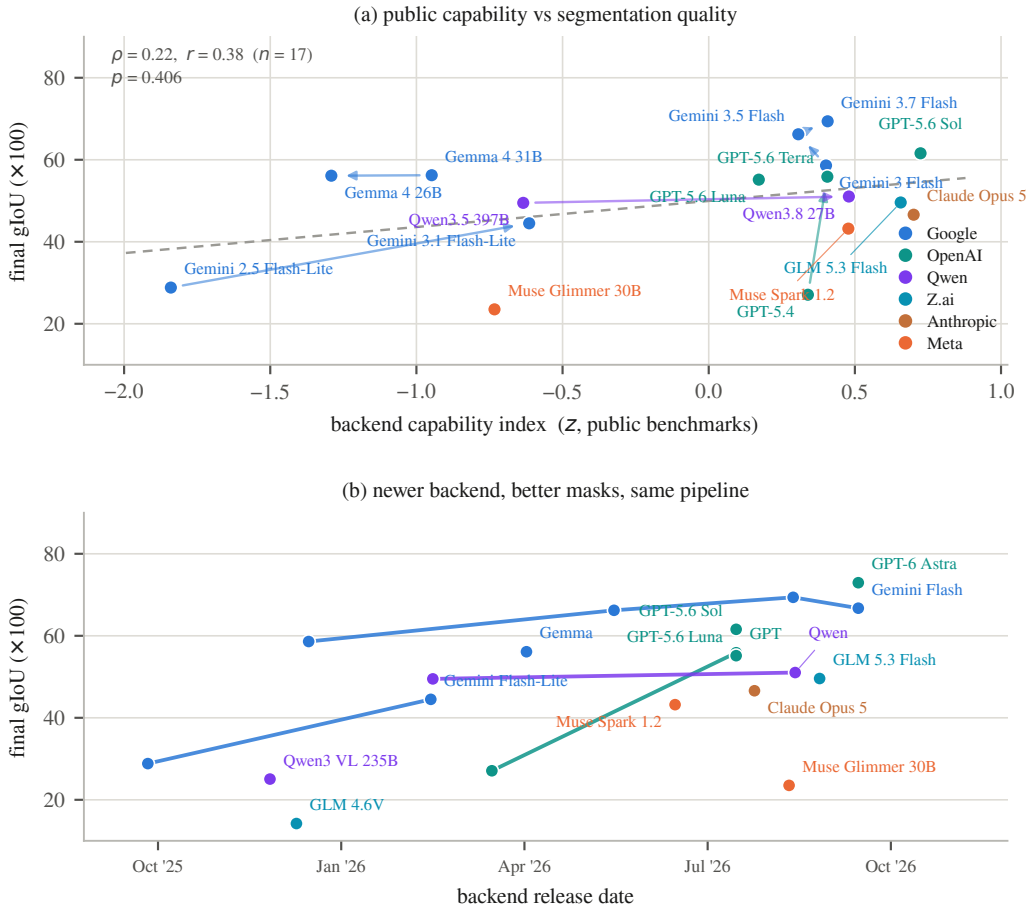


Figure 20: **Backend progress is LLMSeg progress. (a)** A general capability index over public benchmarks (§C.5) against LLMSeg final gIoU on ReasonSeg val, with arrows joining successive generations of one product line and an OLS fit dashed. **(b)** The same runs against release date. Nothing in the pipeline changes between the points of a line.

refinement budget must be checked for each backend. It is also why Figure 5 ranks the backends on what they trace: a bar pair would read as a refinement result on rows where refinement is not one.

E.3 Ablations and design alternatives

Table 13 and Table 14 test the design choices of Section 3 on two splits at once, in the two quantities a practitioner has to trade against each other: final-mask cIoU and MLLM calls per image, on ReasonSeg val and RefCOCO val, all 200 items each. Behind it are fifteen variants per split: eleven individual changes to the reference and four combinations of topology, cleanup, and box adjustment; the full tables, with gIoU, the run-to-run deviations and all component variants, are Table 13 and Table 14. Every ReasonSeg configuration was run three times with an identical setup and the cell is the mean over those runs; the sample deviation across them is what makes the rest of this paragraph readable, since re-running one fixed configuration moves its score by 0.1 to 1.3 gIoU, so that is the resolution of every comparison below. RefCOCO has a single run per configuration, carry no deviation, and are read for direction only. The topology-off row matches the conditioning used in the main results, although its pass budget is larger. The reference row has topology on, and each variant applies its named changes to that reference. Turning topology on reduces the observed scores by 2.1 cIoU on ReasonSeg and 2.5 on RefCOCO at similar call counts, with the same sign on both splits and on gIoU too (1.5 and 2.6), so we leave it off by default and keep it as an option for the item type it exists for (Table 15). The largest component effect is refinement; the remaining changes generally

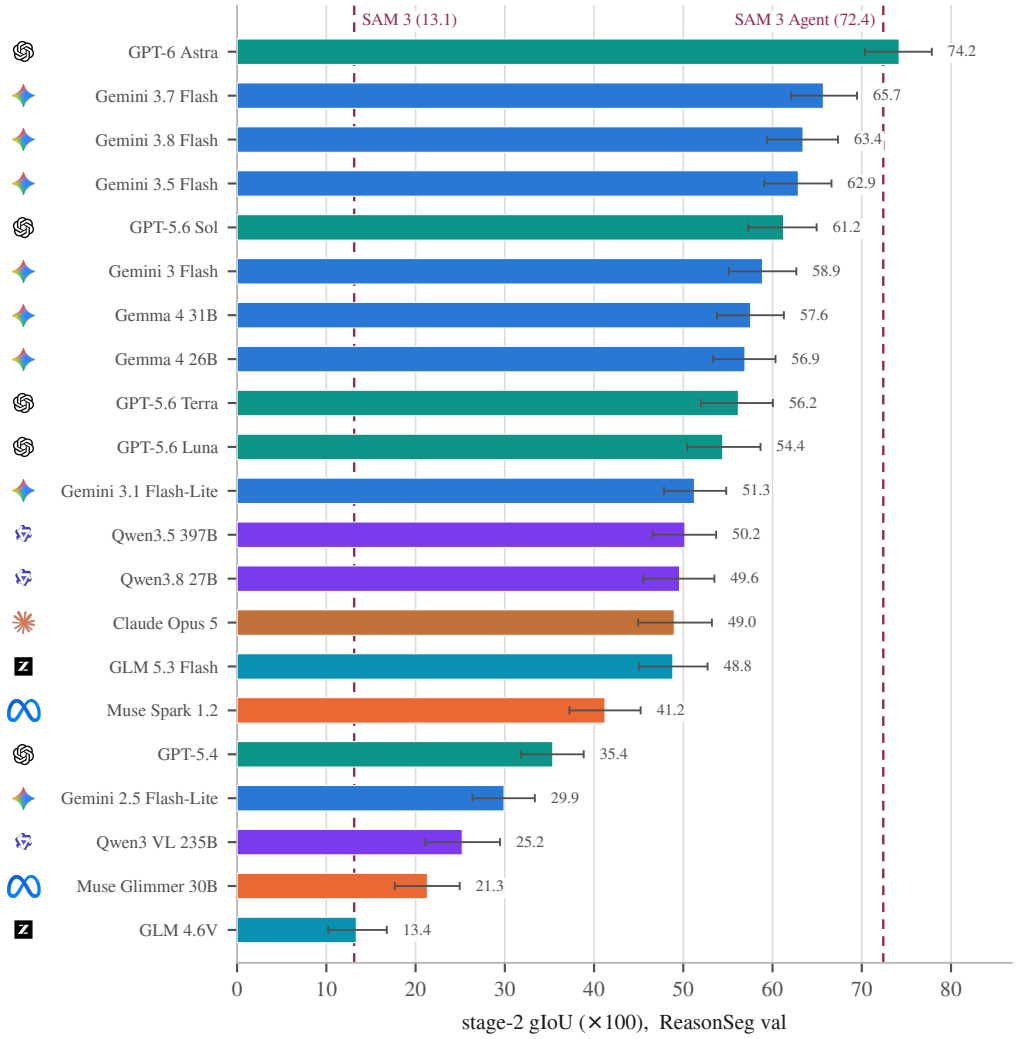


Figure 21: **Complete backend comparison.** All 21 backends behind Figure 5, including Gemini 3.8 Flash, Gemma 4 26B, and Qwen3.8 27B omitted from the selected main chart. Stage-2 gIoU, 95% item-bootstrap intervals, and baselines use the same 200 ReasonSeg val items.

produce smaller differences. Refinement is the exception: removing it costs 6.9 gIoU and 7.5 cIoU here and 12.4 and 15.2 on RefCOCO, a large difference on both splits, although this comparison also changes topology, and it is also where the cost is, 12.6 of the 14.6 calls per image. Outside it the widest gap is low image detail, 3.5 gIoU (3.2 cIoU) here and 4.5 (4.4) on RefCOCO, and most variants sit within two points of the reported configuration on both splits. No alternative is ahead of the topology-off row on both metrics of both splits, and the cost column separates them no better than the quality columns do, which is itself a result: the pipeline is not delicately balanced on its prompt conditioning or on the particular way its calls are batched, and the measured costs help assess these alternatives.

Prompt conditioning. Cutting the stage-1 morphology or topology description out of the prompts does not measurably change the final mask. Measured against the suite's reference run, which carries both, dropping morphology gives 69.9 gIoU and dropping topology 70.7 against its 69.2 ± 1.3, small differences relative to the spread across configurations, and both above it on RefCOCO as well, by 2.4 and 2.6 gIoU (Table 14). Topology conditioning is therefore a design detail rather than a component the results rest on, which is the whole of the case for leaving it off: it is not that describing holes hurts, but that it never measurably helps at the split level, so the configuration we report is the one

Backend	Developer	Released	Cap. index	# bm	stage-2 gIoU	final gIoU	Δ
GPT-6 Astra†	OpenAI	2026-09	–	0	**74.2** [70.4, 77.9]	**72.9** [69.2, 76.6]	-1.2
Gemini 3.7 Flash	Google	2026-08-13	+0.41	7	65.7 [62.1, 69.5]	69.4 [65.9, 72.8]	+3.7
Gemini 3.8 Flash†	Google	2026-09	–	0	63.4 [59.4, 67.3]	66.7 [63.0, 70.4]	+3.3
Gemini 3.5 Flash	Google	2026-05	+0.31	9	62.9 [59.1, 66.6]	66.2 [62.4, 70.0]	+3.3
GPT-5.6 Sol†	OpenAI	2026-07	+0.72	7	61.2 [57.3, 64.9]	61.6 [57.7, 65.5]	+0.3
Gemini 3 Flash	Google	2025-12	+0.40	4	58.9 [55.1, 62.7]	58.6 [55.1, 61.9]	-0.3
Gemma 4 31B†	Google	2026-04-02	-0.95	6	57.6 [53.8, 61.3]	56.2 [52.7, 59.9]	-1.3
Gemma 4 26B†	Google	2026-04-02	-1.29	5	56.9 [53.3, 60.3]	56.1 [52.3, 59.5]	-0.8
GPT-5.6 Terra	OpenAI	2026-07	+0.41	7	56.2 [52.0, 60.0]	55.9 [51.6, 60.1]	-0.3
GPT-5.6 Luna	OpenAI	2026-07	+0.17	7	54.4 [50.5, 58.7]	55.1 [51.0, 59.0]	+0.7
Gemini 3.1 Flash-Lite	Google	2026-02	-0.61	5	51.3 [47.8, 54.8]	44.5 [41.2, 47.8]	-6.8
Qwen3.5 397B†	Qwen	2026-02-16	-0.63	7	50.2 [46.6, 53.7]	49.5 [45.9, 53.1]	-0.7
Qwen3.8 27B†	Qwen	2026-08-14	+0.48	10	49.6 [45.5, 53.5]	51.0 [47.1, 54.7]	+1.4
Claude Opus 5†	Anthropic	2026-07-24	+0.70	8	49.0 [44.9, 53.2]	46.6 [42.6, 50.5]	-2.4
GLM 5.3 Flash†	Z.ai	2026-08-26	+0.66	4	48.8 [45.1, 52.7]	49.6 [45.7, 53.3]	+0.7
Muse Spark 1.2†	Meta	2026-06	+0.48	3	41.2 [37.2, 45.2]	43.2 [39.1, 46.9]	+2.0
GPT-5.4	OpenAI	2026-03	+0.34	4	35.4 [31.8, 38.8]	27.1 [23.8, 30.8]	-8.3
Gemini 2.5 Flash-Lite	Google	2025-09-26	-1.84	4	29.9 [26.4, 33.4]	28.8 [25.6, 32.1]	-1.1
Qwen3 VL 235B	Qwen	2025-11-26	–	0	25.2 [21.1, 29.5]	25.1 [20.8, 29.4]	-0.2
Muse Glimmer 30B†	Meta	2026-08-11	-0.73	7	21.3 [17.7, 24.9]	23.5 [20.0, 27.3]	+2.2
GLM 4.6V†	Z.ai	2025-12-09	–	0	13.4 [10.2, 16.8]	14.2 [11.0, 17.6]	+0.9

Table 11: **The twenty-one backends.** LLMSeg on ReasonSeg val, all 200 items, identical pipeline and identical settings; only the MLLM differs. Both masks are given, with 95% bootstrap intervals and the signed effect Δ of refinement; Figure 5 plots the stage-2 column. "Cap. index" is the general capability index, "# bm" the number of published benchmarks behind it, † a longer per-call budget.

without the extra prompt text. What it buys is structural, not aggregate: it is what the hole carving in Stage 2 and the gated cleanup consume, so without it the model is never told how many closed curves to fit and a holed or multi-part object is cleaned as if it were one solid blob. The targets it can act on are between 0% and 7% of these benchmarks, and on the items scored both ways the paired difference is -0.3 gIoU overall and $+0.1$ on the holed ones, so the option is retained for explicit structural control and is not claimed as a source of aggregate accuracy (Section E.4). Reducing the backend's reasoning effort or the image detail sent with each call leaves quality flat to slightly below (72.3 and 71.5 cIoU here against 74.7, 74.3 and 75.4 on RefCOCO against 79.8), and low reasoning effort is the one setting that is genuinely cheaper: at the same call count, the time an image spends in backend calls falls a quarter, from 177 s to 133 s. The deterministic cleanup is likewise not what carries the result (73.8 cIoU without it, and 76.8 on RefCOCO), which matters because it is the only step in LLMSeg whose geometry the model did not choose.

Is the detection stage needed? Asking the model for splines directly, with no detect stage and therefore no box to confine control points to, tests whether stage 1 earns its call. Without a box the model must place absolute coordinates over the whole image, and there is no invariant preventing a loop from wandering across the scene. On ReasonSeg this costs little that the run-to-run band resolves (69.5 gIoU and 72.1 cIoU against the topology-on reference's 69.2 and 72.6) and saves calls (8.8 vs. 14.3). RefCOCO is also competitive (78.5 gIoU at 10.4 calls against 76.7 at 16.8). Stage-2 cIoU does favor detection on both splits (Table 4). The existing disconnected-target subsets contain $63/200$ ReasonSeg and $18/200$ RefCOCO items, defined by at least two GT components above the annotation-speck threshold, not by instance labels. On these subsets, detection improves initial cIoU by 11.3 and 2.2 points, but its final differences are -4.0 and $+0.8$ (Table 12). Thus detection provides box confinement (Eq. (2)), instance identities, and structural conditioning, not a demonstrated general accuracy advantage on multiple objects.

Why one batched call rather than per-instance crops? An obvious alternative runs the segment stage once per detected instance on a crop of its box, which gives the model more effective resolution per object at a cost that grows linearly with instance count, against the fixed single call of Section 3. It reaches 75.4 cIoU (71.8 gIoU) at 14.2 calls per image, the best ReasonSeg cIoU in the table but only 0.7 above the topology-off configuration and at the same cost, again because most ReasonSeg items

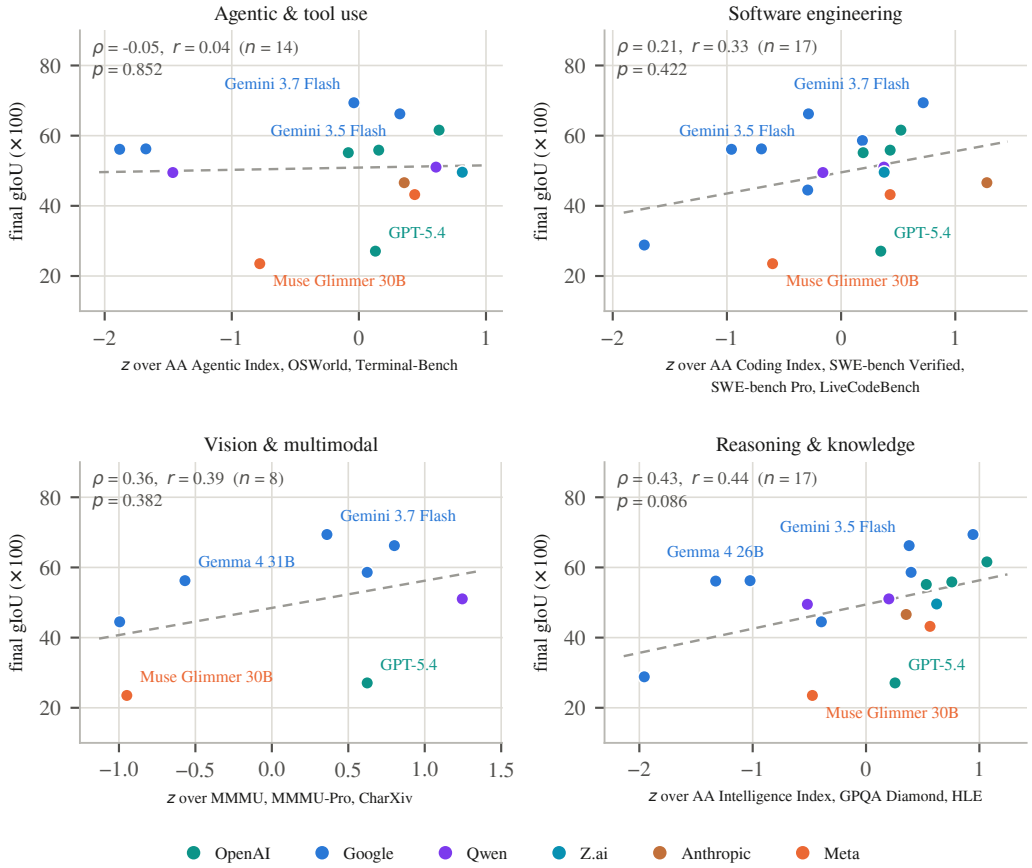


Figure 22: **Which kind of capability predicts LLMSeg quality?** Each panel scores the same twenty-one backends against one capability axis, over the models that publish a number for it; the four points furthest from each panel's fit are named, since the on-trend ones are identified in Figure 20 and Table 11. No axis predicts segmentation quality on this set.

have one instance and one crop is then one call; on RefCOCO it is a point and a half *below* (78.3 at 15.4 calls), so the sign does not survive the second split.

Why one batched refine rather than per-instance crops? The same crop alternative applied to stage 3: keep the batched segment call, then run the refinement loop once per instance on a crop of its box, each with its own overlay. Cost again grows with instance count; the batched refine of Section 3 stays two calls per pass regardless of N. This is the most expensive variant in the table, 21.3 calls per image against 14.6, a 46% increase, for 74.9 cIoU against 74.7, and on RefCOCO it costs two points (77.7 against 79.8). Its cost would grow further on multi-instance benchmarks, where the batched refine's cost does not.

Why B-splines rather than polygons? Asking for a dense polygon instead of spline control points is the natural baseline for our output primitive: it removes the cubic smoothness prior and allows more coordinates per object. It can also fit jagged, irregular objects. At similar call cost the final masks have comparable aggregate scores (69.8 gIoU and 73.1 cIoU, and 78.7 and 79.1 on RefCOCO). The polygon variant uses degree-one loops in both stage 2 and stage 3. Its prompted vertex ranges are 32 to 160 initially and 6 to 24 in refinement, versus 16 to 64 and 4 to 12 for the reference, so neither coordinate nor token budgets are matched. The complementary stage-2 comparison is read before refinement: there the polygon's gIoU edge is bought entirely on small objects, and on the largest quartile of items by ground-truth area the reference leads on both metrics, 75.6 against 73.4 gIoU and 76.0 against 69.7 cIoU, cIoU being the pixel-mass metric and therefore the one that punishes a boundary drawn with too few vertices; the smooth-boundary advantage of Figure 26 is what the

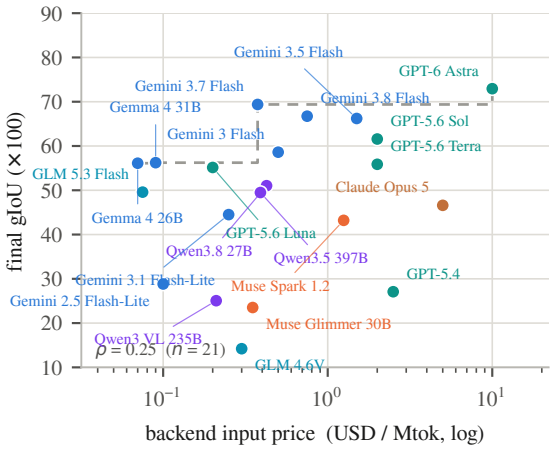


Figure 23: **Quality is not bought.** Backend input price against LLMSeg final gIoU on ReasonSeg val; the dashed staircase is the cheapest-so-far frontier. Price does not order quality.

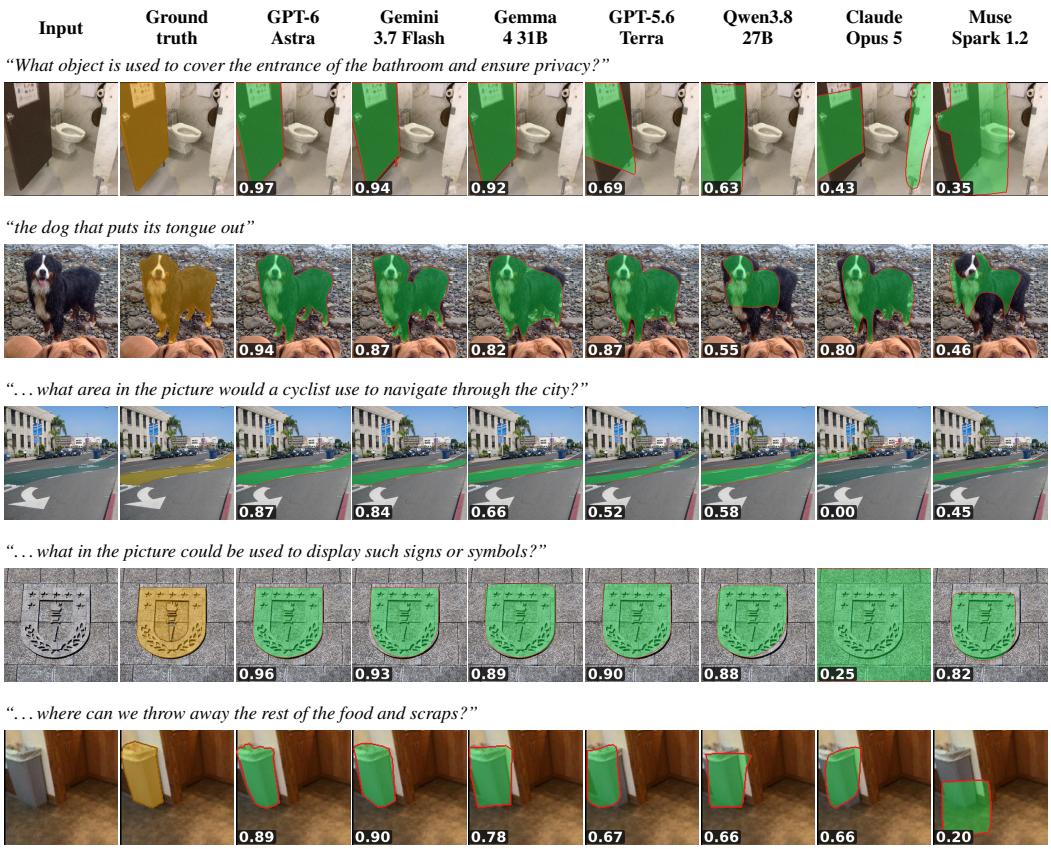


Figure 24: **What the backend gap looks like on one item.** Five ReasonSeg val items, the same pipeline throughout, one column per MLLM backend, ordered as in Figure 5. Every panel is the *stage-2* mask, so the comparison is of what each backend traces rather than of how it refines. Grounding is largely shared; the outline is what falls apart down a row.

spline buys on top of that. Figure 25 shows what that looks like on three items. The ablation is also self-limiting, because the backend chooses the degree itself (Section 3) and chooses 1, which degenerates to the polygon, on 125 of the 195 objects it traced here, so on most of the split it changes the vertex budget and the placement rules rather than the primitive. On the cubic-traced items, the

	ReasonSeg val (n=63)				RefCOCO val (n=18)			
	initial		final		initial		final	
Variant	gIoU	cIoU	gIoU	cIoU	gIoU	cIoU	gIoU	cIoU
Reference	52.4	63.9	57.1	66.1	65.5	63.9	73.4	71.2
w/o detect stage	54.0	52.6	61.1	70.1	63.5	61.7	72.2	70.4
polygon loops	54.9	62.9	60.4	64.9	67.7	66.5	75.1	75.1

Table 12: **Disconnected-target subsets.** Single-run initial/final gIoU and cIoU on the same items in each split. GT masks have at least two components, ignoring components below 0.5% of GT area. Components are not annotated instances: touching objects merge and occluded objects can split.

reference emits 28.9 coordinates per image versus 48.5 for the polygon variant, a 1.7× difference. Over the whole split at stage 2, the reference scores 64.5 gIoU and 69.5 cIoU versus 65.9 and 66.7 for polygons; polygons lead on both metrics in the smaller two area strata. Across three runs, stage-2 gIoU/cIoU are 65.1/68.3 for the reference and 65.3/66.2 for polygons; final scores are 69.2/72.6 and 69.8/73.1. These results support a compact parametric representation, not a uniform or equal-budget segmentation advantage.

A different split, the same backend. RefCOCO val reproduces the main table's one large effect and its many small ones. Removing refinement costs 9.8 gIoU and 12.7 cIoU, nearly twice the ReasonSeg loss, which is what a split of tight, well-delineated everyday objects should do to a method whose refinement loop exists to tighten boundaries. Ranking the sixteen configurations by final gIoU, the two splits agree at $\rho = 0.69$ ($p = 0.003$), so a variant that helps on one is likely to help on the other. The two design alternatives that cost nothing on ReasonSeg cost nothing here either: dropping the detect stage scores 78.5 at 10.4 calls per image against 76.7 at 16.8, and polygon loops score 78.7 against 76.7. That is the same caveat stated in the main text, not a second piece of evidence: a referring expression names one object by construction, so RefCOCO gives the multi-instance machinery no more room than ReasonSeg does.

Disconnected targets. Table 12 separates initial tracing from final refinement on targets whose ground-truth masks contain multiple connected components. Detection improves initial cIoU by 11.3 points on ReasonSeg and 2.2 on RefCOCO; its final differences are −4.0 and +0.8, respectively. The subset therefore supports an initial tracing benefit, not a consistent final-mask advantage or a claim about annotated instances.

E.4 Where the ground truth has holes

The topology ablation of Table 13 removes the hole and component description from all three prompts and moves nothing measurable (70.7 vs. 69.2 gIoU, inside the run-to-run band). That average is taken over a split where almost no target has a hole, so it cannot distinguish "topology does not help" from "the items it could help on are too few to show up". We therefore measure each item's ground truth directly and score the two subsets apart. A hole is an enclosed background component that never touches the image border and covers at least 0.5% of the object's area; anything smaller is annotation speckle. Items with an empty ground truth are excluded, since a hole needs an object, which is also why the LVIS columns here are gIoU over positives rather than the cgF1 of Table 3.

The first finding is the prevalence itself. Over the eleven splits measured, holed targets are 7.1% of ReasonSeg val and 6.6% of ReasonSeg test, 4.7% of LVIS positives, and between 0% and 1.4% of every RefCOCO/RefCOCO+/RefCOCOg split: COCO's polygon annotations essentially never carve one. The subset the topology machinery exists for is a small minority of what these benchmarks contain, which is by itself enough to explain a null split-level ablation.

The second finding is that holed targets are harder for *every* system, and not especially for ours (Table 15, Figure 27). On the referring benchmarks LLMSeg drops 69.9 → 55.5 gIoU ($\Delta = -14.3$, 95% CI $[-24.9, -4.8]$), but SAM 3 drops 31.0 → 11.7 (−19.3) and SAM 3 Agent with the same Gemini-3.7-Flash backend drops 75.9 → 59.3 (−16.5) on the identical items. A holed target is simply a harder target; the explicit topology neither rescues us nor costs us relative to a mask decoder that has none. The one system that barely notices is SAM 3 Agent driven by Gemma-4-31B (−7.1

Variant	gIoU	cIoU	calls/image	s/image
LLMSeg (full, reference run)	69.2 ±1.34	72.6 ±0.23	14.3 ±0.59	175 ±17
w/o refinement (T=0)	63.8 ±0.68	67.2 ±2.72	2.0 ±0.00	24 ±2
w/o morphology conditioning	69.9 ±0.47	73.4 ±2.69	14.9 ±0.32	179 ±7
w/o topology conditioning	70.7 ±0.74	74.7 ±1.62	14.6 ±0.32	177 ±9
low backend reasoning effort	68.5 ±0.30	72.3 ±1.59	14.4 ±0.58	133 ±6
low image detail	67.2 ±0.88	71.5 ±4.60	14.6 ±0.22	180 ±6
w/o deterministic postprocessing	69.6 ±0.55	73.8 ±1.89	14.8 ±0.36	172 ±29
w/o detect stage (splines over whole image)	69.5 ±0.59	72.1 ±0.87	8.8 ±0.26	120 ±19
per-instance cropped calls	71.8 ±0.44	75.4 ±0.44	14.2 ±0.24	169 ±3
polygon loops instead of B-splines	69.8 ±0.57	73.1 ±3.16	14.7 ±0.09	186 ±7
per-instance cropped refine	71.7 ±0.23	74.9 ±0.91	21.3 ±0.49	185 ±3
w/o topology + w/o postprocessing	70.2 ±0.27	74.1 ±2.20	14.8 ±0.37	162 ±12
w/o topology + box-adjust pre-pass	70.0 ±0.27	70.2 ±0.38	15.7 ±0.34	188 ±17
w/o postprocessing + box-adjust pre-pass	69.4 ±0.69	73.6 ±3.55	15.5 ±0.32	166 ±12
w/o topology + w/o postproc. + box-adjust	69.5 ±0.11	73.2 ±1.64	15.9 ±0.10	169 ±12

Table 13: **Ablations on ReasonSeg val** (all 200 items, Gemini-3.7-Flash backend), final mask, the full version of Table 4 with gIoU and all omitted component variants. The first block ablates components of LLMSeg, the second replaces a design choice with its natural alternative, and the third switches two or three at once. The reference run here has topology conditioning on; the configuration the paper reports is the *w/o topology conditioning* row. Cells are the mean over three runs of each configuration with their sample deviation; a difference smaller than its own row's deviation is not a finding. *s/image* is the wall clock an image spends inside backend calls, summed over its three stages: it is one image's latency with many in flight at once, not a throughput figure, and it carries the provider's day-to-day variation, which is why it is the widest column here.

Variant	gIoU	cIoU	calls/image
LLMSeg (full, reference run)	76.7	77.3	16.8
w/o refinement (T=0)	66.9	64.6	2.0
w/o morphology conditioning	79.1	78.8	17.0
w/o topology conditioning	79.3	79.8	17.5
low backend reasoning effort	75.7	74.3	16.6
low image detail	74.8	75.4	15.8
w/o deterministic postprocessing	76.4	76.8	16.8
w/o detect stage (splines over whole image)	78.5	78.8	10.4
per-instance cropped calls	78.8	78.3	15.4
polygon loops instead of B-splines	78.7	79.1	16.6
per-instance cropped refine	77.4	77.7	18.8
w/o topology + w/o postprocessing	76.6	77.9	17.1
w/o topology + box-adjust pre-pass	79.2	79.7	18.4
w/o postprocessing + box-adjust pre-pass	76.9	77.7	18.4
w/o topology + w/o postproc. + box-adjust	77.9	78.3	19.1

Table 14: **Ablations on RefCOCO val** (all 200 items, Gemini-3.7-Flash backend), final mask, one run per configuration. Blocks and rows match Table 13, so a row can be read across the two tables.

and -1.8 on the two item sets), which is a property of which masks SAM 3 proposes, not of any topology reasoning.

The third finding closes the ablation. On the 396 ReasonSeg items that both the topology-on and the topology-off run scored, the mean per-item IoU difference is -0.3 ($[-1.8, 1.2]$) overall and $+0.1$ ($[-5.3, 5.3]$) on the 19 holed ones: targeting the subset does not reveal a hidden effect. This is the measurement behind the position taken in Section 5: topology conditioning is an option that makes explicit hole-loop specification possible where a target has holes, not a source of the accuracy we report, and we do not claim it as one. Two caveats bound how far the number can be pushed: the two runs differ in iteration budget as well (T=12 vs. T=8), and 19 holed items is a small sample, which the interval reflects.

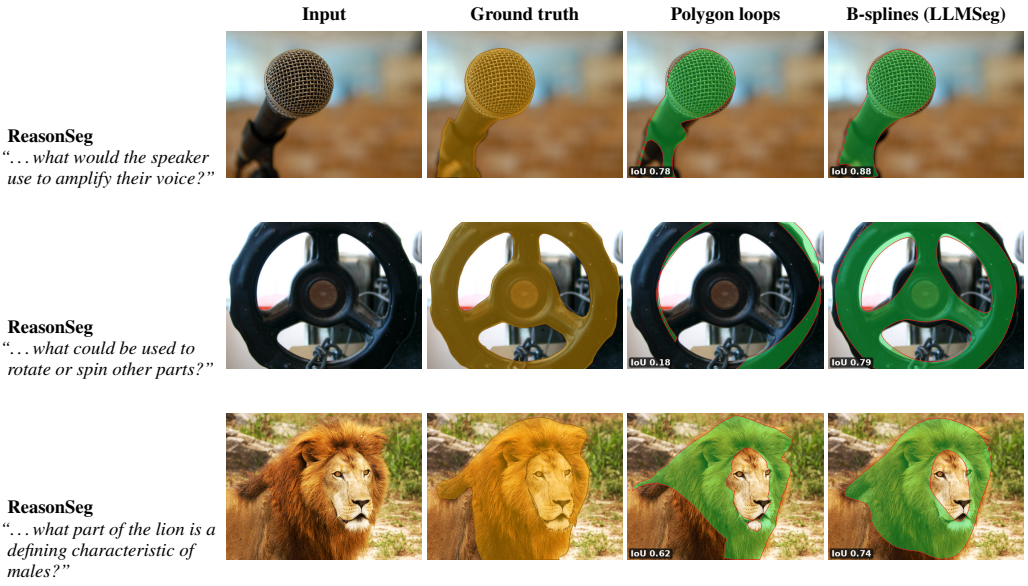


Figure 25: **The output primitive, on curved boundaries.** Stage-2 traces from the polygon variant of Table 13 and from the reference run on identical items, same backend; each chip is that run's recorded stage-2 IoU. Rows are items the reference traced with a *cubic* loop, since the backend picks one degree per call and picks 1, which is the polygon, on most of the split. These are chosen examples of the mechanism, not an aggregate result.



Figure 26: **Explicitly carved holes.** The query is *"the sewage outlet"*. The cyan-box region is enlarged. SAM 3 fills the grate solidly, whereas LLMSeg carves one hole per slot.

Because the two runs sampled different items on RefCOCO and LVIS, each configuration defines its own item set, and every column of a block is scored on that block's items or omitted; a baseline covering less than 95% of them is left blank rather than averaged over a different set. Columns within a block are comparable; the two blocks are not comparable to each other. Everything here is recomputed from the finished runs, with no model re-run.

E.5 Localization or boundary?

Table 3 reports the gap to SAM 3 as two scalars per benchmark, and both are already averaged over the ten COCO IoU thresholds 0.50:0.05:0.95. That average is what hides the shape of the effect: a system that finds every object and traces it loosely, and one that finds fewer objects and traces them tightly, can report the same mean AP. The two are told apart by *where along the threshold axis* the AP goes, so Figure 28 un-averages it for the main-results pipeline configuration (Gemini-3.7-Flash, T=8, topology conditioning off), on the same matched subsets and through the same COCOeval pass.

The separation is clean. At the loosest threshold the two systems are level on both axes: 79.3 vs. 79.6 box AP_{50} on LVIS, and on COCO LLMSeg is *ahead* on box (87.4 vs. 79.5) and level on mask (80.0 vs. 81.3). The gap widens at tighter thresholds. By IoU 0.90 our mask AP has fallen to 16.4 on both benchmarks while SAM 3 holds 42.5 and 24.3, and averaged over the ten thresholds the mask gap is 3 to 6× the box gap, whose interval on COCO covers zero. A mask that is drawn as a closed spline and filled is the right object in the right place to within a box, and a few pixels loose along its boundary, which is exactly the error a threshold sweep makes visible and a mean AP does not. The

Method	Referring / reasoning			LVIS (positives)		
	no holes	holes	Δ	no holes	holes	Δ
Items of the topology-on run (LLMSeg at T=12); 1970+26 / 106+4 items (hole-free + holed)						
LLMSeg	69.9	55.5	-14.3	60.6	54.3	-6.3
SAM 3	31.0	11.7	-19.3	68.3	62.2	-6.2
SAM 3 Agent (Gemini-3.7-Flash)	75.9	59.3	-16.5	–	–	–
SAM 3 Agent (Gemma 4 31B)	74.6	67.5	-7.1	72.9	60.2	-12.7
Items of the main-configuration run (LLMSeg at T=8, topology off); 4884+89 / 766+38 items (hole-free + holed)						
LLMSeg, no topology	71.1	59.7	-11.3	61.3	58.0	-3.2
SAM 3	31.6	16.4	-15.2	65.7	69.4	+3.7
SAM 3 Agent (Gemini-3.7-Flash)	74.6	68.7	-5.9	–	–	–
SAM 3 Agent (Gemma 4 31B)	73.4	71.6	-1.8	73.1	69.3	-3.7

Table 15: **IoU on targets whose ground truth has a hole, against targets that do not.** gIoU over the final mask, pipeline failures included and scored 0; a hole is an enclosed background region covering $\geq 0.5\%$ of the object. Δ is holed minus hole-free within one column. The two row blocks use different item sets and are not comparable to each other.

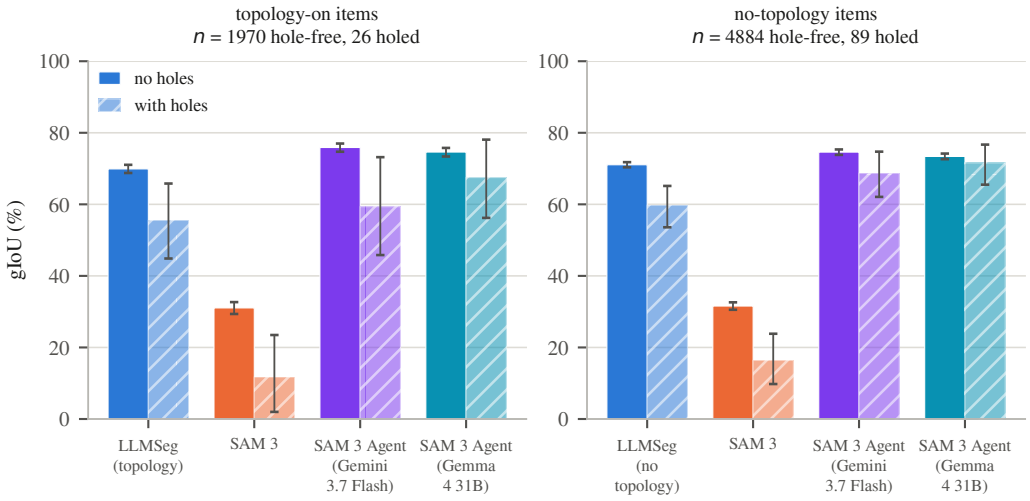


Figure 27: **Holed targets are harder for every system.** gIoU on the hole-free and holed subsets of the ten referring/reasoning splits, pooled over items; whiskers are 95% percentile bootstraps. Each panel is one item set, and every bar in a panel is scored on the same items. The holed subsets are a small minority of these benchmarks.

refinement loop attacks that error and does not finish it: its +10.4 (LVIS) and +12.5 (COCO) mask AP over the un-refined stage-2 mask is spread over every threshold up to 0.85, peaking in the 0.65 to 0.85 band at +14 to +20, and is down to +2.4 and +1.5 by IoU 0.95. Refinement moves the outline substantially closer to the object; what it does not do is make it pixel-tight, and the remaining gap is concentrated at high overlap thresholds. These are subset results and do not establish full-dataset AP parity.

E.6 What a mask costs

Where the time goes. Refinement dominates: stage 3 is 76 to 94% of the wall clock on every backend (89% on Gemini-3.7-Flash), which is what a loop of up to T=12 passes at two calls each should cost against a single detect and a single segment call. The cost is in the number of calls and not in their content: with 15 calls at a published 2.3 s time-to-first-token, Gemini-3.7-Flash spends about 34 of its 153 median seconds waiting for first tokens alone. Call count is also not a proxy for effort spent well — the best backend uses 11.7 calls per image, while the three that spend the most (24.0 to 24.4)

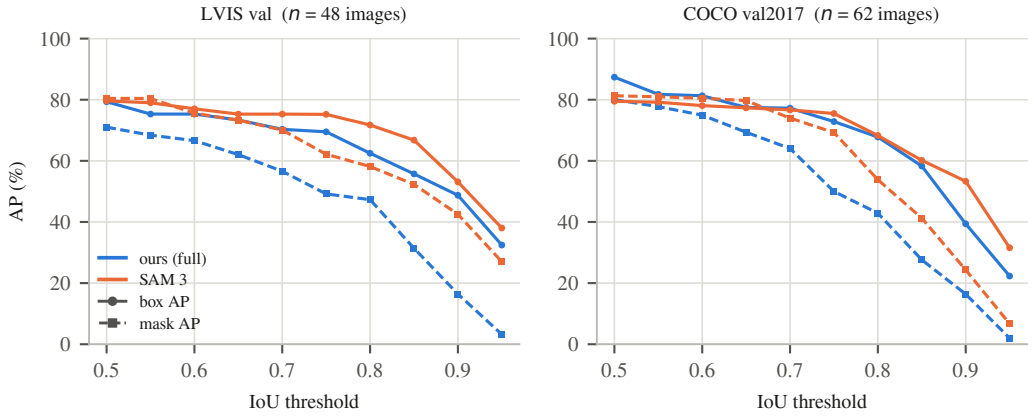


Figure 28: **We localize like SAM 3 and lose the boundary.** COCO-style AP at each IoU threshold, over the AP-eligible images of the matched LVIS and COCO subsets, with boxes taken from the predicted masks. LLMSeg is the main-results configuration. Solid lines are box AP and dashed lines mask AP: the box curves run together while the mask curves fan apart.

Method	published backend specs params	$/Mtok in / out	tok/s	measured / estimated cost per image calls MLLM / SAM 3	median s	est. $ / 1k img	final gIoU
LLMSeg (GPT-6 Astra)	–	10.00 / 50.00	–	11.7 / 0	218	323.9	72.9
LLMSeg (Gemini 3.7 Flash)	–	0.38 / 1.88	98	15.0 / 0	153	15.6	69.4
LLMSeg (Gemini 3.8 Flash)	–	0.75 / 3.75	–	16.2 / 0	419	33.7	66.7
LLMSeg (Gemini 3.5 Flash)	–	1.50 / 9.00	116	17.6 / 0	246	77.9	66.2
LLMSeg (GPT-5.6 Sol)	–	2.00 / 10.00	44	15.0 / 0	665	83.0	61.6
LLMSeg (Gemini 3 Flash)	–	0.50 / 3.00	66	24.4 / 0	588	36.0	58.6
LLMSeg (Gemma 4 31B)	31 B	0.09 / 0.34	35	21.8 / 0	1883	5.0	56.2
LLMSeg (Gemma 4 26B)	26 B	0.07 / 0.34	36	19.4 / 0	2151	3.7	56.1
LLMSeg (GPT-5.6 Terra)	–	2.00 / 12.00	42	10.6 / 0	264	62.3	55.9
LLMSeg (GPT-5.6 Luna)	–	0.20 / 1.20	61	14.6 / 0	368	8.6	55.1
LLMSeg (Qwen3.8 27B)	28 B	0.42 / 2.55	34	12.3 / 0	847	15.4	51.0
LLMSeg (GLM 5.3 Flash)	–	0.07 / 0.25	38	7.3 / 0	1100	1.4	49.6
LLMSeg (Qwen3.5 397B)	403/17 B	0.39 / 2.34	52	12.4 / 0	2920	14.3	49.5
LLMSeg (Claude Opus 5)	–	5.00 / 25.00	60	23.1 / 0	706	321.2	46.6
LLMSeg (Gemini 3.1 Flash-Lite)	–	0.25 / 1.50	71	23.8 / 0	100	17.6	44.5
LLMSeg (Muse Spark 1.2)	–	1.25 / 4.25	89	24.4 / 0	837	76.2	43.2
LLMSeg (Gemini 2.5 Flash-Lite)	–	0.10 / 0.40	102	21.4 / 0	113	5.6	28.8
LLMSeg (GPT-5.4)	–	2.50 / 15.00	42	24.0 / 0	211	176.8	27.1
LLMSeg (Qwen3 VL 235B)	236/22 B	0.21 / 1.90	19	9.0 / 0	128	6.6	25.1
LLMSeg (Muse Glimmer 30B)	30 B	0.35 / 1.50	56	14.7 / 0	1478	13.6	23.5
LLMSeg (GLM 4.6V)	108 B	0.30 / 0.90	12	15.7 / 0	1046	11.4	14.2
SAM 3 Agent (Gemma 4 31B)	31 B	0.09 / 0.34	35	2.3 / 1.4	27	6.4	73.1
SAM 3 Agent (Gemini 3.5 Flash)	–	1.50 / 9.00	116	2.5 / 1.5	13	17.5	72.5
SAM 3 Agent (Gemini 3.7 Flash)	–	0.38 / 1.88	98	2.4 / 1.4	11	8.3	72.4
SAM 3	860 M	–	–	0 / 1	2	4.3	13.1

Table 16: **What a mask costs.** ReasonSeg val, 200 items, the runs of Table 11. Left block: published backend specifications. Right block: cost per image, measured except for the spend column, which is an estimate under one token budget shared by every row and a fixed serving price per SAM 3 forward pass (Section C.6). LLMSeg rows are shaded.

score 58.6, 43.2 and 27.1, because convergence stops a pass sequence that is still moving the mask and lets an unproductive one run to the budget.

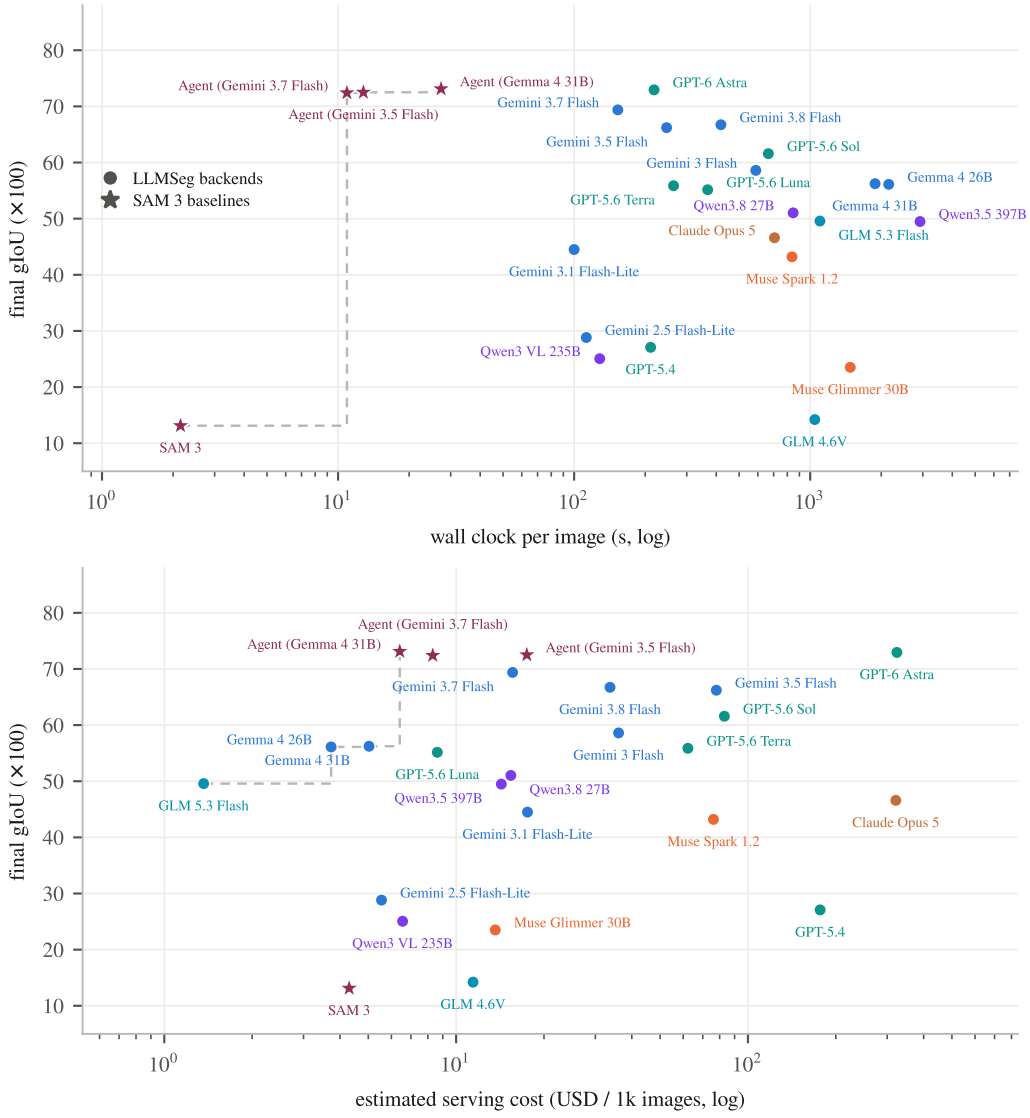


Figure 29: **Quality against what it costs.** The rows of Table 16, by wall clock (left) and by estimated serving cost (right); both axes are logarithmic and the dashed staircase is the cheapest-so-far frontier. Circles are LLMSeg backends, stars the SAM 3 and SAM 3 Agent baselines. Neither axis orders quality tightly.

LLMSeg is the expensive way to reach this quality. On this benchmark SAM 3 Agent is both better and cheaper: driven by the same Gemini-3.7-Flash it reaches 72.4 gIoU against our 69.4 at roughly $1/14$ the wall clock (11 s against 153 s) and just over half the estimated serving cost ($8.31 against $15.61 per thousand images), because it makes 2.4 agent rounds and 1.4 SAM 3 passes where we make 15 vision calls, and because the mask itself comes from a 860 M-parameter forward pass rather than from coordinates the MLLM has to write out. Charging the SAM 3 passes narrows that gap rather than closing it: the agent's 1.4 passes cost it $5.85 of its $8.31, and SAM 3 on its own costs $4.30, which only two LLMSeg backends undercut. That advantage is specific to what SAM 3 can already segment: SAM 3 Agent inherits SAM 3's mask, so where SAM 3 collapses (Section 4.2) the agent collapses with it, while LLMSeg is bounded only by its detect box. The honest summary is that predicting a mask as text buys open-vocabulary reach and pays for it in calls, latency and dollars, and that the price is set by the backend rather than by the pipeline: across the twenty-one backends the spend per thousand images spans 1.4 to 324, a factor of 238, over a gIoU range of 14.2 to 72.9 that it

orders only weakly. The most expensive backend is indeed the best, but it costs $21\times$ the second-best for 3.5 gIoU, and the two cheapest backends outscore nine dearer ones.

E.7 WHERE THE LLM ROUTE WINS AND LOSES

The results support a consistent picture. LLMSeg wins where (i) the query needs *reasoning* (ReasonSeg, Figure 3); (ii) the domain is *rare* (RF-100VL, $2.2\times$ SAM 3's AP_0), including medical and industrial imagery; (iii) *presence* judgment matters (hard-negative cgF1 on LVIS and COCO), where the LLM simply reasons about whether the concept is in the image; and (iv) *boundary form* matters: regular shapes admit compact cubic or straight-edged traces, and, with topology enabled, holes can be specified by interior loops. Neither final boundary smoothness nor hole count is guaranteed, and holed targets are a small minority of these benchmarks (both in Figure 26, quantified in Section E.4). It loses on pixel-tight mask AP and on thin, amorphous or highly irregular regions (Figure 14), where compact geometric traces lose fine boundary detail. Practical trade-offs: LLMSeg costs minutes and multiple API calls per image vs. 30 ms for SAM 3, but runs on the heavily-optimized LLM serving stack, needs no GPU of its own, and produces inspectable intermediate records (boxes, optional topology, per-loop splines, and focus boxes with stated reasons), not verified explanations of the model's internal reasoning.

Limitations. LLMSeg inherits its backend's failure modes: hallucinated detections on ambiguous prompts, occasional coordinate-format instability (mitigated but not eliminated by the repairs of §B), and per-call latency of seconds to minutes, which makes video and real-time use out of scope. Spline loops are a poor primitive for thin or fractal structures (Figure 14), and segmentation quality is upper-bounded by detect-box quality due to the confinement invariant. Finally, API-served backends make exact reproducibility subject to provider model updates; we pin model versions and release all records.

F PROMPT TEMPLATES

This section gives the three prompt templates that drive the pipeline, in the form the model receives them: box 1 is the detection call of Section 3, box 2 the initial tracing call, and box 3 the focus-box proposal call. Text in angle brackets is filled in at run time; the instance and focus-region blocks are shown filled for a two-instance example, since their layout is part of what the model reads. Long lines are re-wrapped to the column width. Everything in black is template text; the #-prefixed green lines are ours, not the model's, so a box can be copied whole and the notes stay comments. The green leading line stands for the image, which is attached to the request alongside the prompt text. The fourth call, the re-tracing half of a refinement pass, is box 2's spline rules restated once per focus region: it repeats the coordinate paragraph and the overlay description of box 3, lists the surviving focus regions with their type and reported shape, fixes the degree to cubic, asks for four to twelve control points per region instead of sixteen to sixty-four, and returns the same id-keyed list of closed loops.

Three properties of these templates matter more than their wording. First, every call states the coordinate space explicitly and every call that returns points restates the box those points must fall inside as a numeric inequality; the code clamps regardless, but a clamped-in point is a worse outline than a correctly placed one. Second, each prompt offers an escape hatch, an empty detection list, an empty loop list for an instance whose box holds nothing, an empty proposal list for a pass that finds no error, so that a failure to find something is expressible and does not have to be invented. Third, batched calls pin the emission order to ascending instance id, which is what makes a truncated reply salvageable.

Prompt #1: Instance Detection (Stage 1)

```
# <the input image>

The image is <W>x<H> pixels (origin top-left, x right, y down).
ALL coordinates are absolute PIXELS with 0 <= x <= <W> and 0 <= y <= <H>.

Find every DISTINCT instance of the following in the image: "<query>"

Return ONLY a JSON object (no prose, no markdown fences) with this exact schema:
{"detections": [{
  "id": <int starting at 0>,
  "label": "<short label>",
  "box": [x1, y1, x2, y2],
  "confidence": <0..1>,
  "morphology": {
    "shape": "<overall silhouette in a few words, e.g. 'compact rounded body with two upright
              protrusions'>",
    "aspect": "<tall|wide|square>",
    "convexity": "<convex|slightly-concave|highly-irregular>",
    "parts": ["<protruding/notable parts, e.g. 'left protrusion'>", "<'base contact'>", "..."]
  },
  "topology": {
    "components": <int number of disconnected pieces of the object silhouette>,
    "holes": <int number of enclosed background holes inside the silhouette>,
    "boundary_loops": <int total closed boundary curves = components + holes>,
    "notes": "<one short phrase, e.g. 'single blob, no holes'>"
  }
}]}

where box is [x1, y1, x2, y2] in the coordinate space described above (x1 < x2, y1 < y2, all values
inside the valid range for that space).
- One entry per DISTINCT instance -- do NOT return duplicate or nested boxes for the same object. Order
  detections in reading order (top-to-bottom, then left-to-right); ids will be reassigned by code, but
  keep them contiguous from 0.
- morphology.aspect must describe the instance as you SEE it, independently of the numbers you wrote in
  box; it is cross-checked against them.
- Make each box LARGE enough to ENTIRELY encompass that instance -- leave generous margin so no part of
  the silhouette (protrusions, extremities, soft edges, head, ears, paws) is clipped by the box. Prefer
  a roomy box over a tight crop; a later stage confines all outline tracing to this box, so an
  undersized box truncates the segmentation.
- morphology describes the object's FORM (shape of its outline) so a later stage can trace it; topology
  describes its STRUCTURE (how many separate blobs and how many holes) so a later stage knows how many
  closed curves to fit.
If nothing matches, return {"detections": []}.
Output the JSON and nothing else.
```

Prompt #2: Outline Tracing (Stage 2)

```
# <the input image>

The image is <W>x<H> pixels (origin top-left, x right, y down).
ALL coordinates are absolute PIXELS with 0 <= x <= <W> and 0 <= y <= <H>.

We have already DETECTED <N> instance(s) of "<query>" (same query / class). Each instance has its own
bounding box; a later stage confines that instance's outline to ITS box only.
```

```
INSTANCES (trace EVERY one):
- id=0 label="<label>" box=[120, 80, 540, 610]
    every x,y you return for id=0 must satisfy: 120 <= x <= 540 and 80 <= y <= 610
    morphology: {"shape": "compact rounded body with two upright ears", "aspect": "tall",
                 "convexity": "slightly-concave", "parts": ["left ear", "tail", "front paws"]}
    topology:   {"components": 1, "holes": 0, "boundary_loops": 1, "notes": "single blob, no holes"}
- id=1 label="<label>" box=[600, 140, 930, 585]
    # ... (one such entry per detected instance)

Your job has TWO parts:
  (1) pick ONE shared overlay tint color for the mask (same for all instances),
  (2) CHOOSE a spline degree (1 or 3) shared by every instance of this class, then for EACH instance
      trace its silhouette with piece-wise CLOSED 2D B-splines of that degree so the spline(s) tightly
      SURROUND that instance.

MASK COLOR (part 1):
Pick ONE color name from this fixed list:
  "lime", "green", "spring_green", "mint", "teal", "cyan", "turquoise", "sky_blue", "blue",
  "periwinkle", "indigo", "violet", "purple", "magenta", "hot_pink", "pink", "coral", "orange",
  "chartreuse"
Choose the color that maximizes contrast of a light translucent tint against BOTH the "<query>" AND its
surrounding background, so a later pass can clearly see what is covered vs not. Rules for the choice:
- ONE tint is shared by every instance: pick a hue that reads clearly on ALL of them and against each
  one's own local background, not just the first.
- Prefer a hue that is NOT already dominant in the object or in the nearby background.
- Do NOT pick a color that would wash out or blend into the object's own colors or hide fine edge
  detail under the tint.
- Return the name EXACTLY as listed (lowercase, underscores).

SPLINE RULES (part 2):
- Return exactly one entry per instance, keyed by the same integer "id".
- Every control point for instance id=K MUST lie inside THAT instance's box=[x1, y1, x2, y2] listed
  above.
- Fit ONE closed B-spline per connected blob of that instance (topology.components), plus ONE
  additional closed B-spline per interior hole (topology.holes), marking holes with "role":"hole".
  Outer silhouette loops use "role":"outer".
- Use each instance's morphology to place control points on the true outline (follow protrusions,
  concavities, corners, and curves named in the morphology).
- If one instance's box is empty, unreadable, or holds nothing matching the "<query>", return an EMPTY
  "splines" list for that id rather than inventing an outline. Still return its id, and still trace
  every other instance normally.
- Your whole reply must fit in ONE response, and <N> instances is a lot of coordinates. Emit entries in
  ASCENDING id order and FINISH each entry before starting the next; stay near the LOW end of the
  control-point range for the simple ones so that nothing gets cut off.
CHOOSE THE SPLINE DEGREE (1 or 3) from the object's form, then place the control points the way that
degree requires:
- Pick DEGREE 1 (closed polyline) for angular / polygonal / straight-edged objects, or simple
  silhouettes where a polygon is enough. Its control points are vertices ON the true object outline
  (not a looser control polygon): place them directly on the edge, ordered around the loop, denser at
  corners/bends and sparser on long straight spans.
- Pick DEGREE 3 (closed cubic B-spline) for organic / rounded / smoothly curved silhouettes. Its
  control points are the B-spline CONTROL POLYGON (the curve is smoother than and sits inside the
  polygon), so push them slightly OUTWARD toward the object edge, ordered around the loop.
- Use the SAME degree for EVERY spline (outer loops and holes). Report it in the top-level "degree"
  field AND set "degree" on each spline to that value.
- Choose the NUMBER of control points from the object's morphology and what you actually see inside the
  box -- typically 16-64 per loop. Use MORE points for intricate / irregular / many-parted outlines
  (closer to 64) and FEWER for simple, smooth, or rounded ones (near 16). Do not aim for a fixed count
  -- let the shape's complexity decide within that range; under-sampling a complex outline is worse
  than a few extra points on a simple one.

Return ONLY this JSON (no prose, no markdown fences):
{"mask_color": "<one name from the list>",
  "mask_color_reason": "<short why this hue contrasts with object+background>",
  "degree": <1 or 3>,
  "degree_reason": "<short why this degree fits the object's form>",
  "instances": [
  {"id": 0, "splines": [
    {"role": "outer", "degree": <same as top-level degree>, "closed": true,
     "control_points": [[x, y], [x, y], ...]}
  ]}
]}
Output the JSON and nothing else.
```

Prompt #3: Proposing Focus Regions (Stage 3)

```
# <the current overlay image>

This image shows the CURRENT segmentation of the "<query>":
  - the translucent <tint> tint marks the current MASK REGION (coverage),
  - the RED outline is the current mask BOUNDARY -- the main edge to compare,
  - each YELLOW rectangle is one instance's detection box (ids drawn beside them).
The <tint> fill is light on purpose: you can still see the <query>'s real edges/texture underneath. Use
the <tint> tint to tell what is already covered; use the red outline + the underlying pixels to judge
where the TRUE edge runs. Do NOT treat the soft tinted edge as the object boundary.
The image is <W>x<H> pixels (origin top-left, x right, y down).
ALL coordinates are absolute PIXELS with 0 <= x <= <W> and 0 <= y <= <H>.

DETECTION BOXES (each focus box MUST lie inside ONE of these):
- id=0 label="<label>" box=[120, 80, 540, 610]
    topology:   {"components": 1, "holes": 0, "boundary_loops": 1, "notes": "single blob, no holes"}
- id=1 label="<label>" box=[600, 140, 930, 585]
    # ... (one such entry per detected instance)

Your task is a COMPARISON: follow the RED outline and, at each part of it, look through the <tint> tint
at the actual object pixels to see where the TRUE edge of the <query> is. Report the places where the
```

```
two disagree.

Find FOCUS BOXES that mark where the mask is wrong and needs local fixing:
  - type "positive": a box that FULLY encloses an area that IS part of the <query> but is NOT tinted /
    lies OUTSIDE the red outline (MISSED, e.g. a protruding part or tip left outside the mask) -> to be
    ADDED.
  - type "negative": a box that FULLY encloses an area that IS tinted / INSIDE the red outline but is
    actually background (wrongly included, e.g. background bleeding beside the silhouette, or a gap
    that is not the <query>) -> to be REMOVED.
Check BOTH error directions along the red outline: missed object (positive) AND tinted background
(negative). Do not only hunt for misses.

Rules:
- Tag every focus box with "instance_id" naming which detection box it belongs to.
- Each focus box [x1, y1, x2, y2] MUST lie inside that instance's detection box.
- Make each box LARGE enough to ENTIRELY encompass the region to be fixed -- leave generous margin so
  no part of the miss / over-include is clipped by the box edge. Prefer one roomy box over a tight
  crop; undersized boxes cause the next-step spline to miss the true outline. Propose 1-6 boxes PER
  instance (up to 6 x <N> total), one per distinct problem area.
- Check EVERY one of the <N> instances listed above, not only the largest or the most obviously wrong
  one. Propose up to 6 boxes PER instance (do not dump the whole budget on one).
- Anchor each box on what you can actually SEE: it should cover the full gap between the red outline
  (or the tinted coverage) and the true edge of the <query> at that spot, plus margin. Place box edges
  on real image features (the object's boundary, a protrusion, the adjacent background), not on the
  soft tint alone.
- A pass may be ALL positive, ALL negative, or a mix -- there is NO requirement to include both types.
  Return only boxes that are genuinely wrong; never add a box of the other type just to have one of
  each.
- Only flag areas that are CLEARLY wrong in the current image; if a spot already looks correct, do not
  propose a box there (earlier fixes already show in the tint).
- For each box, describe the SHAPE of the region to be added/removed in a few words, and how intricate
  its outline is (e.g. "thin curved sliver", "small round bump", "jagged notch", "simple triangular
  wedge"). A later step uses this to decide how many control points to spend on the box, so be honest
  about whether the local edge is simple or wiggly.

Return ONLY this JSON (no prose, no markdown fences). Example shape (mix of types; your list may be
all-positive, all-negative, or mixed):
{"focus_boxes": [
  {"type": "positive", "instance_id": 0, "shape": "<shape + complexity of the region to add>",
    "box": [x1, y1, x2, y2]},
  {"type": "negative", "instance_id": 0, "shape": "<shape + complexity of the region to remove>",
    "box": [x1, y1, x2, y2]}
]}
Output the JSON and nothing else.
```