ChipLayoutBench: A Multimodal Benchmark for Vision-Language Models in Semiconductor Layout Understanding

Anonymous ACL submission

Abstract

Recent multimodal benchmarks have significantly expanded the evaluation of visual reasoning in vision-language models, but they primarily focus on holistic semantic understanding where answers can be inferred directly from images. In many engineering applications, however, the image alone is insufficient: exact answers exist only in the underlying structured data, requiring models to align visual regions with coordinate-level information and reason over precise numerical values. Semiconductor layout analysis is a representative example of this challenge, where engineers analyze rendered chip layouts while relying on the exact coordinate-level geometry underlying the visualization. Despite its practical importance, this fine-grained multimodal grounding capability remains largely unexplored. To address this gap, we introduce ChipLayoutBench, the first benchmark for general semiconductor layout understanding. The benchmark provides text-only, image-only, and multimodal representations across diverse reasoning and grounding tasks. Evaluations on eight recent VLMs show that current models consistently struggle with precise multimodal grounding, often performing worse when both modalities are combined. Our benchmark provides a domain-grounded testbed for advancing geometric reasoning and fine-grained multimodal grounding for structured engineering data.

1 Introduction

Multimodal benchmarks have significantly expanded the visual reasoning abilities evaluated in vision-language models (VLMs), including geometry solving, chart understanding, and reasoning over tables, diagrams, and documents (Masry et al., 2022; Yue et al., 2024; Lu et al., 2024; Zhu et al., 2025). In these settings, the answer can be read off the image at a holistic, semantic level–recognizing salient content, locating a relevant region, or reporting an approximate value. A different setting, however, has received far less attention: when the image alone is insufficient and the exact answer exists only in the underlying coordinate-level data, requiring models to align visual regions with structured data and compute exact quantities. This differs fundamentally from perception tasks such as detection or segmentation, which label or outline image regions but neither recover the exact underlying values nor jointly reason over the image and its coordinate-level data.

Such grounding is essential for real engineering applications, where understanding a figure requires linking visual elements to the exact numerical data behind them. A representative example is electronic design automation (EDA), where engineers inspect rendered chip layouts while reasoning over precise coordinate-level geometry. However, whether current VLMs can perform this fine-grained grounding remains largely unexplored, as existing benchmarks focus primarily on holistic visual reasoning rather than exact element-level alignment between images and structured data. Measuring and understanding this capability is a necessary step toward making VLMs practically useful for such engineering workflows.

Semiconductor physical layouts–the core data in EDA–are the most demanding instance of this challenge. A layout is a dense field of rectilinear polygons defined by exact coordinates and organized into layers, stored in formats such as Graphic Data System II (GDSII). When rendered, these polygons appear visually similar and densely packed, conveying overall spatial structure but not the exact coordinates, layer assignments, or geometric properties encoded in the underlying data. As a result, even a simple query—*which polygon corresponds to this highlighted region, and what is its exact area?*—requires fine-grained grounding between the rendered image and the coordinate-level layout data. Motivated by this, we introduce ChipLayoutBench, the first benchmark for gen-

Figure 1: Overview of CHIPLAYOUTBENCH: We visualize representative questions from recognition and description, localization and retrieval, geometric and spatial reasoning, and VLM-specific evaluation tasks. Our benchmark covers diverse question types designed to evaluate comprehensive understanding of semiconductor physical layouts.

eral semiconductor layout understanding.

Built from real layouts, CHIPLAYOUTBENCH provides each sample in text-only, image-only, and multimodal to isolate the contribution of each representation. The benchmark contains code-verified question–answer pairs spanning five categories: recognition and description, localization and retrieval, and geometric spatial reasoning evaluate structural and numerical understanding; practical layout tasks such as hotspot detection (HSD) and design rule checking (DRC) assess realistic verification capabilities; and a dedicated VLM-specific category directly tests cross-modal grounding by requiring models to align annotated image with their corresponding coordinate-level layout data.

Evaluating eight recent VLMs reveals a consistent limitation: providing both modalities rarely improves performance and often *degrades* accuracy compared to text alone. While models can extract exact values from text and infer coarse spatial structure from images, they frequently fail to correctly align the two modalities. In the VLM-specific tasks, models often retrieve coordinates from nearby polygons rather than the intended target, revealing weak fine-grained grounding between visual regions and coordinate-level data. Because success requires exact element-level correspondence rather than approximate semantic understanding, our benchmark exposes a fundamental weakness of current multimodal reasoning systems. Our contributions are summarized as follows:

- We introduce CHIPLAYOUTBENCH, the first benchmark for general semiconductor layout understanding, targeting exact, fine-grained reasoning and cross-modal grounding between rendered layouts and their exact coordinate-level GDS data.
- We provide every sample in text-only, image-only, and multimodal forms across five task categories, enabling a controlled measurement of how each representation contributes to precise geometric reasoning.
- Through evaluations on eight recent VLMs, we find that current models struggle to jointly reason over visual and textual layout information, often performing *worse* with both modalities than with text alone, revealing weak fine-grained multimodal grounding under exact-precision constraints.

2 Related Work

Reasoning Benchmarks over Structured Inputs Recent benchmarks have been proposed to evaluate reasoning capabilities of language and vision-language models over structured inputs. Mathe-

matical reasoning datasets such as GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and Math-Vista (Lu et al., 2024) focus on symbolic and numerical problem solving. Chart understanding benchmarks such as DVQA (Kafle et al., 2018), PlotQA (Methani et al., 2020), ChartQA (Masry et al., 2022), and ChartMuseum (Tang et al., 2025) evaluate reasoning over structured visual data.

While these benchmarks demonstrate strong reasoning capabilities over structured content, they typically allow approximate interpretations or tolerate small numerical errors. In contrast, our work focuses on reasoning over structured geometric representations where correctness depends on exact coordinate-level precision, highlighting a complementary and underexplored challenge for modern language and vision-language models.

Structured Representations and Vector-Based Inputs Recent work has explored machine learning over structured representations such as vector graphics, diagrams, and CAD data. Prior approaches include vector graphic models such as SVG-VAE (Lopes et al., 2019), DeepSVG (Carlier et al., 2020), and SVGFormer (Cao et al., 2023), as well as LLM-based SVG generation frameworks (Rodriguez et al., 2024; Wang et al., 2025a; Rodriguez et al., 2025). In the CAD domain, DeepCAD (Wu et al., 2021), CADLlama (Li et al., 2025), and ReCAD (Li et al., 2026) explored learning and generation over procedural CAD representations. Benchmarks such as SGP-Bench (Qiu et al., 2025) further evaluate semantic understanding over symbolic graphics programs including SVG and CAD.

However, existing works primarily focus on representation learning, reconstruction, or generation. In contrast, our work evaluates whether models can perform precise geometric and spatial reasoning directly over structured representations with exact coordinate-level relationships under realistic engineering constraints.

Machine Learning for Semiconductor Layout Understanding Machine learning has been widely applied to semiconductor layout analysis tasks such as lithography hotspot detection, optical proximity correction, and layout generation (Yang et al., 2017, 2018; Wen et al., 2022). Recent works have further explored language-model-based and topology-aware representation learning for semiconductor layouts (Chen et al., 2024, 2026; Zhang et al., 2026). While our benchmark also includes application-oriented tasks such as hotspot detection and design rule checking, its primary objective is

Figure 2: Distribution of QA samples across subcategories in the test set.

to evaluate whether current language and vision-language models can understand and reason over structured semiconductor layout representations without domain-specific training.

3 Benchmark construction

3.1 Dataset Taxonomy and Task Design

Our benchmark is organized into five categories, each targeting a different aspect of layout understanding. The question candidates are curated by semiconductor engineers and AI researchers to evaluate fundamental layout understanding, serving as a foundation for developing domain-specific semiconductor AI models and downstream tasks. Example question-answer pairs are in the Table 1.

Recognition and Description evaluates whether a model can identify and describe basic geometric content in a layout. This category includes Counting, which measures whether the model can count polygons satisfying specified conditions; and Attributes, which tests the identification of polygon properties such as vertex count, layer membership, shape type, or size-based extrema.

Localization and Retrieval measures whether a model can find or retrieve relevant polygons from the layout. It includes Shape Localization, which asks the model to retrieve polygon ids based on position, boundary proximity, or distinctive shape cues; Pattern Matching, which focuses on exact geometric matching between polygons and a reference polygon;and Similarity, which evaluates retrieval of the polygon with the closest match under a defined metric.

Geometric and Spatial Reasoning focuses on more compositional reasoning over layout geometry and topology. Its subcategories include Area

and Geometric Measures, covering polygon area, overlap, distance, containment, and related quantitative properties; Topological and Connectivity Reasoning, which tests reasoning over adjacency, connected components, and connectivity paths; and Layer-based Geometry Reasoning, which evaluates cross-layer interactions such as union, intersection, spacing, and mediated connectivity.

Layout Tasks focuses on practical semiconductor layout analysis tasks that are closely related to downstream physical design workflows. In the current benchmark, this category includes Hotspot Detection, which determines whether a layout sample contains vulnerable or problematic patterns, and Design Rule Check, which evaluates the number of rule violations under specified design constraints.

VLM-specific Evaluation focuses on multimodal tasks that require joint reasoning over rendered layout images and textual GDS representations. This category includes Cross-modal Alignment, which evaluates whether the model can associate annotated visual regions with corresponding polygon ids or coordinates, and Image-grounded Reasoning, which measures the ability to reason about visually indicated edges, vertices, and polygon relationships directly from layout renderings.

3.2 Data Curation and Processing

Our dataset is constructed from semiconductor layouts generated using the OpenROAD framework (Ajayi and Blaauw, 2019) across multiple technology nodes and circuit designs. For general layout understanding tasks, we collect layouts from diverse circuits, including *aes, gcd, swerv, jpeg*, and *ibex*, spanning multiple open-source platforms, including *Sky130hd, Nangate45*, and *Asap7*, all distributed under permissive licenses[1]. This diversity provides broad coverage of layout patterns, densities, and structural variations.

Input data collection To construct dataset samples, we employ a grid-based stratified sampling scheme over full-chip layouts to capture both dense and sparse layout regions while preserving local geometric relationships (details in Appendix A.1). From the raw GDSII files, we extract polygonal representations, including vertex coordinates and layer information. For layer-wise sampling, we consider the adjacency and structural dependencies of the physical design. Specifically, functionally related neighboring layers, such as via and metal layers, are jointly sampled to preserve realistic inter-layer connectivity and routing structures commonly observed in semiconductor layouts. This multi-layer sampling process enables the dataset to capture cross-layer geometric dependencies in modern physical design. In addition, we diversify the spatial clip sizes during sampling to capture layout patterns across multiple geometric scales.

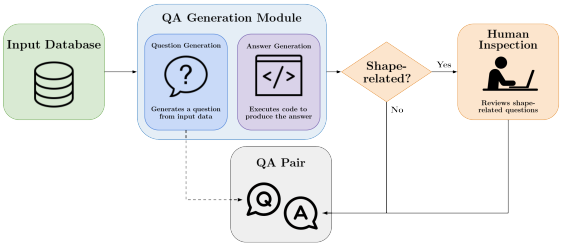


Figure 3: Overview of the QA construction process for CHIPLAYOUTBENCH.

We preserve the original coordinate values as defined in the layout. As a result, coordinate ranges can vary significantly across samples, reflecting differences in layout scale and sampling regions. We retain this variability to reflect realistic layout conditions and to evaluate whether models can reason over geometric structures under varying numerical scales. This introduces an additional challenge for language models, as they must process and reason over numerical values that are not bounded to a fixed range. The polygon representations are standardized through canonical ordering of vertices to ensure consistency in representation.

QA construction Question-answer pairs are generated using a custom engine designed to operate directly on the extracted layout data. Most questions are constructed through rule-based templates combined with deterministic geometric computations, allowing the corresponding ground-truth answers to be obtained directly through code execution[2] for properties such as area, adjacency, containment, and other spatial relationships (Fig. 3). This ensures that the majority of tasks have unambiguous, reproducible, and verifiable answers without reliance on heuristic labeling. For shape-related queries that cannot be fully specified through explicit geometric rules, such as identifying complex patterns like T-shapes, U-shapes, or H-like structures, we incorporate human annotation. These cases are manually inspected by the authors.

Layout tasks In addition to synthetic reasoning tasks, we include application-oriented evaluations

[1] Derived samples follow the access restrictions of the original layout datasets.

[2] AI assistants were used for coding support. All scientific content was verified by the authors.

Category	Subtask	Question	Answer
Recognition & Description	Counting	How many polygons have 4 vertices?	4
	Attributes	Which polygon has the largest area?	poly_00001
Localization & Retrieval	Shape Localization	Which polygon is closest to the top-left corner of the region?	poly_00042
	Pattern Matching	Find all polygons identical to polygon poly_00001.	{poly_00002}
	Similarity	Which polygon has area closest to polygon poly_00013	poly_00019
	Shape Retrieval	Which polygons are connected to polygon poly_00001?	poly_00004
Geometrical & Spatial Reasoning	Area & Geometric Measures	What is the total area of the extruded region in polygon poly_00038?	1.5314
	Topological Reasoning	Which polygons belong to the same connected component as polygon poly_00001?	{poly_00001, poly_00003}
	Layer-based Reasoning	Which polygons are connected to polygon poly_00008 via layer (7, 0)?	{poly_00016, poly_00017}
Layout Tasks	DRC	With the m1 width rule set to 0.018000μm, how many violations are in the sample?	3
	Hotspot Detection	Based on the support examples, should the target sample be labeled hotspot or non-hotspot?	non-hotspot
VLM-specific Evaluation	Cross-modal Alignment	Which polygon ID matches the highlighted polygon in the image?	poly_00082
	Image-grounded Reasoning	In the image, polygon A contains edge 1 and polygon B contains edge 2. What is the distance between edge 1 and edge 2?	0.002

Table 1: Example question–answer pairs for each task subcategory in CHIPLAYOUTBENCH. Questions are generated using predefined templates instantiated from real semiconductor layout content.

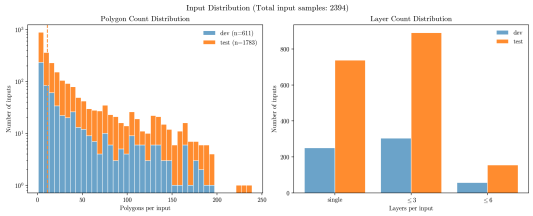


Figure 4: Distribution of input samples. Left: number of polygons per sample. Right: number of samples per layer count.

derived from standard layout verification workflows. For design rule checking, we use KLayout (Koefferlein, 2025) to perform rule-based validation, including checks for width, spacing, and enclosure violations. These evaluations assess whether models can reason about geometric rule constraints directly from layout structures. For hotspot detection, we follow a setup similar to LithoBench (Zheng et al., 2023a). A layout region is labeled as a hotspot when the edge placement error (EPE) exceeds a threshold (EPE > 0).

3.3 Modalities and Input Representations

To study how modern models process structured layout data, we provide each sample in multiple input modalities. This design enables a controlled evaluation of how different representations affect geometric reasoning and structured understanding.

Text-based Representation GDSII is not inherently a language-like format, as it encodes geometric information through hierarchical binary structures rather than sequential tokens. To make layout data compatible with language models, we convert GDS representations into a structured textual format inspired by GLP representations (Fig. 1).

In this representation, each polygon is serialized as a sequence of tokens describing its vertex coordinates and attributes (e.g., layer information), forming a structured textual description of the layout. This conversion preserves exact geometric information while enabling models to process layout data similarly to natural language or source code. To support question-answer generation and spatial reference within the benchmark, each polygon is assigned a unique identifier, which is appended to the serialized representation.

Image-based Representation In the image modality, layouts are rasterized into images using a resolution of 1 nm per 1 pixel, preserving fine-grained geometric details. While this representation provides strong spatial and visual cues, it does not inherently encode absolute coordinate information.

To address this limitation, we augment each image with a minimal textual description specifying the starting coordinate of the top-left polygon in the sampled region, along with the rasterization resolution and the color code to represent the layer information. This provides a reference frame that enables models to relate pixel-space representations to underlying geometric coordinates.

Multimodal Representation In the multimodal setting, models are provided with both the GLP-based textual representation and the corresponding rendered image with spatial grounding information. This allows models to integrate structured numerical information with visual spatial cues.

This setup enables us to investigate whether multimodal models can effectively align symbolic and visual representations, and whether combining modalities improves performance on tasks requiring precise geometric reasoning. In particular, it allows us to analyze whether exact reasoning benefits more from structured representations, visual context, or their combination.

Model	Mode	Average	Rec&Desc	Loc&Ret	Spatial		Layout	
					Acc ↑	Reg ↓	DRC (L1) ↓	HSD (F1) ↑
Open-source models								
Qwen3.5-27B (2026.4)	Text	69.7	71.0	73.2	61.8	0.039	**49.2**	**0.559**
	Image	36.5	44.0	34.5	28.5	81.2	70.9	0.498
	Both	68.4	69.1	75.1	56.1	**0.014**	58.2	0.540
Qwen3.6-27B (2026.4)	Text	**76.2**	**73.7**	**84.0**	**67.3**	0.398	54.3	0.532
	Image	36.6	43.5	34.7	29.3	272.5	75.7	0.470
	Both	72.1	70.4	80.0	61.5	0.036	50.8	0.497
Gemma4-31B-Instruct (2026.4)	Text	73.3	75.2	77.6	63.3	0.248	49.3	0.512
	Image	32.5	38.3	31.3	24.9	89.6	69.7	0.449
	Both	71.0	71.7	77.7	59.0	0.297	53.3	0.489
GLM-4.6V (2025.12)	Text	52.7	63.7	62.5	19.7	2.415	53.4	0.476
	Image	22.1	33.0	15.3	16.8	82.9	71.6	*0.513
	Both	55.8	64.0	60.5	35.5	1.265	52.6	*0.518
InternVL3.5-241B-A28B (2025.8)	Text	40.2	54.1	44.2	12.4	1.322	51.9	0.541
	Image	20.2	30.4	18.2	8.24	1085	69.0	*0.454
	Both	35.6	51.3	36.1	10.9	0.800	54.4	*0.516
Qwen3.5-397B-A17B (2026.4)	Text	60.0	72.0	60.5	39.1	1.590	57.7	0.483
	Image	32.5	42.9	28.8	22.8	2846	70.1	0.518
	Both	51.6	62.3	52.5	34.0	659.8	55.6	0.525
Proprietary models								
GPT5.4-mini	Text	44.7	54.4	52.5	17.0	0.515	**66.6**	0.508
	Image	32.5	40.4	32.9	20.0	1.836	79.5	0.442
	Both	36.9	55.5	35.1	11.5	0.585	67.5	0.405
Gemini3.1-flash-lite	Text	**67.5**	**71.4**	**75.9**	**47.7**	**0.335**	69.7	**0.546**
	Image	36.2	45.3	34.8	24.7	1327	71.3	0.541
	Both	62.8	68.9	69.8	42.0	0.362	69.2	0.512

Table 2: Benchmark performance across modalities and models, ordered by increasing model parameter size. The best results for open-source and proprietary models are highlighted in **bold**, respectively. HSD results marked with an asterisk are evaluated using three in-context support examples due to token-length limitations.

3.4 Dataset Statistics

The final dataset contains 2,394 input layouts, with 611 and 1,783 samples in the dev and test sets, respectively. The dataset contains 2,385 questions for the dev set and 6,926 questions for the test set. Figure 4 illustrates the distributions of the number of polygons, ranging from 1 to 200+, and layers in the layouts, while Figure 2 shows the distribution of questions across different task categories.

3.5 Evaluation Protocol

We evaluate all tasks and input modalities under a unified protocol. For structured question-answering tasks with discrete outputs, including binary and multiple-answer questions, we report exact-match accuracy, requiring predictions to exactly match the ground-truth answers.

For tasks involving numerical outputs, such as geometric measurements (e.g., area, distance, and overlap) and DRC, we formulate the problem as a regression task. In the case of DRC, models are required to predict the number of rule violations within a layout region. Predictions are evaluated using the L1 error with respect to the ground-truth values, reflecting the model's ability to perform quantitative reasoning over structured geometric data. For HSD, which is formulated as a binary classification task, we report the F1 score due to the class imbalance in the dataset.

All evaluations are conducted consistently across text-only, image-only, and multimodal inputs, enabling direct comparison across representations. This setup allows us to isolate the effect of input modality on reasoning performance while keeping the task definitions and ground-truth annotations fixed. Details regarding the instruction templates used to enforce the desired output format are provided in the Appendix B.3.

4 Experiments

4.1 Experimental Setup

We evaluate ChipLayoutBench using 2 proprietary models and 6 open-source models, ranging from 27B to 397B parameters, for 3 different modalities. For the proprietary models, we evaluate on GPT5.4-mini (OpenAI, 2026), Gemini-3.1-Flash-Lite (Google, 2026a). For open-source models, we test Qwen3.5-27B, Qwen3.6-27B, Qwen3.5-397B-A17B (Alibaba, 2026), Gemma4-31B-Instruct (Google, 2026b), GLM4.6V (Z.ai, 2026), and InternVL3.5-241B-A28B (Wang et al., 2025b). All models are evaluated under a unified instruction and experimental setup for fair comparison across modalities and architectures.

4.2 Main Results

We report the overall performance in Table 2 and the VLM-specific evaluation in Table 3. Among

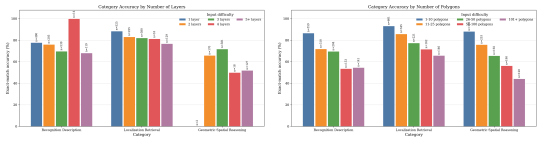


Figure 5: Performance of Qwen3.6-27B with respect to the number of layout layers (left) and polygons (right) in each sample.

the evaluated models, Qwen3.6-27B achieves the strongest overall performance. Interestingly, increasing model scale does not consistently translate to better performance; instead, more recently released models tend to outperform older ones regardless of parameter count. Furthermore, closed-source models do not demonstrate a clear advantage over open-source models, with the best overall performance achieved by an open-source model. This trend may reflect the domain-specific characteristics of the benchmark. Across question categories, most models achieve strong performance on recognition and description, as well as localization and retrieval tasks. In contrast, geometric spatial reasoning tasks, which require precise numerical computation and multi-step reasoning, remain relatively more challenging.

For application-oriented layout tasks such as hotspot detection and design rule checking, the evaluation is intended to assess whether current models can exhibit zero-shot layout reasoning capabilities without task-specific external training. Since hotspot detection depends on lithography simulation and manufacturability-related knowledge that are not explicitly available during pretraining, the overall poor performance suggests that current general-purpose models struggle to reason about practical layout verification tasks directly from geometric structures alone. Similarly, models perform substantially worse on DRC regression tasks, which require precise numerical reasoning and fine-grained geometric measurements over layout structures, indicating that accurate quantitative understanding of semiconductor layouts remains a major challenge for existing models.

Observation 1 Since the benchmark primarily consists of questions requiring exact numerical precision, text-based representations achieve the strongest performance overall. In contrast, all evaluated models struggle when relying solely on the image modality. Although image-based representations provide sufficient cues for coarse structural understanding, they are often inadequate for tasks that require accurate numerical computation or fine-grained geometric reasoning. Interestingly, most models also degrade when both modalities are provided together, potentially indicating weak grounding between textual and visual information.

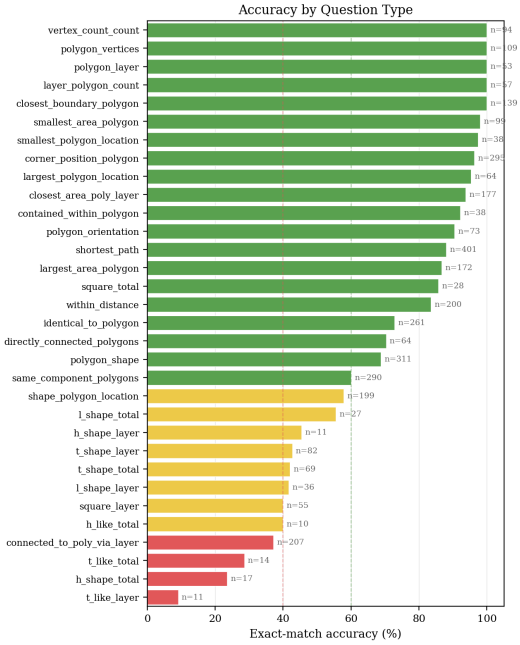


Figure 6: Performance of Qwen3.6-27B across different question types in ChipLayoutBench.

Observation 2 All evaluated models exhibit a similar performance trend across question categories. We visualize the performance across question types using the Qwen3.6-27B model in Fig. 6. In general, perception-oriented tasks, such as finding polygons closest to the boundary, counting polygons, and estimating polygon orientation, are relatively easy for most models. In contrast, performance drops noticeably on tasks requiring complex reasoning, such as connectivity across layers and shape-related semantic understanding. The similar trends across models suggest that current VLMs still exhibit weak grounding between textual and visual information. Interestingly, GLM-4.6V is the only model that consistently improves when both modalities are jointly used, achieving a 15.8% gain in geometric spatial reasoning. While the overall performance gap in the main benchmark remains relatively large at 16.3%, the gap becomes substantially smaller in the VLM-specific evaluation (7.1%), suggesting that stronger image-text grounding enables more effective multimodal reasoning.

Observation 3 We additionally analyze model performance with respect to the complexity of the input layouts, measured by the number of layers and the number of polygons in each sample in Fig. 5. Performance tends to degrade more noticeably as

Model	Average	Cross-modal Alignment		Image Grounded Reasoning	
		Acc ↑	Reg ↓	Acc ↑	Reg ↓
Open-source models					
Qwen3.5-27B	65.1	43.2	139.3	89.1	0.597
Qwen3.6-27B	71.1	58.5	**138.4**	84.9	**0.349**
Gemma4-31B-Instruct	**76.1**	**63.6**	142.4	**89.8**	0.755
GLM-4.6V	69.0	57.2	145.7	81.9	2.543
InternVL3.5-241B-A28B	43.5	21.1	146.6	68.0	1.575
Qwen3.5-397B-A17B	70.5	58.7	156.2	83.3	3.20
Proprietary models					
GPT5.4-mini	**68.3**	**52.3**	146.0	**85.6**	**1.156**
Gemini3.1-flash-lite	61.4	45.3	**141.2**	78.9	2.167

Table 3: Benchmark performance on the VLM-specific evaluation across different models. The average ground-truth values for the regression tasks in cross-modal alignment and image-grounded reasoning are 276 and 0.112, respectively.

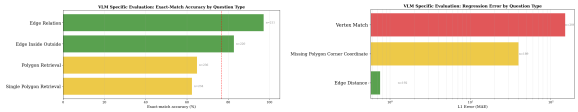


Figure 7: Performance of Gemma4-31B-Instruct across different question types in the VLM-specific evaluation.

the number of polygons increases, suggesting that dense geometric structures and larger numbers of interacting elements substantially increase reasoning difficulty. As qualitatively shown in the Appendix Fig. 17, models often attempt to reason over nearly all polygons in the layout, rather than selectively focusing on the most relevant polygons or suppressing low-confidence candidates during the reasoning process. This behavior suggests that current models struggle to efficiently narrow the reasoning space in complex layouts. Potentially, stronger integration of visual representations could help reduce the effective search space by guiding attention toward geometrically relevant regions.

4.3 VLM-specific Evaluation

The VLM-specific evaluation is designed to analyze how effectively vision-language models perform grounding between textual and visual information. In particular, these tasks examine whether models can accurately associate textual annotations with corresponding geometric regions and numerical attributes in layout images. Among all evaluated models, Gemma4-31B-Instruct achieves the strongest overall performance, but still struggles to estimate exact coordinate values reliably.

We observe consistently weak grounding performance across current VLMs. While models perform relatively better on tasks such as classifying edge relations or inferring simple spatial relationships, which can largely be inferred from the image modality alone, they struggle to reliably identify matching shapes or retrieve exact coordinate values (Fig. 7). Even in relatively simple cases containing only a few polygons, models frequently localize the wrong region after lengthy reasoning processes, often retrieving the coordinates of a neighboring polygon rather than the specified target (Fig. 26 in the Appendix).

These observations suggest that current VLMs rely more heavily on coarse visual reasoning and contextual estimation than on precise text recognition and multimodal grounding. The problem becomes particularly severe in semiconductor layouts, where correctness depends on exact numerical precision and fine-grained geometric correspondence rather than approximate semantic understanding.

5 Conclusion

We introduce ChipLayoutBench, a benchmark for evaluating the layout reasoning capabilities of VLMs on semiconductor layout data across both text and image modalities. Motivated by EDA, our benchmark evaluates fine-grained geometric reasoning and multimodal grounding under exact-precision constraints.

Experimental results show that current models can perform basic structural understanding, but still struggle with precise geometric and quantitative reasoning. In particular, current VLMs exhibit weak grounding between textual and visual information, limiting their ability to reason across modalities during fine-grained spatial analysis.

We believe ChipLayoutBench provides a challenging testbed for advancing geometric reasoning, numerical precision, and multimodal grounding in foundation models. In the long term, layout-aware language and vision-language models may enable more general reasoning over structured spatial data, with potential applications in scientific computing, engineering design, and other geometry-intensive domains.

Limitations

Our benchmark has several limitations. First, the input representation remains under-explored. We adopt a simplified GLP-style textual serialization and rasterized image rendering, which do not fully capture the hierarchical and structural characteristics of real GDSII layouts. Practical semiconductor layouts contain repeated cells, multi-level abstractions, and rich connectivity information, many of which are flattened or omitted in our formulation. In addition, the grounding between textual and visual representations is relatively simple, leaving substantial room for future research on more faithful multimodal alignment for layout data.

Second, although shape-related reasoning questions are among the most challenging categories for current models, we exclude highly ambiguous or difficult-to-define geometric patterns from the benchmark. Shape labels are manually annotated rather than generated through explicit geometric rules, and some irregular structures are omitted to maintain annotation consistency and reliability. However, such irregular geometric patterns often correspond to the practical challenges encountered by semiconductor engineers during design analysis and verification.

Finally, our benchmark is constructed from specific semiconductor layout datasets and may not fully represent the diversity of industrial layouts and manufacturing processes. Because the benchmark is largely template-driven, future models may overfit to benchmark-specific patterns rather than develop generalized layout understanding. We hope this benchmark serves as a foundation for future research on layout-aware multimodal reasoning with more realistic hierarchical representations and expert-defined structural reasoning tasks.

References

Tutu Ajayi and David Blaauw. 2019. Openroad: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference*.

Alibaba. 2026. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5.

Defu Cao, Zhaowen Wang, Jose Echevarria, and Yan Liu. 2023. Svgformer: Representation learning for continuous vector graphics using transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10093–10102.

Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. Deepsvg: A hierarchical generative network for vector graphics animation. *Preprint*, arXiv:2007.11301.

Silin Chen, Kangjian Di, Andy Liu, Guohao Wang, Wenzheng Zhao, Binwu Zhu, Li Du, and Ningmu Zou. 2026. Geometric-robust contrastive learning framework with layout topology representation and self-calibration voting for hotspot detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Yuyang Chen, Yiwen Wu, Jingya Wang, Tao Wu, Xuming He, Jingyi Yu, and Hao Geng. 2024. Llm-hd: Layout language model for hotspot detection with gds semantic encoding. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Google. 2026a. Gemini 3.1 flash-lite. https://deepmind.google/models/gemini/flash-lite/.

Google. 2026b. Gemma 4. https://deepmind.google/models/gemma/gemma-4/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Kushal Kafle, Scott Cohen, Brian Price, and Christopher Kanan. 2018. Dvqa: Understanding data visualizations via question answering. In *CVPR*.

Matthias Koefferlein. 2025. Klayout layout viewer and editor. https://www.klayout.de. Accessed: 2026-05-14.

Jiahao Li, Yusheng Luo, Yunzhong Lou, and Xiangdong Zhou. 2026. Recad: Reinforcement learning enhanced parametric cad model generation with vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 6190–6198.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. 2025. Cad-llama: leveraging large language models for computer-aided design parametric 3d model generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18563–18573.

Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2024. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations (ICLR)*.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279, Dublin, Ireland. Association for Computational Linguistics.

Nitesh Methani, Pritha Ganguly, Mitesh M. Khapra, and Pratyush Kumar. 2020. Plotqa: Reasoning over scientific plots. In *The IEEE Winter Conference on Applications of Computer Vision (WACV)*.

OpenAI. 2026. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Xiao, Katherine Collins, Joshua B Tenenbaum, Adrian Weller, Michael J Black, and Bernhard Schölkopf. 2025. Can large language models understand symbolic graphics programs? In *International Conference on Learning Representations*, volume 2025, pages 26265–26311.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2024. Starvector: Generating scalable vector graphics code from images and text. *Preprint*, arXiv:2312.11556.

Juan A. Rodriguez, Haotian Zhang, Abhay Puri, Aarash Feizi, Rishav Pramanik, Pascal Wichmann, Arnab Mondal, Mohammad Reza Samsami, Rabiul Awal, Perouz Taslakian, Spandana Gella, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2025. Rendering-aware reinforcement learning for vector graphics generation. *Preprint*, arXiv:2505.20793.

Liyan Tang, Grace Kim, Xinyu Zhao, Thom Lake, Wenxuan Ding, Fangcong Yin, Prasann Singhal, Manya Wadhwa, Zeyu Leo Liu, Zayne Sprague, Ramya Namuduri, Bodun Hu, Juan Diego Rodriguez, Puyuan Peng, and Greg Durrett. 2025. Chartmuseum: Testing visual reasoning capabilities of large vision-language models. *Preprint*, arXiv:2505.13444.

Feiyu Wang, Zhiyuan Zhao, Yuandong Liu, Da Zhang, Junyu Gao, Hao Sun, and Xuelong Li. 2025a. Svgen: Interpretable vector graphics generation with large language models. *arXiv preprint arXiv:2508.09168*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, and 56 others. 2025b. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *Preprint*, arXiv:2508.18265.

Liangjian Wen, Yi Zhu, Lei Ye, Guojin Chen, Bei Yu, Jianzhuang Liu, and Chunjing Xu. 2022. Layoutransformer: Generating layout patterns with transformer via sequential pattern modeling. In *Proceedings of the 41st IEEE/ACM international conference on computer-aided design*, pages 1–9.

Rundi Wu, Chang Xiao, and Changxi Zheng. 2021. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6772–6782.

Haoyu Yang, Shuhe Li, Yuzhe Ma, Bei Yu, and Evangeline FY Young. 2018. Gan-opc: Mask optimization with lithography-guided generative adversarial nets. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6.

Haoyu Yang, Jing Su, Yi Zou, Bei Yu, and Evangeline FY Young. 2017. Layout hotspot detection with feature tensor generation and deep biased learning. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, and 1 others. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9556–9567.

Z.ai. 2026. Glm-4.6v: Open source multimodal models with native tool use. https://z.ai/blog/glm-4.6v.

Xinyun Zhang, Yuyang Chen, Yiwen Wu, Su Zheng, Ran Chen, Min Li, Hao Geng, Binwu Zhu, Mingxuan Yuan, and Bei Yu. 2026. From contrastive to generative alignment: Large-scale hierarchical multi-modal pre-training for hotspot detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Su Zheng, Haoyu Yang, Binwu Zhu, Bei Yu, and Martin Wong. 2023a. Lithobench: Benchmarking ai computational lithography for semiconductor manufacturing. *Advances in Neural Information Processing Systems*, 36:30243–30254.

Su Zheng, Bei Yu, and Martin Wong. 2023b. Openilt: An open source inverse lithography technique framework. In *2023 IEEE 15th International Conference on ASIC (ASICON)*, pages 1–4. IEEE.

Junnan Zhu, Jingyi Wang, Bohan Yu, Xiaoyu Wu, Junbo Li, Lei Wang, and Nan Xu. 2025. Tableeval: A real-world benchmark for complex, multilingual, and multi-structured table question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7137–7157.

A Dataset Collection

A.1 Sampling Strategy

Algorithm 1 Tile Sampling and Filtering Pipeline

Require: GDS layout $\mathcal{L}$
Ensure: Filtered tile set $\mathcal{T}$

```
Preprocess layout L
Divide layout into grid-based cells C
for each cell c ∈ C do
    Compute polygon density of c
    if density of c is below threshold then
        continue                                  ▷ Stage 1
    end if
    Randomly sample positions inside c
                                                  ▷ Stage 2
    for each sampled position p do
        Extract tile centered at p
        Clip polygons inside tile boundary
        if overlapping polygons on the same layer are separated then
            Merge separated polygons
        end if                                    ▷ Stage 3
        Compute precise polygon density inside tile
        if tile density is below threshold then
            continue                              ▷ Stage 4
        end if
        if tile highly overlaps with existing samples then
            continue                              ▷ Stage 5
        end if
        Save tile into T
    end for
end for
    return T
```

Each design has different layout sizes and geometric scales. Moreover, because semiconductor layouts often contain large empty regions, naive grid-based sampling may generate many redundant samples, while purely random sampling does not guarantee sufficient coverage of the overall layout distribution. To better balance sample diversity and spatial coverage, we adopt a stratified sampling strategy following the protocol described in Algorithm A.1.

Stage 1: Cell Filtering This stage filters out cells whose polygon density is too low after dividing the layout into grid-based cells.

Stage 2: Sampling Inside Cell This stage performs random sampling within each selected cell.

Stage 3: Tile Extraction This stage extracts tiles from the original GDS layout. Since polygons on the same layer may be separated despite having overlapping regions, an additional polygon merging process is applied.

Stage 4: Density Filtering If the polygon density within a clipped sample is too low, the sample is considered to contain insufficient pattern information and is filtered out.

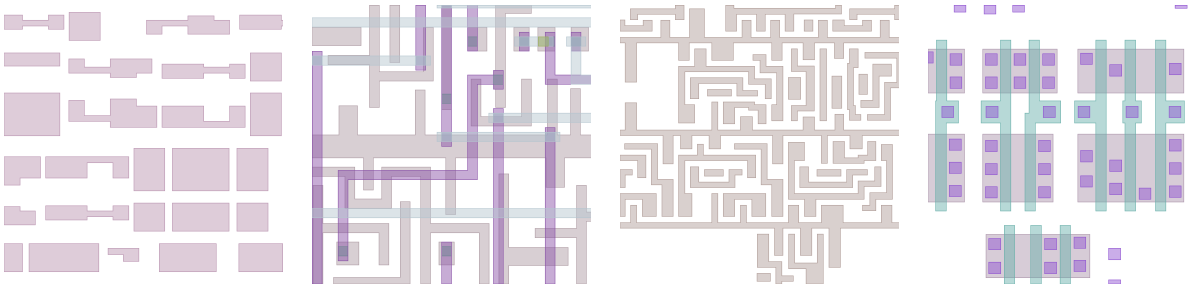
Stage 5: Duplicate Filtering Because random sampling may generate highly overlapping samples within the same cell, duplicate-like samples are filtered based on overlap conditions.

A.2 Dataset distribution

Layer and Clip Configuration Each design contains different technology nodes and layout scales; therefore, we adapt the clip size for each design to capture diverse geometric scales and pattern densities effectively. To reflect the structural characteristics of semiconductor layouts, we perform layer sampling in two ways: (1) shape-diverse sampling for single-layer inputs, and (2) adjacent-layer sampling based on the physical and functional relationships between layers for multi-layer inputs. Detailed design specifications are summarized in Table 4, and example input samples are illustrated in Fig. 8.

Additional Distribution information From the collected input sources, we randomly split the dataset into 5%, 15%, and 80% portions for the dev, test, and training sets, respectively, and visualize their distributions in Fig. 9. The training set will be released upon request.

Since the primary goal of this benchmark is to evaluate general layout understanding, we additionally assess models on a realistic downstream task of hotspot detection using a smaller curated dataset, where the distribution is shown in Fig. 10. A layout sample is classified as a hotspot when the edge placement error (EPE) exceeds zero, where EPE is computed through inverse lithography technology (ILT) simulation. Specifically, we perform ILT simulation using OpenILT (Zheng et al., 2023b) on the Metal-2 layer of selected *gcd* layouts in the



(a) Active layer in Nangate45 (b) m1,v1,m2,v2,m3,v3 in Nangate45 (c) li1 in Sky130hd (d) Diff,poly,licon1 in Sky130hd

Figure 8: Sample layouts with different layer configuration

Table 4: Layer and clip configurations for each semiconductor design used in CHIPLAYOUTBENCH.

Design	Circuit	Layout size	Grid size	Clip sizes	Single-layer	Multi-layers
Sky130hd	aes, ibex	622.28um x 622.28um	5x5	1024, 2048, 4096, 8192	M1, M2, npc, li1	licon1-li1, diff-poly-licon1, mcon-met1-via
Nangate45	jpeg, swerv	249.85um x 250.6um	5x5	512, 1024, 2048, 4096	active, metal2	active-poly-contact, m1-v1-m2-v2-m3-v3
Asap7	gcd, swerv wrapper	16.2um x 16.2um	1x1	256, 512, 1024, 2048	m1, lisd	fin-gate-gcut-active, lig-lisd, m1-v1-m2-v2

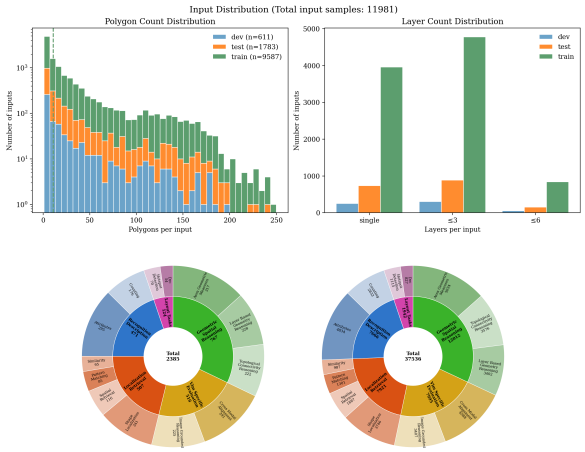


Figure 9: Distribution of the full benchmark dataset. The bottom-left and bottom-right donut plots illustrate the QA category distributions for the development and training splits, respectively.

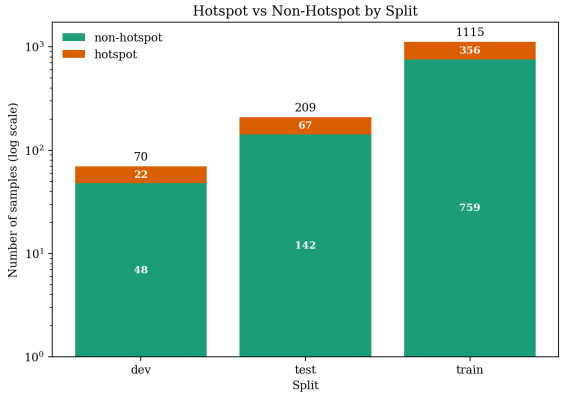


Figure 10: Distribution of the hotspot detection dataset across different data splits.

Nangate45 technology node.

A.3 Shape inspection

Since shape categories cannot be reliably defined through explicit rules alone, we manually annotate polygon shapes using the GUI shown in Fig. 11 and update the corresponding answers accordingly.

B Details and More Examples in QA

B.1 Sub-category Description

Counting: Questions that ask for numeric counts of layout objects matching simple criteria, such as polygons on a layer, polygons with a certain vertex count, or polygons assigned to a benchmark shape label.

Attributes: Questions about intrinsic polygon or layout properties, including shape class, layer membership, vertex count, orientation, and largest or smallest polygon by area.

Shape localization: Questions that ask which polygon satisfies a spatial or shape-based description, such as the top-left polygon, the smallest polygon, or the polygon with a specific shape label.

Pattern matching: Questions that ask for all polygons geometrically identical to a reference polygon after translation normalization, excluding rotated, reflected, or scaled variants.

Similarity: Questions that retrieve the polygon, often on another layer, whose area is closest to a reference polygon.

Spatial retrieval: Questions that return polygon sets based on spatial relationships such as direct connection, containment, or being within a specified distance of a reference polygon.

Area geometric measures: Questions involving

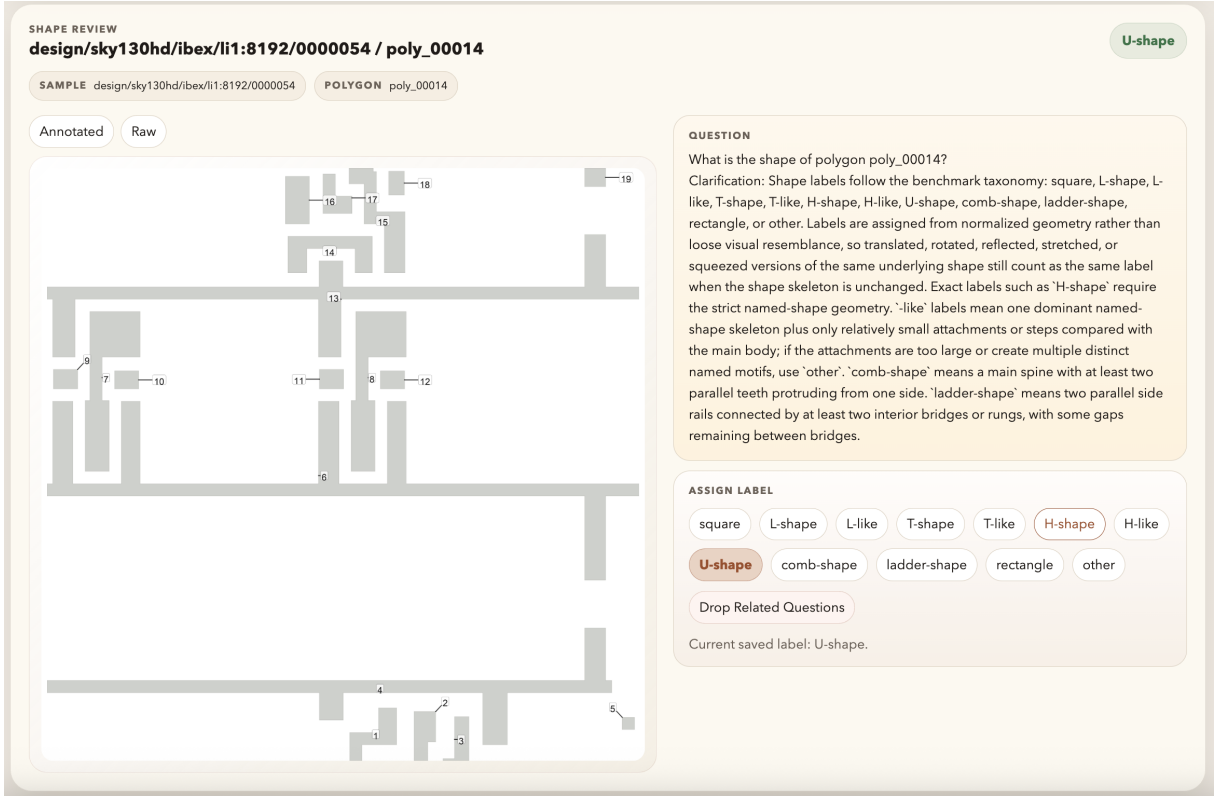


Figure 11: Graphical user interface (GUI) used for shape inspection.

numeric geometry calculations, including polygon area after directional edge expansion and extruded-region area, depth, or distance.

Layer based geometry reasoning: Questions that reason over whole layers as geometry unions, including layer area, layer OR area, cross-layer distance, protrusion, and mediated cross-layer connectivity.

Topological connectivity reasoning: Questions about graph-like polygon connectivity, such as connected components and shortest paths through overlapping or boundary-sharing polygons.

Cross modal alignment: VLM-specific questions that require matching image evidence to GDS text, such as identifying highlighted polygons, matching highlighted vertices to coordinates, recognizing cropped polygons, or finding a polygon missing from the text but present in the image.

Image grounded reasoning: VLM-specific questions grounded in annotated images of edges, asking about edge distance, edge-pair relationship, or whether one edge lies on the inside-facing or outside-facing side of another.

B.2 QA Examples

We visualize additional example QA pairs in Fig. 12. Note that, for each evaluation modality, only the corresponding input representation (text, image, or both) is provided to the model query.

B.3 Prompt template for QA

The prompt used to enforce a consistent answer format is shown in Fig. 13. This instruction is appended to all questions across every modality. For the image-only modality, an additional prompt in Fig. B.3 is appended to ensure that answers are generated using the correct unit system.

C Additional Experimental Setup and Results

C.1 Experimental Setup

We set the maximum generation length to 16,384 tokens and the temperature to 0.7 for all experiments, except for the hotspot detection and VLM-specific evaluation tasks, where the maximum token length is reduced to 4,096. In addition, because InternVL3.5 supports a relatively smaller context window compared to other evaluated models, we also limit the maximum token length to 4,096 for the DRC task. Predictions exceeding the maximum token limit are treated as incorrect. For exact-match tasks, answers without the <answer> tag and truncated outputs caused by the token limit are also

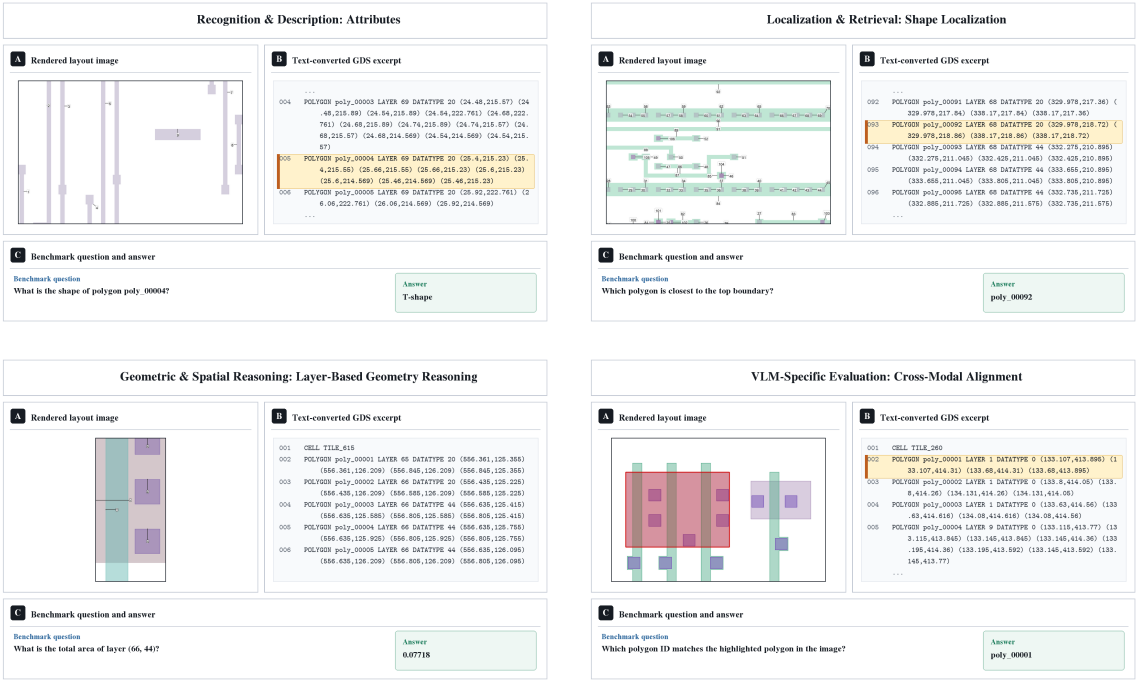


Figure 12: Additional example question-answer pairs from CHIPLAYOUTBENCH.

Table 5: Number of invalid predictions across different models.

Model	Mode	# of invalid predictions
Qwen3.5-27B	Text	230
	Image	898
	Both	314
Qwen3.6-27B	Text	104
	Image	104
	Both	263
Qwen3.5-397B-A17B	Text	383
	Image	1183
	Both	517
Gemma4-31B-Instruct	Text	49
	Image	14
	Both	33
InternVL3.5-241B-A28B	Text	63
	Image	3
	Both	64
GLM4.6V	Text	52
	Image	0
	Both	111

considered incorrect. For regression tasks, invalid or missing numerical outputs cannot be used to compute errors; therefore, such predictions are excluded from the regression error calculation in the main results table. The number of invalid predictions is separately reported in Table 5.

C.2 Qualitative Results

We visualize additional qualitative examples with correct predictions in Fig. 15, Fig. 16, and Fig. 17, and failure cases in Fig. 18, Fig. 19, and Fig. 20. The models generally produce correct answers for basic layout understanding tasks (Fig. 16 and Fig. 17) by sequentially reasoning over individual polygons and their geometric relationships. However, models struggle to infer exact semantic shapes from the GDS text representation alone (Fig. 18 and Fig. 20). In addition, the example in Fig. 19 demonstrates that models can still fail on relatively simple geometric reasoning tasks even in the text-only setting, indicating limitations in precise structural understanding from sequential layout representations.

We further visualize example outputs for the image-only modality in Fig. 21 and Fig. 22, the multimodal setting in Fig. 23 and Fig. 24, and VLM-specific tasks in Fig. 25 and Fig. 26. In the image-only setting, the model can correctly identify target shapes when the number of polygons is relatively small, but fails to estimate precise geometric distances using pixel-level information alone, as expected. For connected-component reasoning tasks, performance improves when both textual and visual modalities are jointly utilized, suggesting that multimodal information can provide complementary structural cues. However, the failure case in Fig. 24 shows that the reasoning process does not meaningfully reference the image modality, further indicating weak grounding between textual and visual representations.

GDS Layout Data:
{gds_content}

Question:
{question}

OUTPUT FORMAT INSTRUCTIONS (CRITICAL - READ CAREFULLY):

You MUST format your response exactly as shown below:

Reasoning: [Your step-by-step analysis goes here]

Answer: [Your final answer goes here]

<answer>YOUR_ANSWER</answer>

YOUR ANSWER TYPE: {type_desc}

FEW-SHOT EXAMPLES:
{examples_text}

CRITICAL RULES:

1. Your response MUST end with <answer>...</answer> on a separate line.
2. For lists: comma-separated IDs only (poly_00001, poly_00002, poly_00003)
3. NO backticks, NO brackets, NO quotes inside <answer> tags.
4. NO explanation inside <answer> tags — just the answer value.
5. Responses without <answer> tags will be marked WRONG.

END your response with:

<answer>YOUR_ANSWER</answer>

Answer Type Examples:

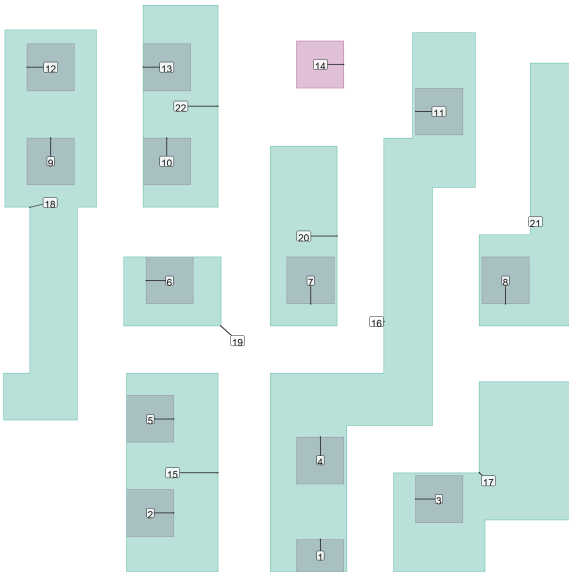
Type	Description	Example
integer	Count (integer number)	<answer>5</answer>
float	Floating point number	<answer>0.006912</answer>
polygon_id	Single polygon ID	<answer>poly_00017</answer>
list	List of polygon IDs	<answer>poly_00001, poly_00002</answer>
boolean	Yes or No	<answer>yes</answer>
location	Location identifier	<answer>poly_00017</answer>
string	Text answer	<answer>your_answer</answer>

Figure 13: Prompt template used to enforce a structured answer format across all evaluation settings.

UNIT INSTRUCTION (CRITICAL FOR NUMERIC ANSWERS):

The image scale is given in nm/pixel, but benchmark numeric answers use micrometers.
Use $1\ \mathrm{nm} = 0.001\ \mu\mathrm{m}$.
Distances and coordinate values should be reported in $\mu\mathrm{m}$, and areas should be reported in $\mu\mathrm{m}^2$.
When measuring from pixels, multiply pixel distances by 0.001 for $\mu\mathrm{m}$, and multiply pixel areas by $(0.001)^2$ for $\mu\mathrm{m}^2$.

Figure 14: Additional prompt used for unit clarification in the image-only modality.



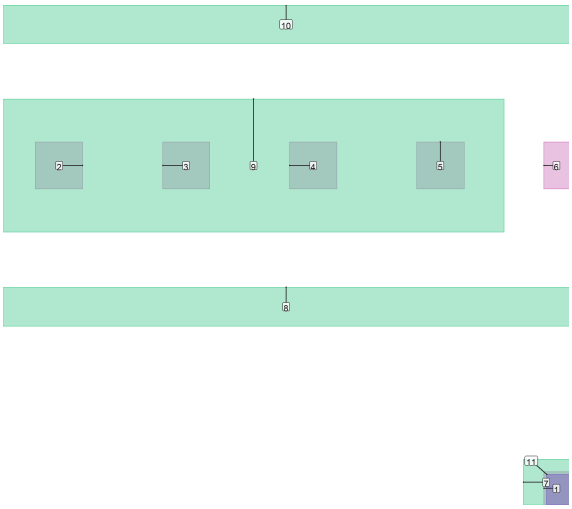
Question: Which polygons belong to the same connected component as polygon poly_00010?
Clarification: Polygons in the same connected component as polygon X means all polygons reachable from polygon X through a chain of overlaps or shared boundaries. The answer includes polygon X itself.

Response:
Reasoning: First, determine the bounding box of poly_00010: x ranges from 97.155 to 97.325, and y ranges from 442.545 to 442.715. Next, check all other polygons for 2D overlaps or shared boundaries. Poly_00022 (LAYER 67) spans x=[97.155, 97.425] and y=[442.465, 443.195], which completely contains poly_00010, creating a direct overlap. Poly_00013 (LAYER 66) spans x=[97.155, 97.325] and y=[442.885, 443.055], which is also fully contained within poly_00022, linking it to the component via poly_00022. All other polygons were checked for intersection with these three; none overlap due to clear gaps in either the x or y dimensions (e.g., poly_00019 and poly_00006 end at y=442.285, leaving a 0.18 gap below poly_00022; poly_00009 and poly_00018 end at x≈226496.985, leaving a gap to the left; poly_00014 and poly_00020 start at x≈226597.615, leaving a gap to the right). Thus, the connected component contains only these three polygons.

Answer: poly_00010, poly_00013, poly_00022
<answer>poly_00010, poly_00013, poly_00022</answer>

Answer by VLM: `poly_00010, poly_00013, poly_00022`
Ground Truth: `poly_00010, poly_00013, poly_00022`

Figure 15: Sample output from Qwen3.6-27B in the text-only modality with a correct prediction. The layout image is additionally provided for visualization and answer verification purposes only.



Question: How many square shapes are in the layout?
Clarification: `square` refers to the exact benchmark shape label, not a loose visual resemblance. Approximate resemblance alone is not sufficient for this label. Rotation, reflection, and stretched or squeezed aspect ratios are allowed as long as the same named-shape skeleton is preserved.

Response:

Reasoning:
I will analyze each polygon to determine if it is a square by checking if its width and height are equal.

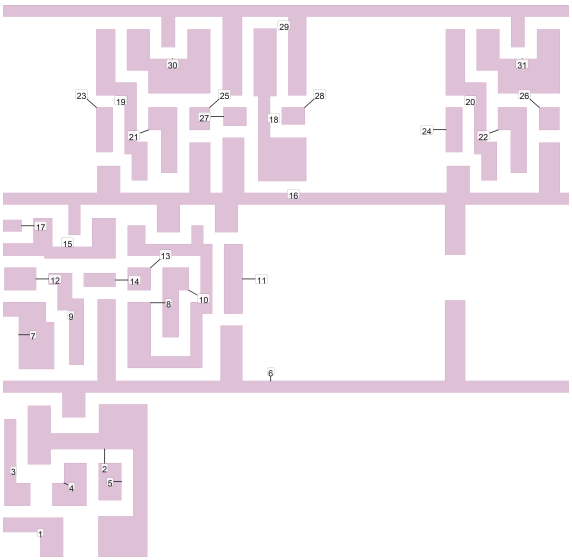
1. poly_00001: (172.185, 192.015) to (172.275, 191.894). Width = 172.275 - 172.185 = 0.09. Height = 192.015 - 191.894 = 0.121. Not a square.
2. poly_00002: (170.345, 193.035) to (170.515, 193.205). Width = 170.515 - 170.345 = 0.17. Height = 193.205 - 193.035 = 0.17. This is a square.
3. poly_00003: (170.805, 193.035) to (170.975, 193.205). Width = 170.975 - 170.805 = 0.17. Height = 193.205 - 193.035 = 0.17. This is a square.
4. poly_00004: (171.265, 193.035) to (171.435, 193.205). Width = 171.435 - 171.265 = 0.17. Height = 193.205 - 193.035 = 0.17. This is a square.
5. poly_00005: (171.725, 193.035) to (171.895, 193.205). Width = 171.895 - 171.725 = 0.17. Height = 193.205 - 193.035 = 0.17. This is a square.
6. poly_00006: (172.185, 193.035) to (172.275, 193.205). Width = 172.275 - 172.185 = 0.09. Height = 193.205 - 193.035 = 0.17. Not a square.
7. poly_00007: (172.11, 191.894) to (172.275, 192.06). Width = 172.275 - 172.11 = 0.165. Height = 192.06 - 191.894 = 0.166. Close, but not exactly equal (0.165 vs 0.166).
8. poly_00008: (170.227, 192.54) to (172.275, 192.68). Width = 2.048, Height = 0.14. Not a square.
9. poly_00009: (170.227, 192.88) to (172.04, 193.36). Width = 1.813, Height = 0.48. Not a square.
10. poly_00010: (170.227, 193.56) to (172.275, 193.7). Width = 2.048, Height = 0.14. Not a square.
11. poly_00011: (172.195, 191.894) to (172.275, 192.005). Width = 0.08, Height = 0.111. Not a square.

Counting the squares: poly_00002, poly_00003, poly_00004, poly_00005.

Answer: 4
<answer>4</answer>

Answer by VLM: 4
Ground Truth: 4

Figure 16: Sample output from Gemma4-31B-Instruct in the text-only modality with a correct prediction. The layout images are additionally provided for visualization and answer verification purposes only.



Question: Which polygon has the smallest area?

Response:
To find the smallest area, we calculate the area of each polygon based on the provided coordinates. Since these are all rectilinear polygons, we can use the shoelace formula or break them into rectangles.

- poly_00001: (328.055-327.186) * (563.865-563.655) + (328.055-327.725) * (563.295-563.655) = 0.869 * 0.21 + 0.33 * -0.36 = 0.18249 - 0.1188 (incorrect) -> Rectangles: (0.869 * 0.21) - (0.33 * 0.36) = 0.18249 - 0.1188 = 0.06369
- poly_00002: Complex shape. (327.875-327.545) * (565.485-564.635) + ... ≈ 2248 0.33 * 0.85 + 0.7 * 0.42 + ... ≈2248 0.57
- poly_00003: (327.375-327.205) * (565.29-564.365) + (327.58-327.375) * (564.365-564.035) = 0.17 * 0.925 + 0.205 * 0.33 = 0.15725 + 0.06765 = 0.2249
- poly_00004: (328.07-327.9) * (564.655-564.365) + (328.395-328.07) * (564.655-564.035) = 0.17 * 0.29 + 0.325 * 0.62 = 0.0493 + 0.2015 = 0.2508
- poly_00005: (328.9-328.57) * (564.655-564.115) = 0.33 * 0.54 = 0.1782
- poly_00006: Large complex shape. Area ≈2248 2-3 units.
- poly_00007: (327.805-327.186) * (566.985-566.775) + (327.925-327.805) * (566.695-566.015) = 0.619 * 0.21 + 0.12 * 0.68 = 0.12999 + 0.0816 = 0.21159
- poly_00008: Complex shape. Area ≈2248 0.5 units.
- poly_00009: (328.185-327.845) * (567.405-567.235) + (328.355-328.185) * (567.235-566.075) = 0.34 * 0.17 + 0.17 * 1.16 = 0.0578 + 0.1972 = 0.255
- poly_00010: (329.875-329.495) * (567.485-566.475) - (329.875-329.73) * (567.485-567.155) = 0.38 * 1.01 - 0.145 * 0.33 = 0.3838 - 0.04785 = 0.33595
- poly_00011: (330.655-330.385) * (567.825-566.815) = 0.27 * 1.01 = 0.2727
- poly_00012: (327.665-327.205) * (567.485-567.155) = 0.46 * 0.33 = 0.1518
- poly_00013: (329.325-328.99) * (567.485-567.155) = 0.335 * 0.33 = 0.11055
- poly_00014: (328.815-328.355) * (567.405-567.205) = 0.46 * 0.2 = 0.092
- poly_00015: (327.625-327.186) * (567.835-567.655) + (328.815-327.625) * (568.2-567.615) = 0.439 * 0.18 + 1.19 * 0.585 = 0.07902 + 0.69615 = 0.77517
- poly_00016: Large complex shape. Area > 1 unit.
- poly_00017: (327.455-327.186) * (568.175-568.005) = 0.269 * 0.17 = 0.04573
- poly_00018: (331.145-330.815) * (570.945-569.965) - (331.145-331.05) * (569.965-569.365) + ... ≈2248 0.33 * 0.98 = 0.3234
- poly_00019: (328.81-328.535) * (570.935-569.975) - (329.12-328.81) * (570.165-569.305) + ... ≈2248 0.275 * 0.96 = 0.264
- poly_00020: Similar to poly_00019 ≈2248 0.264
- poly_00021: (329.705-329.29) * (569.805-568.855) - (329.705-329.475) * (569.805-569.475) = 0.415 * 0.95 - 0.23 * 0.33 = 0.39425 - 0.0759 = 0.31835
- poly_00022: Similar to poly_00021 ≈2248 0.31835
- poly_00023: (328.775-328.535) * (569.805-569.155) = 0.24 * 0.65 = 0.156
- poly_00024: (333.835-333.595) * (569.805-569.155) = 0.24 * 0.65 = 0.156
- poly_00025: (330.18-329.885) * (569.805-569.475) = 0.295 * 0.33 = 0.09735
- poly_00026: (335.24-334.945) * (569.805-569.475) = 0.295 * 0.33 = 0.09735
- poly_00027: (330.71-330.375) * (569.805-569.535) = 0.335 * 0.27 = 0.09045
- poly_00028: (331.555-331.22) * (569.805-569.555) = 0.335 * 0.25 = 0.08375
- poly_00029: Complex shape. Area ≈2248 0.3 units.
- poly_00030: (329.31-328.98) * (570.935-570.335) - (329.855-329.31) * (570.505-570.335) + ... ≈2248 0.33 * 0.6 = 0.198
- poly_00031: Similar to poly_00030 ≈2248 0.198

Comparing the calculated areas:
poly_00017: 0.04573
poly_00001: 0.06369
poly_00028: 0.08375
poly_00027: 0.09045
poly_00014: 0.092

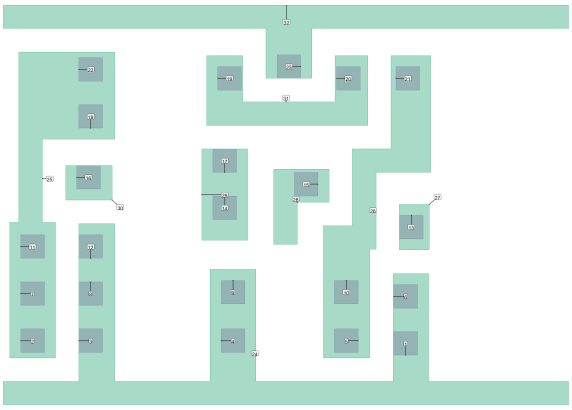
The smallest area is poly_00017.

Reasoning: I calculated the area of each polygon. poly_00017 is a simple rectangle with dimensions 0.269 x 0.17, resulting in an area of 0.04573, which is the smallest among all listed polygons.

Answer: poly_00017
<answer>poly_00017</answer>

Answer by VLM: `poly_00017`
Ground Truth: `poly_00017`

Figure 17: Additional sample output with a correct prediction.



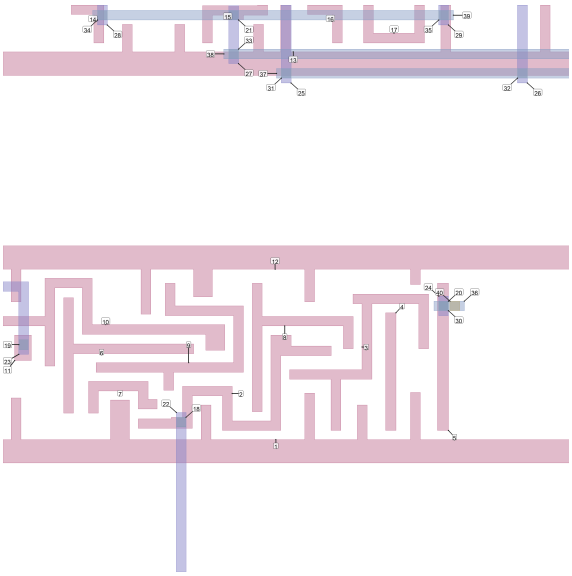
Question: Where is the U-shape-shaped polygon located?
Clarification: ‘U-shape‘ refers to the exact benchmark shape label, not a loose visual resemblance. Approximate resemblance alone is not sufficient for this label. Rotation, reflection, and stretched or squeezed aspect ratios are allowed as long as the same named-shape skeleton is preserved. This question is used only when exactly one polygon in the layout is labeled ‘U-shape‘ under the benchmark shape taxonomy. Answer with the polygon ID.

Response:

Reasoning: By examining the vertex coordinates of each polygon, poly_00025 clearly traces a U-shaped geometry. Its 10 vertices form a continuous path that creates a vertical back segment on the left with two horizontal arms extending to the right, connected by an inner horizontal segment. This structure perfectly matches the standard benchmark definition of a U-shape (allowing for rotation). Other polygons in the layout are either simple rectangles or more complex comb-like/multi-segment structures, making poly_00025 the unique and correct match.

Answer: poly_00025
<answer>poly_00025</answer>
Answer by VLM: `poly_00025`
Ground Truth: `poly_00031`

Figure 18: Sample output from Qwen3.6-27B in the text-only modality with an incorrect prediction. The layout image is additionally provided for visualization and answer verification purposes only.



Question: What is the shortest path between polygon poly_00013 and polygon poly_00028 via connected polygons?
Clarification: The shortest path is the unique minimum-hop sequence of polygons where consecutive polygons overlap or share a boundary. The answer includes both endpoint polygons.

Response: To find the shortest path between ‘poly_00013‘ and ‘poly_00028‘, we analyze the spatial relationships (overlaps or shared boundaries) between the polygons.

1. **Analyze ‘poly_00013‘**:
Layer 11, Bounding Box: X[274.306, 278.402], Y[552.915, 553.424].
This polygon acts as a base bar for several other shapes in the Y range 552.9 to 553.4.

2. **Analyze ‘poly_00028‘**:
Layer 13, Bounding Box: X[274.99, 275.06], Y[553.28, 553.424].
Coordinates: (274.99, 553.28), (274.99, 553.424), (275.06, 553.424), (275.06, 553.28).

3. **Check for Direct Connection**:
‘poly_00013‘ covers X from 274.306 to 278.402 and Y from 552.915 to 553.424.
‘poly_00028‘ is contained entirely within the bounding box of ‘poly_00013‘. Specifically, ‘poly_00028‘’s X range [274.99, 275.06] is within [274.306, 278.402] and its Y range [553.28, 553.424] is within [552.915, 553.424].
Since ‘poly_00028‘ is located within the area covered by ‘poly_00013‘, they overlap.

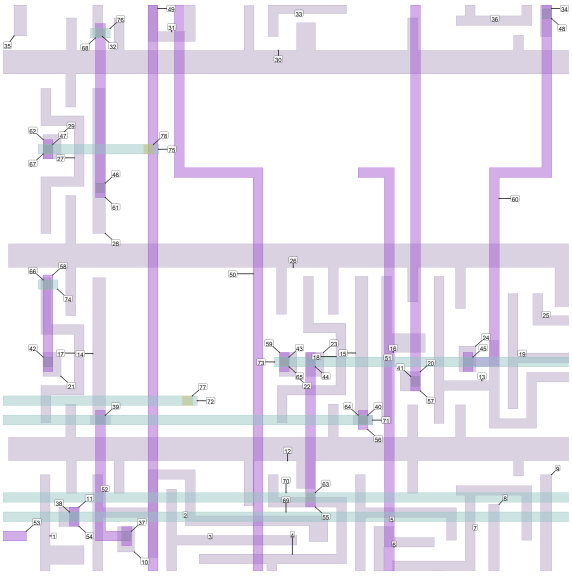
4. **Determine Shortest Path**:
Because ‘poly_00013‘ and ‘poly_00028‘ overlap directly, the shortest path consists of only these two polygons.

Reasoning: poly_00013 is a large polygon on layer 11 that spans the coordinates where poly_00028 (on layer 13) is located. Since they overlap, the shortest path is the direct connection between them.

Answer: poly_00013, poly_00028
<answer>poly_00013, poly_00028</answer>

Answer by VLM: `poly_00013, poly_00028`
Ground Truth: `poly_00013, poly_00039, poly_00028`

Figure 19: Sample output from Gemma4-31B-Instruct in the text-only modality with an incorrect prediction. The layout image is additionally provided for visualization and answer verification purposes only.



Question: Where is the U-shape-shaped polygon located?
Clarification: ‘U-shape‘ refers to the exact benchmark shape label, not a loose visual resemblance. Approximate resemblance alone is not sufficient for this label. Rotation, reflection, and stretched or squeezed aspect ratios are allowed as long as the same named-shape skeleton is preserved. This question is used only when exactly one polygon in the layout is labeled ‘U-shape‘ under the benchmark shape taxonomy. Answer with the polygon ID.

Response:

To identify the U-shaped polygon, we need to look for a polygon with a geometry that resembles a "U" (a concave shape with a base and two parallel arms).

1. **Analyze candidates for complex shapes:**
‘poly_00002‘: (12.01,590.615) -> (12.08,590.615) -> (12.08,590.375) -> (12.41,590.375) -> (12.41,590.65) -> (12.75,590.65) -> (12.75,590.315) -> (13.71,590.315) -> (13.71,590.13) -> (13.575,590.13) -> (13.575,590.245) -> (12.68,590.245) -> (12.68,590.58) -> (12.48,590.58) -> (12.48,590.015) -> (12.41,590.015) -> (12.41,590.305) -> (12.08,590.305) -> (12.08,589.917) -> (12.01,589.917). This is a highly complex, narrow, winding shape.
‘poly_00004‘: (12.78,589.97) -> (12.78,590.04) -> (13.775,590.04) -> (13.775,590.38) -> (13.28,590.38) -> (13.28,590.615) -> (13.35,590.615) -> (13.35,590.45) -> (13.845,590.45) -> (13.845,589.97) -> (13.34,589.97) -> (13.34,589.917) -> (13.27,589.917) -> (13.27,589.97). This is a complex shape with several turns.
‘poly_00007‘: (14.18,589.92) -> (14.18,589.99) -> (14.705,589.99) -> (14.705,590.465) -> (14.64,590.465) -> (14.64,590.535) -> (15.185,590.535) -> (15.185,590.14) -> (15.115,590.14) -> (15.115,590.465) -> (14.775,590.465) -> (14.775,589.92) -> (14.55,589.92) -> (14.55,589.917) -> (14.48,589.917) -> (14.48,589.92). This is another complex shape.
‘poly_00012‘: This is a large, complex boundary shape with multiple cut-outs (holes/indentations).
‘poly_00013‘: (14.48,590.985) -> (14.48,592.05) -> (14.55,592.05) -> (14.55,591.295) -> (14.88,591.295) -> (14.88,591.585) -> (14.95,591.585) -> (14.95,591.02) -> (15.15,591.02) -> (15.15,591.355) -> (15.456,591.355) -> (15.456,591.285) -> (15.22,591.285) -> (15.22,590.95) -> (14.88,590.95) -> (14.88,591.225) -> (14.55,591.225) -> (14.55,590.985). Looking at the coordinates: It goes up to 592.05, comes down to 591.295, goes right, goes up to 591.585, goes right, goes down to 591.02, goes right, goes up to 591.355, goes right, goes down to 591.285, goes left, goes down to 590.95, goes left, goes up to 591.225, goes left, goes down to 590.985. This creates a "W" or "M" like shape.
‘poly_00017‘: (11.635,591.26) -> (11.875,591.26) -> (11.875,591.63) -> (11.635,591.63) -> (11.635,592.04) -> (11.705,592.04) -> (11.705,591.7) -> (11.945,591.7) -> (11.945,591.19) -> (11.705,591.19) -> (11.705,590.99) -> (11.635,590.99). This is a narrow shape.
‘poly_00018‘: (13.345,591.26) -> (13.77,591.26) -> (13.77,591.64) -> (13.535,591.64) -> (13.535,592.05) -> (13.605,592.05) -> (13.605,591.71) -> (13.84,591.71) -> (13.84,591.19) -> (13.415,591.19) -> (13.415,590.99) -> (13.345,590.99). Similar to poly_00017.
‘poly_00026‘: A large boundary shape with multiple indentations.
‘poly_00030‘: Another large boundary shape with multiple indentations.

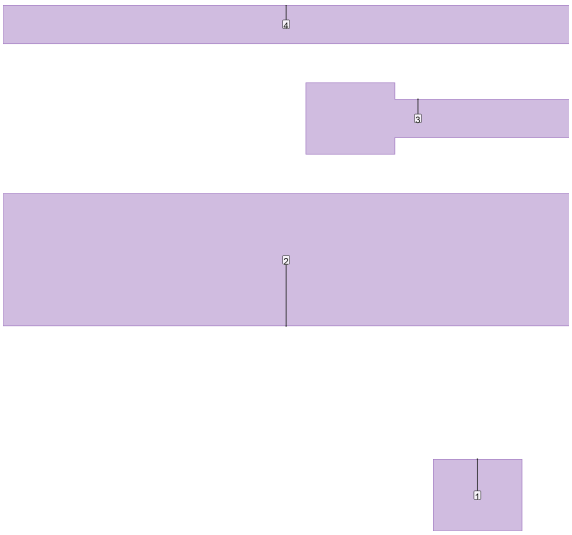
2. **Re-evaluating ‘poly_00004‘ and ‘poly_00007‘:**
‘poly_00004‘: x range [12.78, 13.845], y range [589.917, 590.615]. Points: (12.78, 589.97) -> (12.78, 590.04) -> (13.775, 590.04) -> (13.775, 590.38) -> (13.28, 590.38) -> (13.28, 590.615) -> (13.35, 590.615) -> (13.35, 590.45) -> (13.845, 590.45) -> (13.845, 589.97) -> (13.34, 589.97) -> (13.34, 589.917) -> (13.27, 589.917) -> (13.27, 589.97).
‘poly_00007‘: x range [14.18, 15.185], y range [589.917, 590.535]. Points: (14.18, 589.92) -> (14.18, 589.99) -> (14.705, 589.99) -> (14.705, 590.465) -> (14.64, 590.465) -> (14.64, 590.535) -> (15.185, 590.535) -> (15.185, 590.14) -> (15.115, 590.14) -> (15.115, 590.465) -> (14.775, 590.465) -> (14.775, 589.92) -> (14.55, 589.92) -> (14.55, 589.917) -> (14.48, 589.917) -> (14.48, 589.92).
Wait, let’s look at ‘poly_00007‘’s y-coords: 589.92 -> 589.99 -> 589.99 -> 590.465 -> 590.465 -> 590.535 -> 590.535 -> 590.14 -> 590.14 -> 590.465 -> 590.465 -> 589.92.
The x-coords for the "arms": (14.64 to 14.705) and (15.115 to 15.185) are the vertical segments. The base is at the top (590.465 to 590.535) or bottom.
Looking at ‘poly_00007‘:
Left arm: x=14.64-14.705, y from 589.99 to 590.465.
Right arm: x=15.115-15.185, y from 590.14 to 590.465.
Base: y=590.465 to 590.535, x from 14.64 to 15.185.
This is a U-shape (rotated).

Reasoning: Analyzing the coordinates of poly_00007, it describes a shape with two vertical arms and a connecting base, which matches the U-shape benchmark.

Answer: poly_00007
<answer>poly_00007</answer>

Answer by VLM: `poly_00007`
Ground Truth: `poly_00031`

Figure 20: Additional sample output with an incorrect prediction.

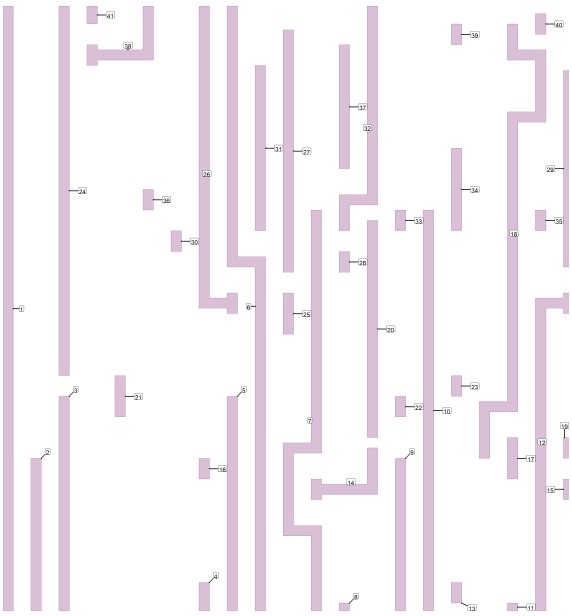


Question: How many T-shape shapes are in the layout?
Clarification: ‘T-shape‘ refers to the exact benchmark shape label, not a loose visual resemblance. If the polygon only approximately looks like ‘T-shape‘ because it has extra small attachments or steps, it should be treated as ‘T-like‘ instead. Rotation, reflection, and stretched or squeezed aspect ratios are allowed as long as the same named-shape skeleton is preserved. Image-only note: the annotated image resolution is 2048x1900 pixels. Use [400.553, 169.56] as the layout top-left reference coordinate. Top-left polygon poly_00004 starts at [400.553, 169.56]. The annotated image is rendered at 1 nm /pixel. Pixel size is 1 nm /pixel in x and 1 nm /pixel in y. The annotated image is rerendered directly from the layout geometry with zero extra padding, so the image boundary matches the layout bounding box at the recorded physical scale. Use the provided top-left reference coordinate together with the recorded pixel size metadata when converting between pixels and layout coordinates. Layer colors: (68, 20) uses RGB(155, 113, 190).

Response:
Reasoning:
1. I need to identify "T-shape" shapes in the layout.
2. I examine the provided image and the numbered polygons.
3. Polygon 4 is a simple rectangle (bar).
4. Polygon 3 consists of a vertical rectangular section and a horizontal rectangular section meeting at a right angle, forming a T-shape (rotated 90 degrees).
5. Polygon 2 is a simple rectangle (bar).
6. Polygon 1 is a simple rectangle (square).
7. Only polygon 3 matches the "T-shape" benchmark description.

Answer: 1
<answer>1</answer>
Answer by VLM: 1
Ground Truth: 1

Figure 21: Sample output from Gemma4-31B-Instruct in the image-only modality with a correct prediction.

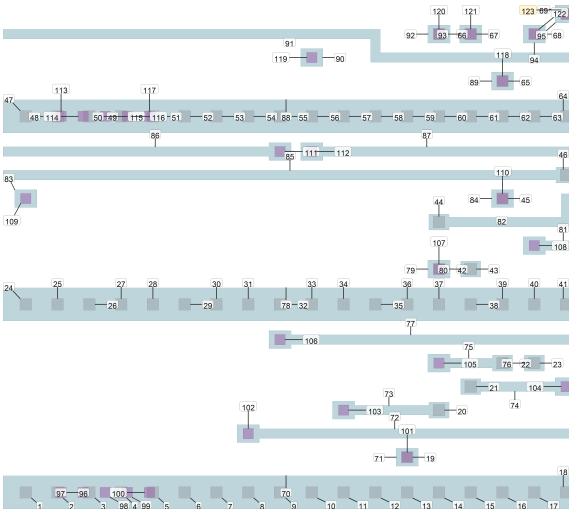


Question: What is the area of polygon poly_00039 after expanding its right-side edges by distance 0.5?
Clarification: Clarification: This question expands only the polygon edges that face the named direction by distance d. The reported area is computed from the exact expanded polygon geometry, not from the bounding box. Image-only note: the annotated image resolution is 3837x4096 pixels. Use [288.29, 52.622] as the layout top-left reference coordinate. Top-left polygon poly_00001 starts at [288.29, 52.622]. The annotated image is rendered at 1 nm/pixel. Pixel size is 1 nm/pixel in x and 1 nm/pixel in y. The annotated image is rerendered directly from the layout geometry with zero extra padding, so the image boundary matches the layout bounding box at the recorded physical scale. Use the provided top-left reference coordinate together with the recorded pixel size metadata when converting between pixels and layout coordinates. Layer colors: (13, 0) uses RGB(176, 122, 169).

Response:
Reasoning:
1. **Identify poly_00039**: Locate the polygon labeled "39" in the image. It is a small rectangular block in the top right area.
2. **Determine dimensions of poly_00039**:
Top-left corner (approximate pixels): [3115, 115]
Bottom-right corner (approximate pixels): [3215, 175]
Width = 3215 - 3115 = 100 pixels.
Height = 175 - 115 = 60 pixels.
Converting to micrometers (1 nm/pixel = 0.001 um/pixel):
Width = 100 * 0.001 = 0.1 um.
Height = 60 * 0.001 = 0.06 um.
3. **Expand right-side edges**:
The expansion distance is 0.5 um.
The right-side edge is expanded by 0.5 um.
New Width = Original Width + 0.5 um = 0.1 um + 0.5 um = 0.6 um.
The height remains unchanged = 0.06 um.
4. **Calculate the new area**:
Area = New Width * Height = 0.6 $\mu\mathrm{m}$ * 0.06 $\mu\mathrm{m}$ = 0.036 $\mu\mathrm{m}^2$.

Answer: 0.036
<answer>0.036</answer>
Answer by VLM: 0.036
Ground Truth: 0.0798

Figure 22: Sample output from Gemma4-31B-Instruct in the image-only modality with an incorrect prediction.



Question: Which polygons belong to the same connected component as polygon poly_00086?
Clarification: Polygons in the same connected component as polygon X means all polygons reachable from polygon X through a chain of overlaps or shared boundaries. The answer includes polygon X itself.

Response:
To determine the connected component containing ‘poly_00086‘, we analyze the coordinates and layers of the polygons to find overlaps or shared boundaries.

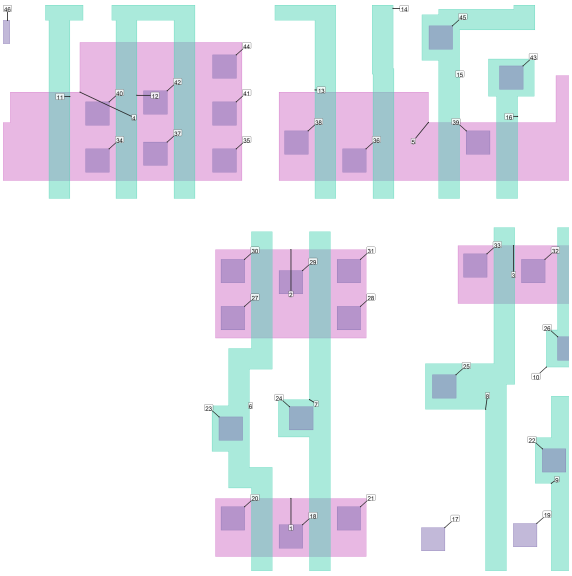
1. **Analyze ‘poly_00086‘**:
‘poly_00086‘ (LAYER 68, DATATYPE 20): (543.62, 287.74) to (547.47, 287.88) - a horizontal bar.
2. **Identify Overlaps/Connections**:
‘poly_00086‘ shares a boundary or overlaps with:
‘poly_00085‘ (LAYER 68): (543.62, 287.4) to (551.812, 287.585). (Wait, checking coordinates: 287.74 is above 287.585. No overlap).
‘poly_00087‘ (LAYER 68): (547.93, 287.94) to (551.812, 287.88). (No overlap).
‘poly_00088‘ (LAYER 68): (543.62, 288.08) to (551.812, 288.56). (No overlap).
Looking at the image, ‘poly_00086‘ is part of a larger structure. Let’s look at polygons that might be on different layers but at the same location.
‘poly_00086‘ (L68) is at Y range [287.74, 287.88].
‘poly_00051‘ (L67) is at (545.705, 288.235) to (545.875, 288.405). (No overlap).
Let’s check the ‘poly_00047‘ to ‘poly_00064‘ group (L67). These are small squares.
‘poly_00050‘ (L67): (545.245, 288.235) to (545.415, 288.405). (No overlap).
Wait, let’s check ‘poly_00086‘ again. It is a bar. Is it connected to ‘poly_00085‘ or ‘poly_00087‘?
‘poly_00085‘ is at Y [287.355, 287.585].
‘poly_00086‘ is at Y [287.68, 287.94].
‘poly_00087‘ is at Y [287.68, 287.94].
Looking at the coordinates for ‘poly_00086‘: ‘(543.62, 287.74) (543.62, 287.88) (547.47, 287.88) (547.47, 287.94) (547.79, 287.94) (547.79, 287.68) (547.47, 287.68) (547.47, 287.74)‘.
Looking at the coordinates for ‘poly_00087‘: ‘(547.93, 287.94) (548.25, 287.94) (548.25, 287.88) (551.812, 287.88) (551.812, 287.74) (548.25, 287.74) (548.25, 287.68) (547.93, 287.68)‘.
These two are nearly adjacent but not overlapping.
However, ‘poly_00086‘ and ‘poly_00087‘ both overlap with ‘poly_00111‘ (L68, DATATYPE 44) and ‘poly_00112‘ (L68, DATATYPE 44)?
‘poly_00111‘: (547.555, 287.885) to (547.705, 287.735).
‘poly_00111‘’s X range is [547.555, 547.705] and Y range is [287.735, 287.885].
‘poly_00086‘’s X range is [543.62, 547.79] and Y range is [287.68, 287.94].
‘poly_00111‘ is inside ‘poly_00086‘.
‘poly_00112‘: (548.015, 287.885) to (548.165, 287.735).
‘poly_00112‘’s X range is [548.015, 548.165] and Y range is [287.735, 287.885].
‘poly_00087‘’s X range is [547.93, 551.812] and Y range is [287.68, 287.94].
‘poly_00112‘ is inside ‘poly_00087‘.
Now check for other connections. ‘poly_00085‘ (Y: 287.355-287.585) and ‘poly_00086‘ (Y: 287.68-287.94) do not overlap.
‘poly_00088‘ (Y: 288.08-288.56) and ‘poly_00086‘ (Y: 287.68-287.94) do not overlap.
Looking at the image, ‘poly_00086‘ and ‘poly_00087‘ are parts of the same horizontal line structure.
Wait, the question asks for the connected component. In the image, these horizontal lines are separated by gaps.
‘poly_00086‘ is connected to ‘poly_00111‘. ‘poly_00087‘ is connected to ‘poly_00112‘.
Is there anything connecting ‘poly_00086‘ to ‘poly_00087‘?
Check ‘poly_00086‘ (X: 543.62 to 547.79) and ‘poly_00087‘ (X: 547.93 to 551.812). Gap of 0.14.
Check other polygons. ‘poly_00085‘ is a long bar from 543.62 to 551.812.
‘poly_00085‘ Y range: [287.355, 287.585]. ‘poly_00086‘ Y range: [287.68, 287.94]. No overlap.
‘poly_00088‘ Y range: [288.08, 288.56]. No overlap.
Looking at the image, ‘poly_00086‘ is a segment of a line. It is connected to the small purple square ‘poly_00111‘.

Reasoning: ‘poly_00086‘ is a polygon on layer 68. Its coordinates overlap with ‘poly_00111‘ (layer 68), which is a small square located within the boundaries of ‘poly_00086‘. No other polygons overlap with ‘poly_00086‘ or ‘poly_00111‘.

Answer: poly_00086, poly_00111

<answer>poly_00086, poly_00111</answer>
Answer by VLM: `poly_00086, poly_00111`
Ground Truth: `poly_00086, poly_00111`

Figure 23: Sample output from Gemma4-31B-Instruct using both modalities with a correct prediction.



Question: Which polygons belong to the same connected component as polygon poly_00009?
Clarification: Polygons in the same connected component as polygon X means all polygons reachable from polygon X through a chain of overlaps or shared boundaries. The answer includes polygon X itself.

Response:
"To determine the connected component containing ‘poly_00009‘, we examine the coordinates and overlaps of the polygons.

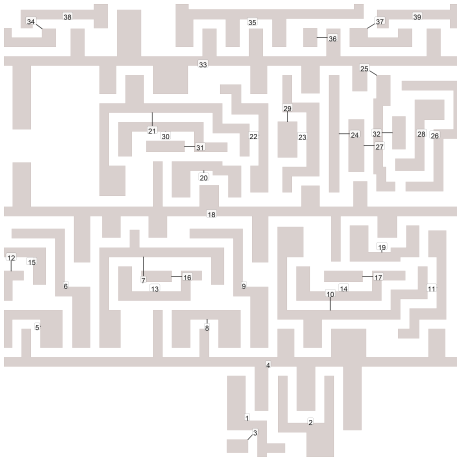
1. **Analyze poly_00009**:
- ‘poly_00009‘: (49.83, 57.885) to (50.073, 58.515). Layer 66.
2. **Check for overlaps/shared boundaries with poly_00009**:
- ‘poly_00010‘: (49.91, 58.725) to (50.073, 59.735). It shares a boundary or is very close to ‘poly_00009‘ (x-range 49.91-50.073 overlaps 49.83-50.073).
- ‘poly_00008‘: (49.035, 58.42) to (49.68, 59.735).
- ‘poly_00007‘: (47.97, 58.22) to (48.345, 59.705).
- ‘poly_00006‘: (47.49, 58.115) to (47.925, 59.705).
- Looking at the GDS layout, polygons ‘poly_00006‘ through ‘poly_00010‘ represent vertical routing lines/segments on Layer 66.
- ‘poly_00009‘ (x: 49.83-50.073, y: 57.885-58.515) and ‘poly_00010‘ (x: 49.91-50.073, y: 58.725-59.735) are aligned vertically.
- Checking Layer 65 polygons (horizontal) that might connect these:
- ‘poly_00001‘: (47.515, 57.355) to (48.605, 57.775).
- ‘poly_00002‘: (47.515, 58.935) to (48.605, 59.575).
- ‘poly_00003‘: (49.27, 59.185) to (50.073, 59.605).
- ‘poly_00003‘ (x: 49.27-50.073, y: 59.185-59.605) overlaps ‘poly_00010‘ (x: 49.91-50.073, y: 58.725-59.735).
- ‘poly_00003‘ also overlaps ‘poly_00008‘ (x: 49.035-49.68, y: 58.42-59.735) because the x-ranges (49.27-50.073 and 49.035-49.68) and y-ranges overlap.
- ‘poly_00008‘ overlaps ‘poly_00007‘ via horizontal connections if they exist, but checking the coordinates, they are distinct vertical bars.
- However, looking at the connectivity of the v̈ertical bars(poly_00006, 00007, 00008, 00009, 00010), they are often connected by horizontal bars (poly_00001, 00002, 00003, 00004, 00005).
- ‘poly_00009‘ is connected to ‘poly_00010‘ (vertically aligned/touching).
- ‘poly_00010‘ is connected to ‘poly_00003‘ (overlap).
- ‘poly_00003‘ is connected to ‘poly_00008‘ (overlap).
- ‘poly_00008‘ is connected to ‘poly_00002‘ (overlap).
- ‘poly_00002‘ is connected to ‘poly_00007‘ (overlap).
- ‘poly_00007‘ is connected to ‘poly_00006‘ (via ‘poly_00002‘ or others).
- ‘poly_00006‘ is connected to ‘poly_00001‘ (overlap).
- Thus, ‘poly_00001‘ through ‘poly_00010‘ form a connected cluster.
- Further, ‘poly_00011‘ through ‘poly_00016‘ are vertical bars in a different horizontal row (y 60-61).
- ‘poly_00004‘ and ‘poly_00005‘ are horizontal bars (y 60-61) that connect ‘poly_00011‘ through ‘poly_00016‘.
- ‘poly_00005‘ (x: 47.975-50.073, y: 60.075-60.835) overlaps ‘poly_00010‘ (x: 49.91-50.073, y: 58.725-59.735)? No, the y-ranges (60.075-60.835 and 58.725-59.735) do not overlap.
- Wait, checking ‘poly_00010‘ again: y is 58.725 to 59.735. ‘poly_00005‘ y is 60.075 to 60.835. They do not touch.
- Checking other connections: ‘poly_00003‘ (y: 59.185-59.605). ‘poly_00005‘ (y: 60.075-60.835). No overlap.
- The vertical bars ‘poly_00006-00010‘ and horizontal bars ‘poly_00001-00003‘ are connected. The vertical bars ‘poly_00011-00016‘ and horizontal bars ‘poly_00004-00005‘ are connected.
- Is there any link between these two groups?
- Check ‘poly_00010‘ (y: 58.725-59.735) and ‘poly_00016‘ (y: 59.945-60.955). No overlap.
- The polygons ‘poly_00017‘ through ‘poly_00046‘ are small contact/via polygons. Many of these overlap the larger layout polygons.
- ‘poly_00022‘ (49.88, 58.135) overlaps ‘poly_00009‘ (49.83-50.073, 57.885-58.515).
- ‘poly_00026‘ (49.99, 58.945) overlaps ‘poly_00010‘.
- ‘poly_00032‘ (49.73, 59.505) overlaps ‘poly_00003‘.
- Since ‘poly_00009‘ is connected to ‘poly_00001-00010‘ and ‘poly_00001-00003‘, and the via polygons ‘poly_00017-00033‘ overlap these, they are all part of the same component.

Reasoning: poly_00009 is connected to poly_00010 (vertical alignment/touching), which is connected to poly_00003 (overlap). poly_00003 is connected to poly_00008, which is connected to poly_00002, and so on. This forms a connected network including poly_00001 through poly_00010 and the small via polygons (poly_00017-poly_00033) that overlap them.

Answer: poly_00001, poly_00002, poly_00003, poly_00006, poly_00007, poly_00008, poly_00009, poly_00010, poly_00017, poly_00018, poly_00019, poly_00020, poly_00021, poly_00022, poly_00023, poly_00024, poly_00025, poly_00026, poly_00027, poly_00028, poly_00029, poly_00030, poly_00031, poly_00032, poly_00033

<answer>poly_00001, poly_00002, poly_00003, poly_00006, poly_00007, poly_00008, poly_00009, poly_00010, poly_00017, poly_00018, poly_00019, poly_00020, poly_00021, poly_00022, poly_00023, poly_00024, poly_00025, poly_00026, poly_00027, poly_00028, poly_00029, poly_00030, poly_00031, poly_00032, poly_00033</answer>
Answer by VLM: `poly_00009, poly_00022`
Ground Truth: `poly_00001, poly_00002, poly_00003, poly_00006, poly_00007, poly_00008, poly_00009, poly_00010, poly_00017, poly_00018, poly_00019, poly_00020, poly_00021, poly_00022, poly_00023, poly_00024, poly_00025, poly_00026, poly_00027, poly_00028, poly_00029, poly_00030, poly_00031, poly_00032, poly_00033`

Figure 24: Sample output from Gemma4-31B-Instruct using both modalities with an incorrect prediction.



Question: The image shows a single polygon cropped from the GDS. Which polygon ID matches this polygon?
Clarification: Clarification: Exactly one answer choice has the same polygon shape as the cropped image.

Response:
The image shows a complex, multi-segmented shape consisting of a central horizontal bar with several vertical extensions (both upwards and downwards) and some small notches/cutouts.

Let's analyze the GDS data for a polygon that matches this complex structure:
1. **poly_00004**: Describes a horizontal bar from x=131.298 to 139.49, with several vertical segments (up to 98.855 and down to 96.685). It has notches and protrusions. 2. **poly_00018**: Describes a horizontal bar from x=131.298 to 139.49, with several vertical segments (up to 101.545 and down to 99.72). It also has notches and protrusions.
3. **poly_00033**: Describes a horizontal bar from x=131.298 to 139.49, with vertical segments (up to 104.285 and down to 102.125).

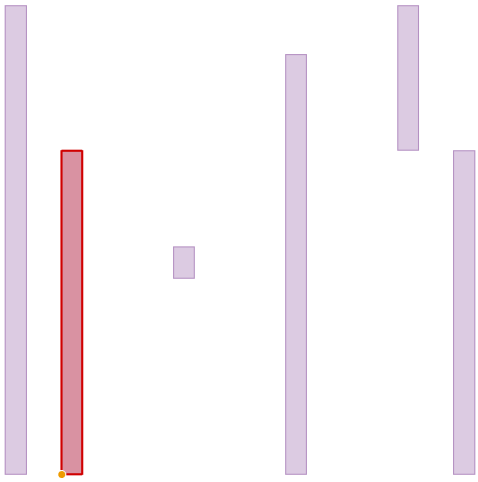
Comparing the visual image to the coordinates:
- The image shows a long horizontal line with vertical "teeth".
- Looking at 'poly_00018', the coordinates define a structure starting at (131.298, 100.555), going across to (139.49, 100.555), and having various vertical offsets like (133.995, 101.545) and (132.855, 99.72). This matches the "comb-like" or "barcode-like" structure seen in the image.
- 'poly_00004' is similar but shifted in Y.
- 'poly_00033' is also similar but shifted in Y.
- However, looking closely at the image, there are specific patterns of protrusions. The image has a tall protrusion on the far left, several smaller ones, and a larger one towards the middle-right.
- Let's check 'poly_00018' again: It has a protrusion at (131.785, 101.525) and (133.995, 101.545) and (137.615, 101.185), and downward segments at (132.855, 99.72), (136.075, 99.72), (137.385, 99.735). This matches the pattern of vertical lines seen in the cropped image.

Reasoning: The cropped image displays a single complex polygon with a horizontal baseline and multiple vertical extensions. Poly_00018 matches this geometry, including the specific locations and heights of the vertical bars and the gaps between them.

Answer: poly_00018
<answer>poly_00018</answer>
Answer by VLM: `poly_00018`
Ground Truth: `poly_00018`

Figure 25: Sample answer from Gemma4-31B-Instruct for the VLM-specific evaluation. The fully annotated image is additionally provided to illustrate the correspondence between polygon IDs and the predicted answer.

Question: What is the x-coordinate of the highlighted vertex in the image?



GDS Text

```
POLYGON poly\_00001 LAYER 13 DATATYPE 0
(320.78,257.533) (320.85,257.533)
(320.85,255.485) (320.78,255.485)

POLYGON poly\_00002 LAYER 13 DATATYPE 0
(320.97,255.485) (320.97,256.9)
(321.04,256.9) (321.04,255.485)

POLYGON poly\_00003 LAYER 13 DATATYPE 0
(321.73,255.485) (321.73,257.32)
(321.8,257.32) (321.8,255.485)

POLYGON poly\_00004 LAYER 13 DATATYPE 0
(322.3,255.485) (322.3,256.9)
(322.37,256.9) (322.37,255.485)

POLYGON poly\_00005 LAYER 13 DATATYPE 0
(321.35,256.34) (321.35,256.48)
(321.42,256.48) (321.42,256.34)

POLYGON poly\_00006 LAYER 13 DATATYPE 0
(322.11,256.9) (322.11,257.533)
(322.18,257.533) (322.18,256.9)
```

Response:
The image displays a layout with several vertical rectangles. A specific vertex is highlighted with an orange dot at the bottom-left corner of the second vertical rectangle from the left (the one outlined in red).

Looking at the provided GDS Layout Data, there are several polygons listed, but they appear to be very small features (micro-scale) and do not correspond to the large rectangular shapes shown in the image. However, the question asks for the x-coordinate of the highlighted vertex.

In typical layout analysis tasks of this nature, if the GDS data provided does not match the visual elements, we must look for clues or assume the question refers to the coordinates of the object in the provided data if applicable. However, the GDS data provided contains very small polygons (‘poly_00001‘ to ‘poly_00006‘) centered around x=321 and y=256.

The highlighted vertex is the bottom-left corner of the red rectangle.
Let’s re-examine the provided GDS data. The data lists points like (320.78, 257.533), (320.85, 257.533), etc. None of these describe large rectangles.

Wait, looking at the image again, the highlighted point is at the bottom left of the red rectangle. If we assume the coordinates provided in the GDS data are the only source of truth for coordinates in this problem, we should check if any of them align. However, the GDS data provided is clearly for a different set of objects (very small dots/lines) than the large blocks shown.

Actually, looking closer at the prompt’s structure, often in these specific tests, the provided GDS data is a distractor or refers to a different part of the chip, but the intended answer comes from a coordinate system mapped to the image. However, without a scale, I must check the GDS data for any pattern.

The provided polygons are:
poly_00001: x from 320.78 to 320.85
poly_00002: x from 320.97 to 321.04
poly_00003: x from 321.73 to 321.8
poly_00004: x from 322.3 to 322.37
poly_00005: x from 321.35 to 321.42
poly_00006: x from 322.11 to 322.18

Since no other coordinates are provided and the question asks for a specific x-coordinate, and the provided GDS data is the only numerical source, there might be a mistake in the provided data or I must find the intended value. Let’s re-read. The highlighted vertex is at the bottom left of the red rectangle. If the red rectangle is one of the polygons, its coordinates would be listed. None of these are rectangles of the size shown.

However, if I must provide a number from the data, and looking at the vertex’s relative position, if the red rectangle’s left edge is at 320.78 (the first x-coord listed), that would be a possibility. But usually, these problems have a direct mapping. Let’s look at the polygons again. They are all very thin.

Wait, looking at the image, the highlighted point is a single vertex. If the provided polygons are actually the "vertices" or "markers", poly_00001 starts at 320.78.

Reasoning: The highlighted vertex is at the bottom-left of the red rectangle. Based on the provided GDS data, the first x-coordinate listed for the first polygon is 320.78.

Answer: 320.78
<answer>320.78</answer>
Answer by VLM: `320.78`
Ground Truth: `320.97`

Figure 26: Additional sample output with an incorrect answer.