Appendix for: Vision language models are *blind*

A Description of models tested

On our benchmark, we find that some chat interfaces perform *worse* than their API counterparts (*e.g.*, the system on `gemini.google.com` is worse than `Gemini-1.5 Pro` on `aistudio.google.com`) perhaps due to their extra finetuning [4] or specific `system` prompts [2] that attempt to align VLMs with a company's policies. Similarly, we find `GPT-4o` and Claude 3 models in `perplexity.ai` to perform worse than the original API models. To make sure we test the best VLMs available, we access all four models via their available APIs on OpenAI, Google, and Anthropic.

We describe below the exact API versions and settings for each model.

A.1 `GPT-4o`

We access the API for `GPT-4o` (gpt-4o-2024-05-13) via `platform.openai.com` and use all *default* settings including:

- `temperature`: 1.0
- `detail`: auto (see API doc), *i.e.*, the system will automatically decide whether to use the "low-res" (85 tokens) or "high-res" mode (85 tokens and a set of 170 tokens for every 512×512 tile).

A.2 `Gemini-1.5 Pro`

`Gemini-1.5` (gemini-1.5-pro-latest) API is accessible via `aistudio.google.com`, and we use all *default* settings.

A.3 `Claude-3 Sonnet`

We access the anthropic API via `console.anthropic.com` to use `Sonnet-3` (claude-3-sonnet-20240229) with *default* settings.

A.4 Claude-3.5 Sonnet

We follow the same process as for Sonnet-3 and use console.anthropic.com to access Sonnet-3.5 (claude-3-5-sonnet-20240620) API with *default* settings.

A.5 LLaVA OneVision-qwen2

We run the publicly available code of LLaVA OneVision-qwen2 locally with temperature=0.2.

A.6 Phi-3.5-vision-instruct

We host Phi-3.5-vision-instruct on a local machine via their Hugging Face page with *default* settings.

A.7 InternVL-2

We access InternVL-2 via their Hugging Face repository, and use the *default* settings.

B Hallucinations and educated guesses are among VLMs' common failures

What is the color of the overlapping area between two circles?

The overlapping area between the two circles is turquoise (a shade of cyan).

The overlapping area of a green circle and a blue circle would be **cyan**.

There is no overlapping area between the two circles in this image. The image shows two separate circles - one green circle and one blue circle - that are not intersecting or overlapping with each other. ✗

I apologize, but there is no overlapping area between the two circles in this image. The image shows two distinct circles - one green and one blue ✗ [...]

Fig. F1: Sometimes VLMs (and) fail to see clearly and make educated guesses.

C Evaluating open-source models on BlindTest

On average, 8 well-known open-source models perform *worse* than the closed-source VLMs on BlindTest. However, the best-performing open-source model (LLaVA OneVision-qwen2-72B-ov), performs on-par with Sonnet-3 (45.92 vs. 45.73; Tab. T1). As model size increases, VLMs tend to perform better. For instance, the accuracy increase from LLaVA OneVision-qwen2-ov 0.5B to 72B is +27.1 (18.82 → 45.92; Tab. T1). In sum, **open-source VLMs exhibit similar limitations to closed-source models on BlindTest.**

Table T1: Open-source VLMs underperform the closed-source ones on BlindTest.

Model	Size									Task mean
Random		33.33	50.00	5.77	20.00	20.00	25.00	4.55	33.33	24.00
GPT-4o	n/a	41.61	75.91	74.23	41.25	20.21	55.83	39.58	53.19	50.23
Gemini-1.5	n/a	66.94	**93.62**	83.29	20.25	24.17	87.08	39.39	53.13	58.48
Sonnet-3	n/a	43.41	86.46	72.06	29.79	1.87	65.00	36.17	31.11	45.73
Sonnet-3.5	n/a	**75.36**	90.82	**87.88**	**66.46**	**77.71**	**92.08**	**74.26**	**58.19**	**77.84**
LLaVA OneVision-qwen2-ov	72B	45.83	90.92	44.71	20.00	11.74	87.07	8.95	58.06	45.92
LLaVA OneVision-qwen2-si	72B	45.33	83.48	38.14	20.00	11.46	57.50	10.23	48.06	38.41
LLaVA OneVision-qwen2-ov	7B	48.17	83.93	42.79	20.00	7.29	42.92	21.02	47.22	39.17
LLaVA OneVision-qwen2-si	7B	44.50	84.67	40.22	20.00	7.29	58.75	14.01	55.00	40.00
LLaVA OneVision-qwen2-ov	0.5B	17.28	75.07	9.78	12.50	9.58	20.42	0.38	5.56	18.82
LLaVA OneVision-qwen2-si	0.5B	33.14	73.21	6.25	27.29	2.50	14.58	1.13	26.11	23.03
InternVL-2	8B	47.28	91.00	57.69	20.00	13.96	28.33	7.57	60.28	40.76
Phi-3.5-vision-instruct	4.2B	37.78	83.63	16.51	18.75	11.46	32.50	11.74	19.72	29.01
Mean		45.55	84.39	47.79	26.36	16.60	53.50	22.03	42.97	42.28

D Advanced prompting techniques

D.1 Finding: Meta-prompting and 2-shot examples do not improve the VLMs' performance on the two circles task (●●)

We run GPT-4o and Sonnet-3.5 on the two circle task (●●) with **2-shot** (providing 2 example images with answers) and **meta-prompting**[1] [3]. We find them to perform worse than our baseline prompts (Tab. T2). An explanation is that the VLMs already understand the questions but are limited by the ability to "see". These techniques are not helpful perhaps because BlindTest tasks intuitively do not benefit from thinking aloud.

Table T2: In-context examples and meta-prompting do not improve the overall accuracy of the Sonnet-3.5 and GPT-4o on the ●● task.

Prompt		
Baseline	**91.66**	**72.69**
Meta-prompting [3]	90.53	65.62
2-shot	77.93	68.00

[1] *Describe the image in detail first, and then answer: Are the two circles overlapping? Y/N.*

E Two touching circles task

E.1 Benchmark Construction Details

To create our benchmark, we use 5 parameters to control the diversity of the samples.

- **Color**: We fix the colors for each circle to use {*magenta*, *dodgerblue*}.
- **Image size**: We use the physical size, and the DPI arguments in *matplotlib* to initialize the image size. The physical size is fixed to 5×5, and the DPI $\in \{100, 200, 300\}$. The output image sizes are {384, 769, 1155}px.
- **Diameter**: We use uniform diameters for both circles and choose the value proportional to the image size, where the diameter is $\{\frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}\}$ of the image size.
- **Distance**: The boundary-to-boundary distance between circles is a fraction of the diameter chosen from {-0.25, -0.2, -0.15, -0.1, -0.05, 0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}. Based on our definition, center-to-center distance is (2+distance)×diameter.
- **Rotation**: We include 2 main rotations (vertical and horizontal), and 2 diagonal rotations.

We use the center of the image as the origin so that it always aligns with the midpoint of distances between two circles. This systematic process results in a benchmark comprising 768 images (see Tab. T3 and Fig. F2).

Code The code is available at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/TouchingCircle/TwoTouchingCircles.ipynb`.

Prompts

1. *Are the two circles touching each other? Answer with Yes/No.*
2. *Are the two circles overlapping? Answer with Yes/No.*

Groundtruth We consider two circles overlapping and touching (O, T) if $d < 0.0$; non-overlapping but touching ($\overline{O}$, T) if $d = 0.0$; and non-overlapping & non-touching ($\overline{O}$, $\overline{T}$) when $d > 0.0$ (Fig. F3). Random-baseline accuracy: 50%.

Table T3: Number of samples for each category is the same in our benchmark, where they sum to 768.

Parameter	Values	Samples	Total Samples
Image size (C)	384px	256	768
	769px	256	
	1155px	256	
Diameter (ϕ)	$\frac{C}{4}$	192	768
	$\frac{C}{5}$	192	
	$\frac{C}{6}$	192	
	$\frac{C}{7}$	192	
Distance	-0.25$\times\phi$	48	768
	-0.2$\times\phi$	48	
	-0.15$\times\phi$	48	
	-0.1$\times\phi$	48	
	-0.05$\times\phi$	48	
	0.0$\times\phi$	48	
	0.05$\times\phi$	48	
	0.1$\times\phi$	48	
	0.15$\times\phi$	48	
	0.2$\times\phi$	48	
	0.25$\times\phi$	48	
	0.3$\times\phi$	48	
	0.35$\times\phi$	48	
	0.4$\times\phi$	48	
	0.45$\times\phi$	48	
	0.5$\times\phi$	48	
Rotation	Vertical	192	768
	Horizontal	192	
	Diagonal 1	192	
	Diagonal 2	192	

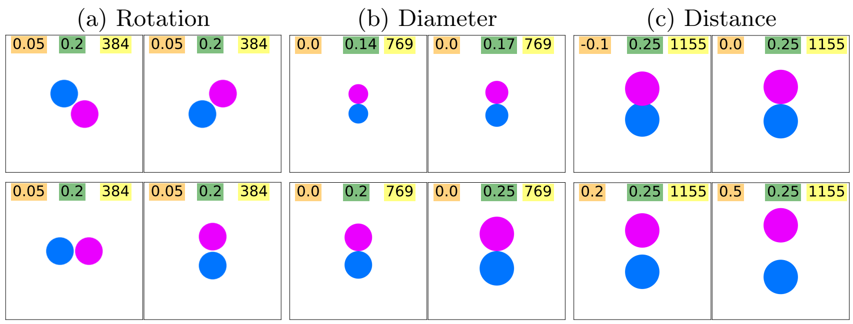


Fig. F2: Samples in the benchmark include various settings for drawing two circles. We start with choosing a rotation (a) and change other parameters of each plot, *e.g.*, the diameter (b), the distance between perimeters (c), and the image size (in pixels). We show the parameters that can be changed to generate different samples inside the legend.

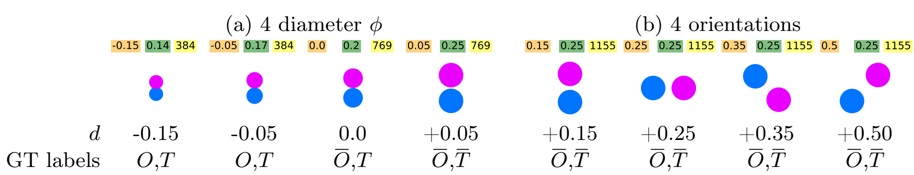


Fig. F3: For each image size and distance d, we vary diameter (a) and orientation (b). Groundtruth: O: overlapping. T: touching. $\overline{O}$: non-overlapping. $\overline{T}$: non-touching.

E.2 Finding: image resolution does not affect VLMs performance

Fig. F4-left shows that VLMs are almost invariant to the image resolution. For example, GPT-4o and Sonnet-3's performance saturates at 769px, and Sonnet-3.5 slightly performs worse at 769 and 1155px compared to 384px. Gemini-1.5, however, is fairly consistent across different resolutions. Based off these results, we conclude that VLMs' ability to see the intersection of two circles does not depend on the quality of the image.

E.3 Finding: the vertical rotation closes the gap between models' performance

As shown in Fig. F4-middle, arranging the circles in vertical rotation causes the models to perform similarly on the benchmark. Although Gemini-1.5 slightly performs better at diagonal and Sonnet-3.5 at horizontal rotation, VLMs perform relatively better at vertical rotation. This suggests that the task complexity due to various rotations is not the main source of low performance in VLMs.

E.4 Finding: Increasing the distance improves the VLMs' accuracy

VLMs perform better when the distance increases from zero to positive values (see Fig. F4-right). However, Sonnet-3.5 is more conservative than other VLMs that mostly answer "Yes", which results in its lowest performance at negative distances.

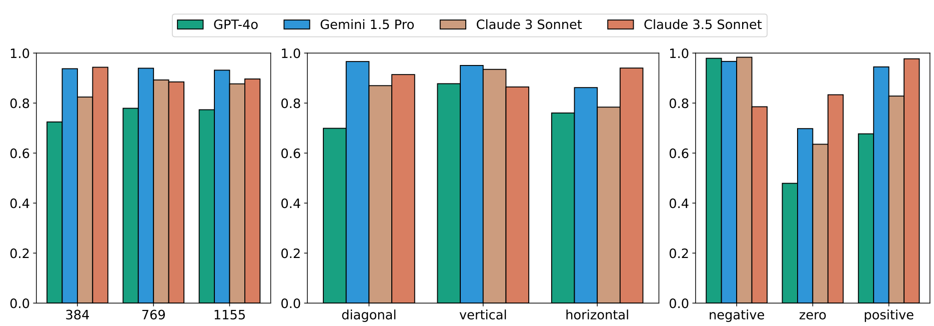


Fig. F4: There is no correlation between the resolution of the image (left) and VLMs' performance. Across various rotations (middle), VLMs perform almost the same at vertical. Most failure cases are at boundary distances (right).

E.5 Finding: VLMs prefer a specific rotation

Tab. T4 shows that VLMs prefer different rotations. For example, `GPT-4o` performs best at vertical, `Gemini-1.5` at diagonal, `Sonnet-3.5` at horizontal, and `Sonnet-3` at vertical.

Table T4: VLM accuracy is often best at a specific two-circle orientation. Across three different resolutions, `GPT-4o` and `Sonnet-3` perform much better when two circles are arranged vertically. In contrast, `Gemini-1.5` and `Sonnet-3.5` prefer the diagonal and horizontal orientations, respectively.

	Resolution	Rotation				
a	384px	VERTICAL	**82.03**	93.75	**96.09**	90.62
		HORIZONTAL	71.87	89.84	75.78	**96.87**
		DIAGONAL	69.97	**95.70**	78.90	94.92
b	769px	VERTICAL	**89.84**	95.31	**92.97**	84.37
		HORIZONTAL	80.47	85.94	83.59	**91.41**
		DIAGONAL	70.70	**97.26**	90.23	89.06
c	1155px	VERTICAL	**91.41**	96.09	91.41	84.37
		HORIZONTAL	75.78	82.81	75.78	**93.75**
		DIAGONAL	71.09	**96.87**	**91.80**	90.23

E.6 Finding: Various coloring of the two circles has minimal effects on the overall accuracy

VLMs' performance in the ●● task does not change substantially (±5.79% for `Sonnet-3.5` and ±10.81% for `GPT-4o`) when tested against different colors (Tab. T5).

Table T5: The small △ in VLMs' performance by changing the colors of the circles suggests that coloring does not significantly impact the VLMs' vision capabilities.

Color		
Magenta - Blue (baseline)	90.93	76.10
Red - Black	96.72	86.91
Yellow - Green	94.39	80.52
Dark gray - Light gray	93.62	82.25

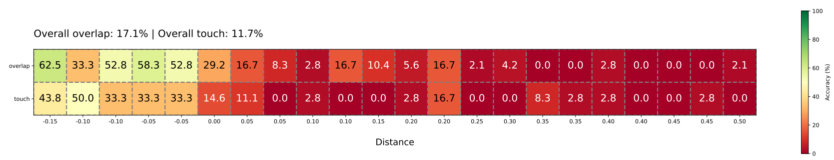
E.7 Results for fine-tuning Bunny on the two touching circle

In order to determine if fine-tuning could improve the model's performance on this task we attempted to fine-tune Bunny [1] (Bunny-v1.1-Llama-3-8B-V with the original weights) on the two touching circles task. We fine-tuned Bunny using datasets of sizes: 10K, 20K, 50K, and 100K samples, each containing a balanced number of instances where the circles are either overlapping or separate (equal number of YES/NO answers in the training set).

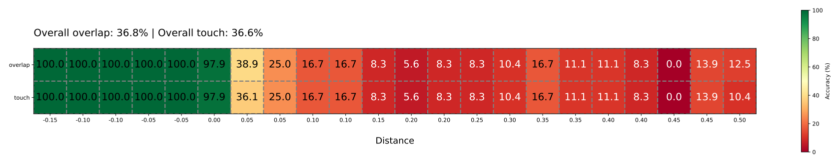
The baseline model, without any fine-tuning, achieved 17.1% accuracy for task overlap and 11.7% for touching circles. After fine-tuning, we observed improvements with smaller datasets, such as 10K and 20K cases, where accuracy reached up to 36.8%. However, increasing the number of samples did not lead to better performance. In some instances, such as with the 50K dataset, the model failed to predict anything and only generated the *end-of-text* token.

The loss values for all these experiments were very close to zero, indicating that the model overfits the training set but fails to generalize. This suggests that training on this task is not straightforward and may require a combination of multiple tasks or that this problem does not have a simple solution.

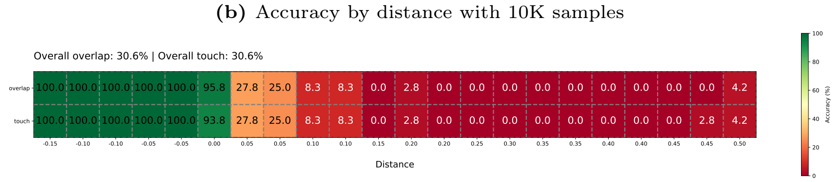
Fig. F5 shows the accuracy breakdown by distance of the two circles. The model's performance improves when the circles are overlapping, but when there is a long distance between them, the model does not generalize well and cannot provide accurate answers.



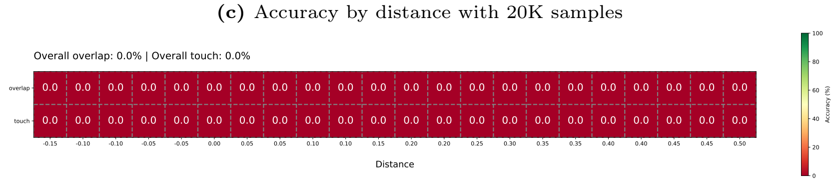
(a) Accuracy by distance without fine-tuning



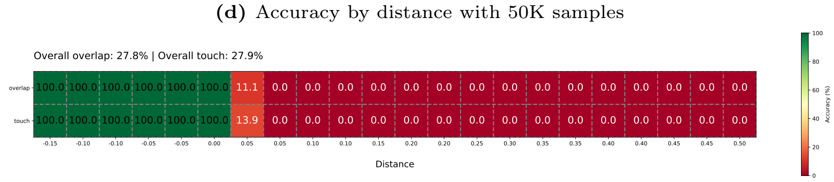
(b) Accuracy by distance with 10K samples



(c) Accuracy by distance with 20K samples



(d) Accuracy by distance with 50K samples



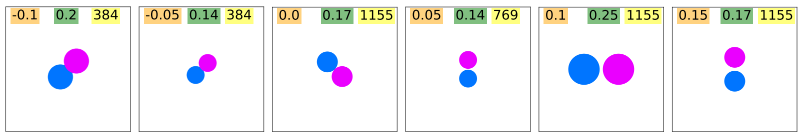
(e) Accuracy by distance with 100K samples

Fig. F5: Comparison of Bunny's accuracy by distance with and without fine-tuning

E.8 Additional examples

We show examples of models' responses to the prompts on the two touching circles task in Fig. F6.

Are the two circles touching each other? Answer with Yes/No.

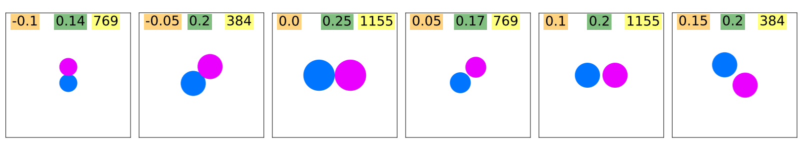


GPT-4o	Yes ✓	Yes ✓	Yes ✓	No ✓	Yes ✗	Yes ✗
Gemini-1.5	Yes ✓	Yes ✓	Yes ✓	Yes ✗	No ✓	No ✓
Sonnet-3	Yes ✓	Yes ✓	Yes ✓	Yes ✗	Yes ✗	Yes ✗
Sonnet-3.5	Yes ✓	No ✗	Yes ✓	No ✓	No ✓	No ✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F6: Some VLMs consistently fail by not seeing the distance between the two circles at large distances and resolutions (GPT-4o and Sonnet-3–rightmost). Gemini-1.5 and Sonnet-3 also struggle to see the small gaps between circles (*distance=0.05*)

Are the two circles overlapping? Answer with Yes/No.



GPT-4o	Yes ✓	Yes ✓	Yes ✗	Yes ✗	No ✓	Yes ✗
Gemini-1.5	No ✗	Yes ✓	Yes ✗	No ✓	No ✓	No ✓
Sonnet-3	Yes ✓	Yes ✓	Yes ✗	Yes ✗	Yes ✗	No ✓
Sonnet-3.5	No ✗	No ✗	No ✓	No ✓	No ✓	No ✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F7: VLMs consistently fail at smaller distances. However, when the gap is large and clearly visible, GPT-4o remains unreliable. Sonnet-3.5 tends to conservatively answer "No" regardless of the actual distance between the two circles.

F Task 3: Identifying the circled letter in a string

F.1 Task construction

Each image is created using a combination of the below variables.

– **String**: We test three strings {Acknowledgement , Subdermatoglyphic , tHyUiKaRbNqWeOpXcZvM }. While they all have letters of varying widths and heights, the first two are English words while the latter is a string of random characters. Subdermatoglyphic is the longest English word that has no letter repetitions.
– **Letter**: We draw the oval () over every letter, one at a time, in each string.
– **Font**: We use 2 different font families for each word, OpenSans and Helvetica.
– **Oval () thickness**: We generate the with 3 various line thicknesses.
– **Scaling factor**: Since each letter has a unique size, we use a scaling factor to control the size of the .

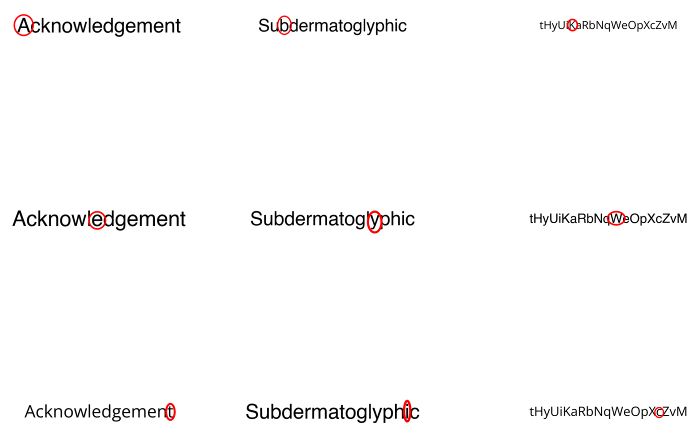
Finally, we render the text on a white canvas with a size of 1250×1250 pixels, and we produce 90 images for Acknowledgement , 102 for Subdermatoglyphic , and 120 samples for tHyUiKaRbNqWeOpXcZvM (see Tab. T6 and Fig. F8).

Evaluation To determine the models' prediction, we extract the character enclosed in {curly braces} in the models' response.

Code The code is available at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/CircledWord/GenerateSamples.ipynb`.
Prompts

1. *Which letter is being circled? Please provide your answer in curly brackets, e.g. {a}*
2. *Which character is being highlighted with a red oval? Please provide your answer in curly brackets, e.g. {a}*

Groundtruth letters must match predicted letters exactly (case-insensitive).



(a) Acknowledgement (b) Subdermatoglyphic (c) tHyUiKaRbNqWeOpXcZvM

Fig. F8: Our benchmark comprises three different words, of which one letter is circled by the red oval in each image.

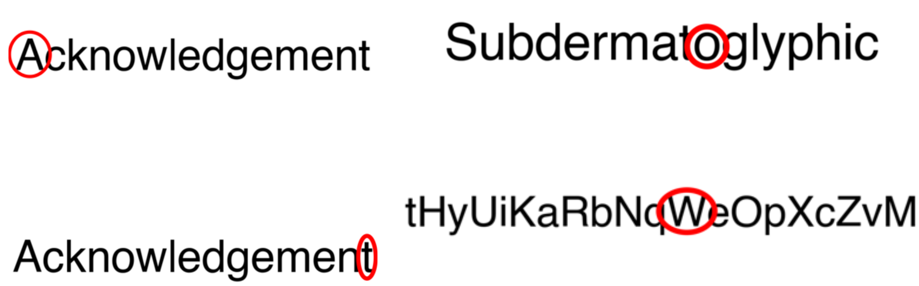


Fig. F9: Which letter is being red circled?

Table T6: The breakdown of our benchmark based on the number of different parameters shows our data is balanced for each word.

Parameter	Values	Samples	Total Samples
Word	Acknowledgement	90	312
	Subdermatoglyphic	102	
	tHyUiKaRbNqWeOpXcZvM	120	
Font	OpenSans	156	312
	Helvetica	156	
Image size (C)	1250x1250 pixels	312	312
Oval thickness	$\frac{1}{200} \times C$	104	312
	$\frac{1}{250} \times C$	104	
	$\frac{1}{300} \times C$	104	

F.2 Finding: VLMs mostly confuse the adjacent character for the circled letter

Models often mistake the neighboring characters as actual circled letters. For example, Fig. F10 shows that for `Sonnet-3.5` in the word Acknowledgement , all of the incorrect predictions for "n" are the neighboring letter "t". For `Gemini-1.5`, in the word tHyUiKaRbNqWeOpXcZvM , where letters "K" and "a" are adjacent, 100% of the mispredictions for "K" are the letter "a" (see Fig. F11).

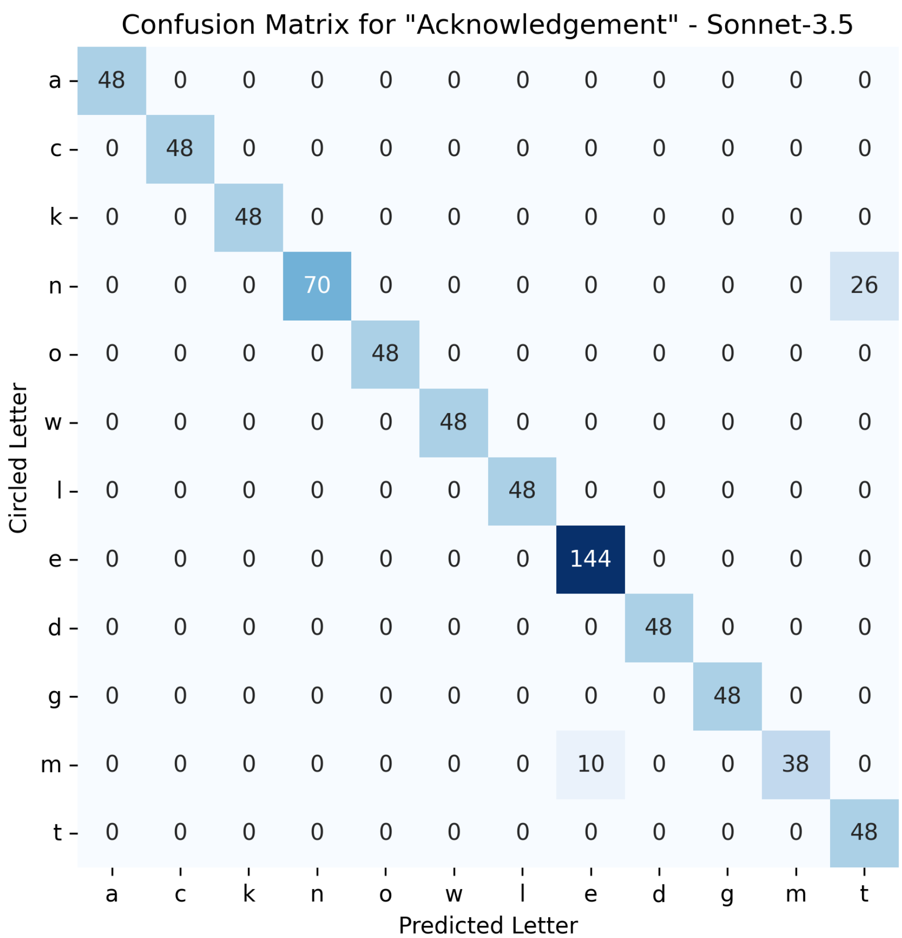


Fig. F10: Adjacent letters are the most common wrong prediction for `Sonnet-3.5` in Acknowledgement , *e.g.*, letter "e" is predicted instead of "m" 20.83% of the time, or letter "t" is predicted instead of "n" 27.08% of the time.

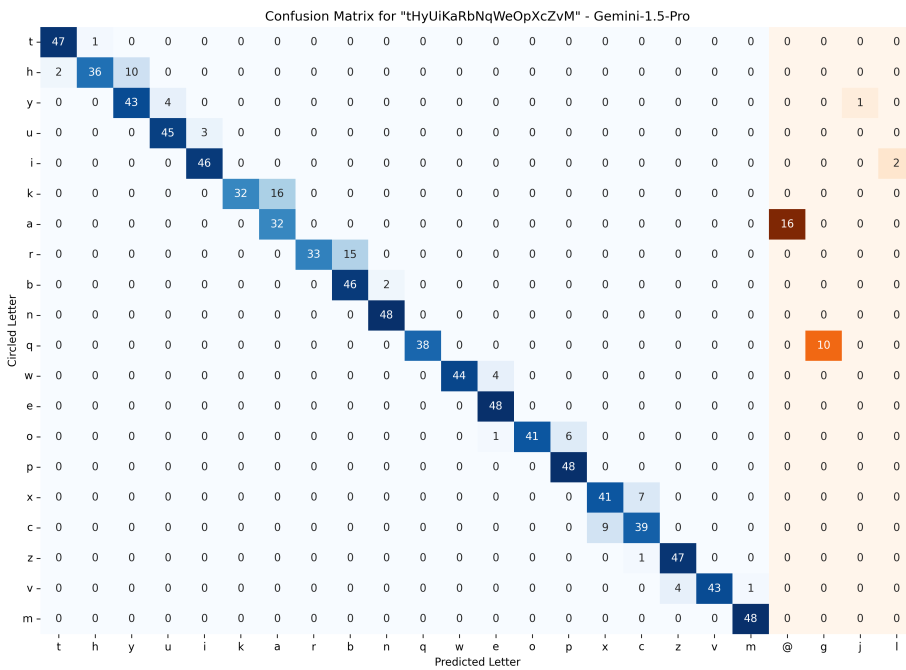


Fig. F11: `Gemini-1.5` tends to confuse adjacent letters like "a" and "K". `Gemini-1.5` also hallucinates the circled letter "a" as being the character "@" and it confuses the letter "q" for the letter "g".

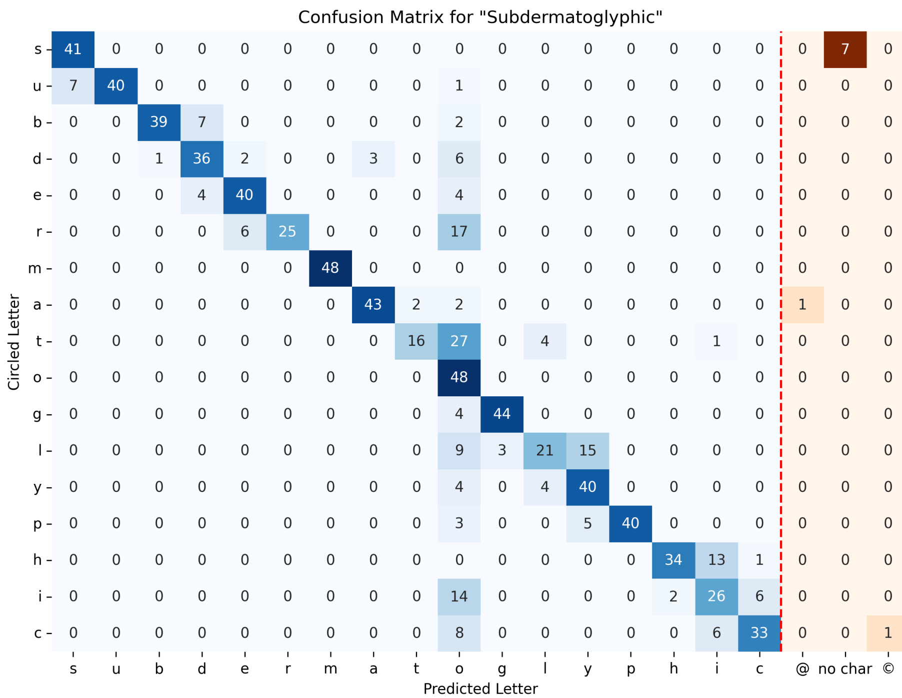


Fig. F12: Aggregate confusion matrix summed over all 4 VLMs and 12 responses for each letter in the word Subdermatoglyphic. Models mostly mispredict characters near the circled letter. VLMs sometimes hallucinate characters that do not even exist in the word, *e.g.*, “@” (right panel).

F.3 Finding: GPT-4o and Gemini-1.5 confuse the red oval as part of the letter

Figs. F13 and F14 show that Gemini-1.5 and GPT-4o sometimes fail to recognize that the red oval is not part of the letter. Gemini-1.5 tends to predict that the circled letter "a" is actually the "@" sign (see Fig. F13). GPT-4o on the other hand tends to predict "o" regardless of which letter is circled (Fig. F14).

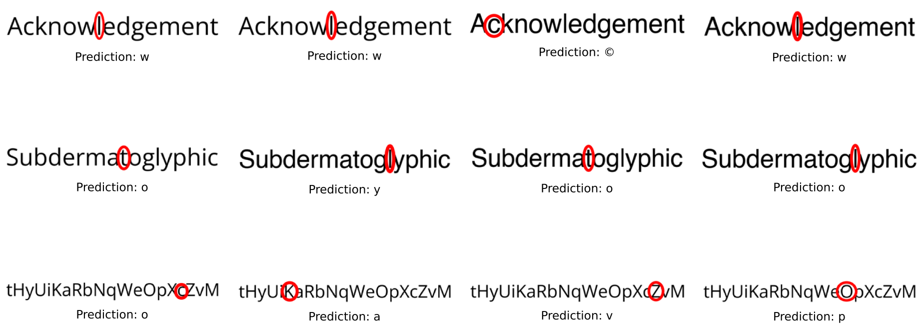


Fig. F13: Random samples for different words, and Gemini-1.5's predictions, where Gemini-1.5 mostly predicts the adjacent letters or confuses the red oval as part of the circled letter.

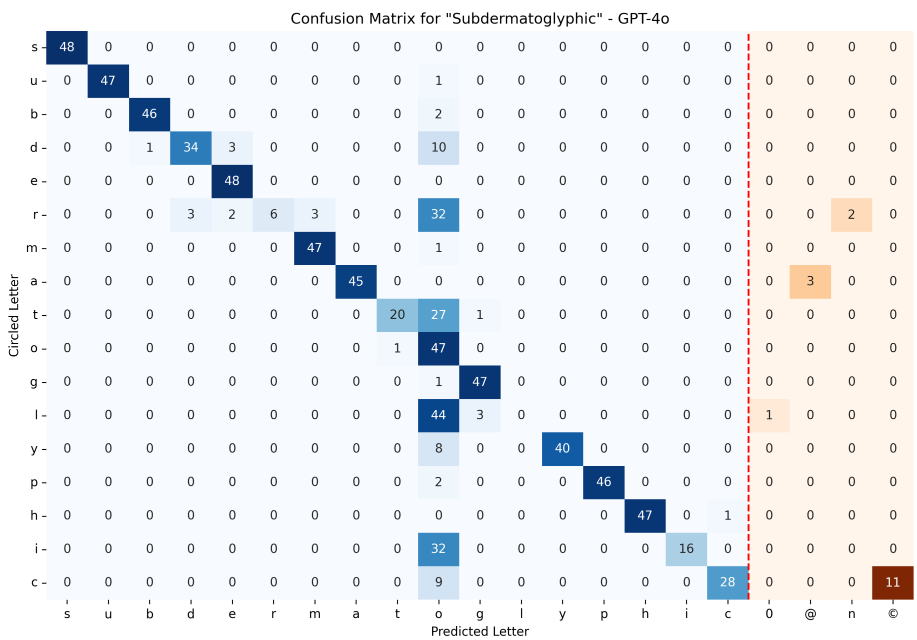


Fig. F14: `GPT-4o` is the most sensitive to confuse the red oval as part of the letter, where it often predicts “o” instead of “l”, “y”, “p”, and “c” in the word Subdermatoglyphic .

F.4 Finding: Models perform similarly across two common font families

As shown in Fig. F15, models do not show a significant variance over different fonts, suggesting our choice of font is not a reason for their unreliable performance.

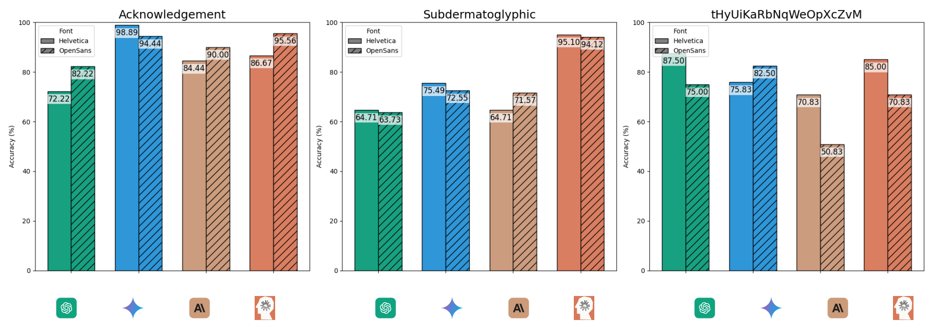


Fig. F15: VLMs do not show a consistent trend between fonts. This suggests that using different spacing between letters, letter styles, and letter size has minimal effects on the VLMs' ability to see the content of the red oval.

F.5 Finding: models are invariant to our choice of prompts

Our choice of prompts has little impact on the performance of the models as depicted in Fig. F16.

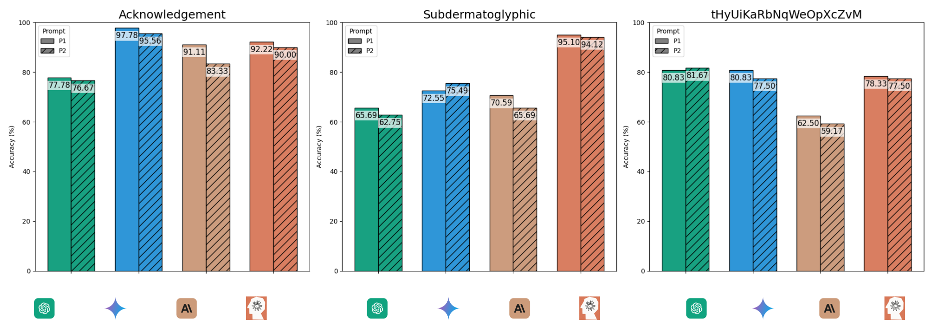


Fig. F16: Model performance breakdown for different prompts P1: *"Which letter is being circled? Please provide your answer in curly brackets, e.g. {a}"* and P2: *"Which character is being highlighted with a red oval? Please provide your answer in curly brackets, e.g. {a}"*. Regardless of the prompts, VLMs perform similarly in seeing the contents of the red oval.

F.6 Finding: VLMs fail to see the circled letter on various English words

On average, SOTA VLMs show the same limitations across various English words (see Tab. T7). However, the mean accuracy over words with less and non-repetitive letters is higher than others (the mean accuracy over the word question is 90.36% while it is 79.86% on the word civilization).

Table T7: Overall, VLMs perform better on short words without repetitive letters, *e.g.*, question . Yet, the mean accuracy of four SOTA VLMs (86.43%) is far from being 100% accurate.

String					Mean
question	98.96	100.00	62.50	100.00	90.36
syndrome	91.67	91.67	70.83	91.58	86.44
environmental	82.69	89.10	73.72	89.74	83.81
civilization	79.17	87.50	61.80	90.97	79.86
tournament	89.17	92.50	85.00	100.00	91.67
Mean accuracy	88.33	92.15	70.77	94.46	86.43

Table T8: Except for `GPT-4o`, all other models have a higher accuracy (%) on the two English words than on the random string, suggesting that VLMs might leverage their familiarity with a known word to make educated guesses.

String					Mean
Acknowledgement	77.22	96.67	87.22	91.11	88.06
Subdermatoglyphic	64.42	74.02	68.14	94.10	75.25
tHyUiKaRbNqWeOpXcZvM	81.25	79.17	60.83	77.92	74.79
Mean accuracy	74.23	83.29	72.06	87.88	79.37

F.7 Additional Examples

Examples from our evaluation of VLMs on the circled letter task are shown in Fig. F17.

Which character is being highlighted with a red oval? Please provide your answer in curly brackets, e.g. {a}

GPT-4o	e	✗	e	✗	c	✓	g	✓	@	✗	r	✗
Gemini-1.5	m	✓	l	✓	c	✓	o	✗	a	✓	b	✓
Sonnet-3	e	✗	n	✗	i	✗	o	✗	i	✗	n	✗
Sonnet-3.5	e	✗	l	✓	©	✗	g	✓	a	✓	b	✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F17: Most failure cases consist of predicting the adjacent letters (predicting "e" instead of "m" in GPT-4o, Sonnet-3, and Sonnet-3.5 leftmost), and confusing the red oval as part of the circled letter (predicting "©" instead of "c" in Sonnet-3.5).

G Counting the number of line intersections task

G.1 Benchmark Construction Details

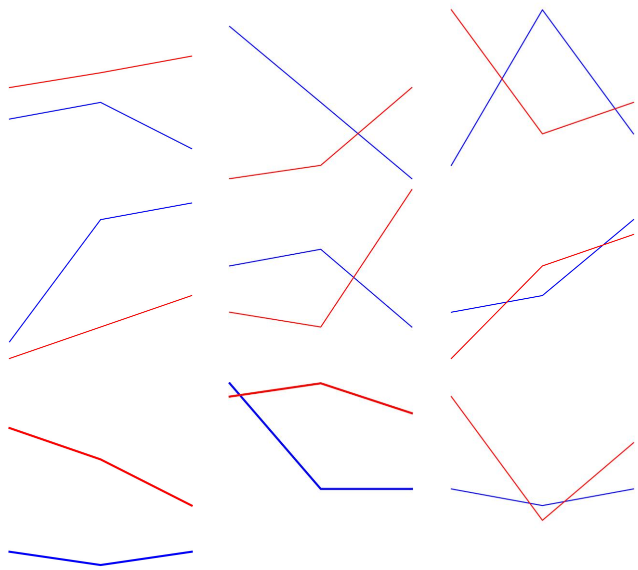
To create our benchmark, we use 5 parameters to control the diversity of the samples.

- **Color**: We fix the colors for each line to use {*blue*, *red*}
- **Image size**: We generate 3 different image resolutions, {384, 768, 1152}px, to include high and low-resolution samples in our tests.
- **Grid size**: We divide the whole image into a 12×12 grid with 144 equal sized cells. Then, we choose 3 points for each (blue and red) line on the grid to draw the line plots. To make sure the line plots do not overlap and the intersections are well distanced from the edges, we fix the x-coordinates, and randomly sample y-coordinates from the invisible 12×12 grid. The grid sampling also avoids picking the same coordinates for both blue and red line.
- **X-coordinate**: For each point, we choose x= $0, \frac{C}{2}, C$ px.
- **Y-coordinate**: For each x-coordinate above, we randomly sample a y-coordinate from 12 pre-defined rows in the 12×12 grid. We also ensure that no blue and red points share the exact same (x,y) coordinates.
- **Line thickness**: We vary the line widths with standard *matplotlib* values (2 and 4), which renders into a width of $0.005 \times C$ and $0.01 \times C$ pixels, respectively.
- **Number of intersections**: We count the intersections based on the three points defined for each line ((x, y_1), (x, y_2), and (x, y_3)).

We repeat the process until we have 2 line widths×3 image sizes×100 samples of 0, 1, and 2 intersections, resulting in 1800 images (see Tab. T9 and Fig. F18).
Code The code to generate the images is at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/LineIntersection/2dline.ipynb`.
Prompts We ask each question using two different wordings:

1. *How many times do the blue and red lines touch each other? Answer with a number in curly brackets, e.g., {5}.*
2. *Count the intersection points where the blue and red lines meet. Put your answer in curly brackets, e.g., {2}.*

Groundtruth answers are $\in \{0, 1, 2\}$ (random-baseline accuracy: 33.33%).



(a) zero intersection (b) one intersection (c) two intersections

Fig. F18: Samples from the two intersecting lines benchmark that contain 0, 1, or 2 line intersections.

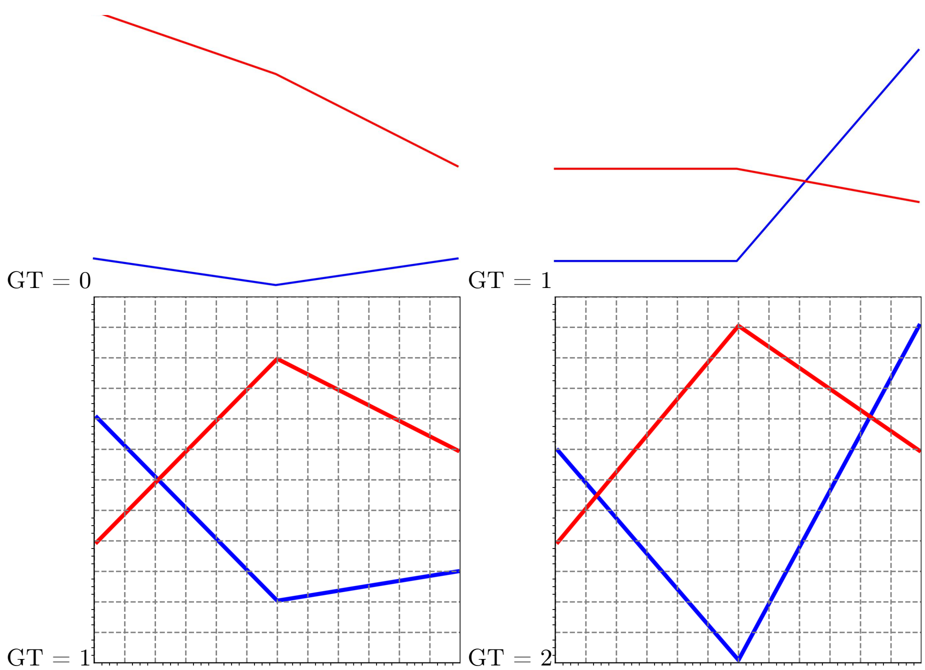


Fig. F19: Count intersections

Table T9: We generate the same number of images based on various parameters to have a balanced benchmark.

Parameter	**Values**	**Samples**	**Total Samples**
Image size (C)	384px	600	1800
	768px	600	
	1152px	600	
Line thickness	$0.005 \times C$	900	1800
	$0.01 \times C$	900	
Number of intersections	0	600	1800
	1	600	
	2	600	

G.2 Finding: image resolution does not influence VLM's ability to count the line intersections

Fig. F20 depicts that increasing the resolution in our plots does not help the VLMs see and count the intersections of 2D lines.

G.3 Finding: VLMs perform similarly when the lines intersect twice

Our benchmark consists of images with 0 to 1 line intersections with an identical number of samples in each category. We break down the performance of each model based on the groundtruth in Fig. F20-right to analyze where VLMs mostly fail in their overall performance. As depicted in Fig. F20-right, VLMs perform relatively better on 2 intersections (except `Sonnet-3.5` that is best on 1 intersection) and perform worse on 0 intersections (`Sonnet-3` is the worst).

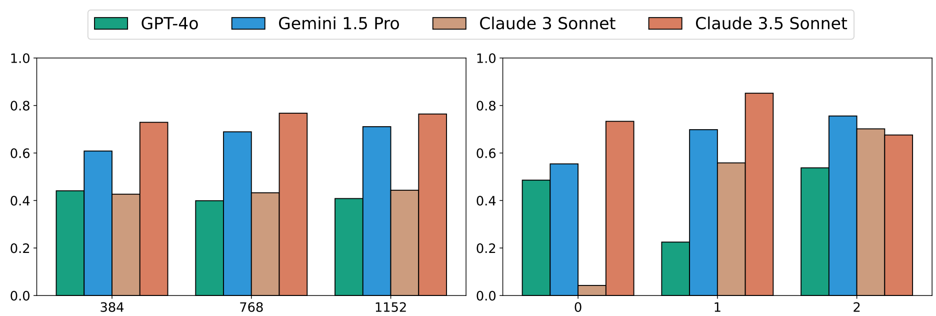
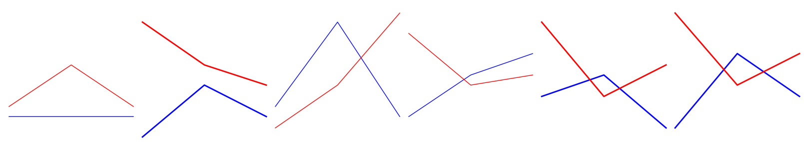


Fig. F20: VLMs are not sensitive to the image resolution to see and count the number of intersections (left). When the lines intersect twice, VLMs perform similarly, while `Sonnet-3.5` has the least variance over different numbers of intersections (right).

G.4 Additional Examples

We show examples of models' responses to the prompts on the counting the number of line intersections task in Fig. F21.

Count the intersection points where the blue and red lines meet. Put your answer in curly brackets, e.g., {2}.

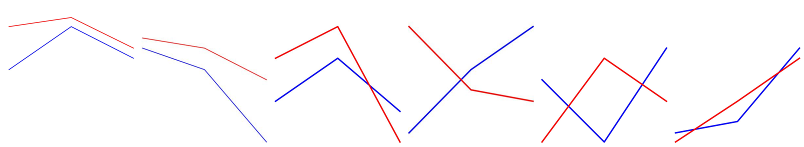


GPT-4o	2	✗	0	✓	2	✗	2	✗	3	✗	4	✗
Gemini-1.5	2	✗	1	✗	1	✓	1	✓	2	✓	2	✓
Sonnet-3	2	✗	2	✗	2	✗	1	✓	4	✗	1	✗
Sonnet-3.5	2	✗	1	✗	1	✓	1	✓	2	✓	2	✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F21: All VLMs, except Sonnet-3.5, fail at counting when there is no intersection, even when the gap between 2 lines is large (second image on the left).

How many times do the blue and red lines touch each other? Answer with a number in curly brackets, e.g., {5}.



GPT-4o	1	✗	0	✓	2	✗	2	✗	4	✗	1	✗
Gemini-1.5	1	✗	1	✗	1	✓	2	✗	1	✗	1	✗
Sonnet-3	4	✗	1	✗	2	✗	1	✓	4	✗	1	✗
Sonnet-3.5	1	✗	0	✓	2	✗	1	✓	3	✗	2	✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F22: VLMs cannot reliably count the intersections between the blue and red plots.

H Counting the number of nested squares

H.1 Benchmark Construction Details

We use 5 parameters to create the images of nested squares.

- **Depth**: For each image, we draw $N \in \{2, 3, 4, 5\}$ nested squares on the image. We refer to each square in this collection as a depth.
- **Initial size**: We choose a random size for the first square in the bounds of the image size.
- **Reduction factor**: We draw squares such that each depth is entirely contained by its previous depth. We use a reduction factor to scale the square sizes.
- **Center**: The first square's center is chosen to ensure it is entirely visible in the image. For the remaining squares, we choose the center based on the space between the previous square and the new reduced size.
- **Line thickness**: We use standard *matplotlib* line width parameter of (2=3px, 3=4px, 4=6px).

We continue to generate images until we have 30 samples for each depth, resulting in 120 images overall (see Tab. T10 and Fig. F23).

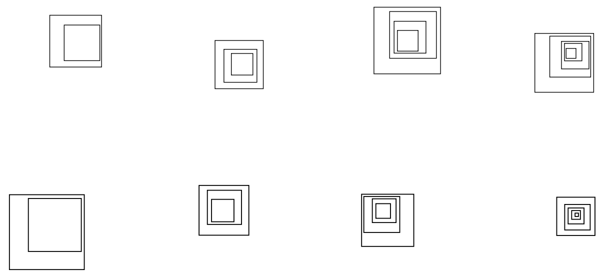
Code The code is available at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/NestedSquares/GenerateSamples.ipynb`.

Post-processing: To determine the model's answer, we use Python to extract the number from curly braces in the response e.g. {3}.

Prompts

1. *How many squares are in the image? Please answer with a number in curly brackets e.g., {10}.*
2. *Count total number of squares in the image. Answer with only the number in numerical format in curly brackets e.g. {3}.*

Groundtruth answers are $\in \{2, 3, 4, 5\}$ (random-baseline accuracy: 25%).



(a) 2 squares (b) 3 squares (c) 4 squares (d) 5 squares

Fig. F23: Random examples from the nested square task that have 2, 3, 4 or 5 squares in the image.

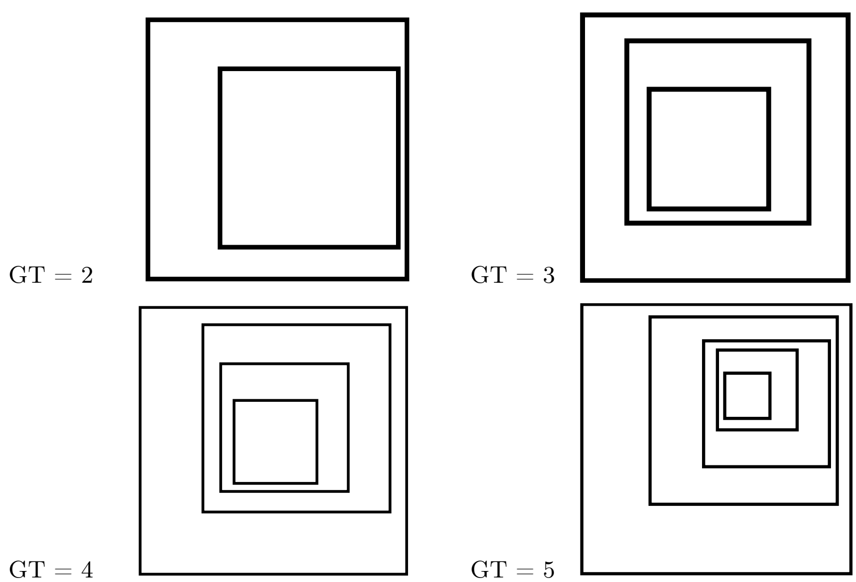


Fig. F24: Count squares

Table T10: Nested squares include the same number of samples for each parameter value in our benchmark.

Parameter	**Values**	**Samples**	**Total Samples**
Image size	1000px	120	120
Line thickness	3px	40	120
	4px	40	
	6px	40	
Number of Squares	2	30	120
	3	30	
	4	30	
	5	30	

H.2 Finding: the best-performing model is affected by line width

Tab. T11 shows that the best VLM on the nested square task (`Sonnet-3.5`) is more accurate as line width increases. In contrast, the worst model (`GPT-4o`) shows the opposite trend as the line thickness changes, suggesting that `GPT-4o` confuses squares when the borderline is thick.

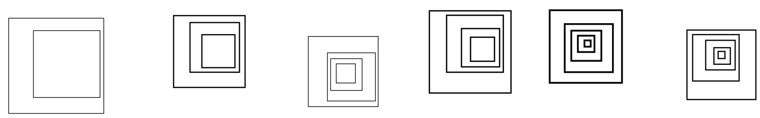
Table T11: Line thickness has minimal effect on VLMs' performance, suggesting that visual attributes of shapes are not critical to VLMs when asked to count the shapes.

Line width				
2	58.75	86.25	61.25	90.00
3	56.25	85.00	67.50	91.25
4	52.50	90.00	66.25	95.00
Average	55.83	87.08	65.00	92.08

H.3 Additional Examples

We show examples of models' responses to the counting the number of nested squares task in Fig. F25.

Count total number of squares in the image. Answer with only the number in numerical format in curly brackets e.g. {3}.

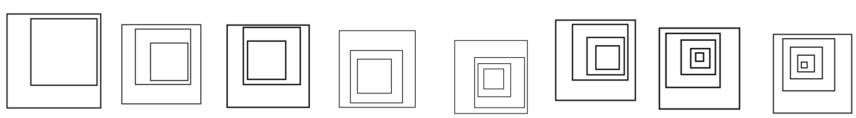


GPT-4o	2 ✓	5 ✗	5 ✗	5 ✗	5 ✓	5 ✓
Gemini-1.5	2 ✓	3 ✓	5 ✗	5 ✗	5 ✓	5 ✓
Sonnet-3	5 ✗	4 ✗	5 ✗	5 ✗	5 ✓	4 ✗
Sonnet-3.5	2 ✓	3 ✓	4 ✓	4 ✓	4 ✗	4 ✗

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F25: VLMs fail to reliably count the number of nested squares in an image.

How many squares are in the image? Please answer with a number in curly brackets e.g., {10}.



GPT-4o	3 ✗	5 ✗	3 ✓	5 ✗	5 ✗	5 ✗	6 ✗	6 ✗
Gemini-1.5	2 ✓	3 ✓	2 ✗	3 ✓	5 ✗	5 ✗	5 ✓	4 ✗
Sonnet-3	2 ✓	4 ✗	2 ✗	4 ✗	5 ✗	4 ✓	4 ✗	5 ✓
Sonnet-3.5	2 ✓	3 ✓	3 ✓	3 ✓	4 ✓	4 ✓	4 ✗	5 ✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F26: Counting nested squares is not easy for VLMs even when there are only two squares (leftmost). The task becomes harder as the count increases from 2 to 5. Sonnet-3.5 performs the best (92.08%) but still not at the 100% by humans.

I Counting the shapes in an Olympic-like logo

I.1 Benchmark Construction Details

We create the benchmark by generating images containing shapes resembling the Olympic logo by choosing a combination of settings.

- **Image size**: We fix the physical size of the image in *matplotlib* to 5×5, and change the resolution by changing the DPI value, which is $\in \{100, 200, 300\}$ to get images with sizes $\{384, 769, 1155\}$px.
- **Number of shapes**: We choose a number from {5, 6, 7, 8, 9}.
- **Color**: Each image is generated using two different coloring schemes. We generate an all-black version and a second version by randomly sampling colors from a colormap in *matplotlib*.
- **Distance**: To generate the interlaced shapes, we use a small boundary-to-boundary distance factor for each row of the shapes. We fix this value to 0.1 proportional to the diameter of circles or side length of pentagons.
- **Diameter**: We choose a uniform diameter for all the circles in each image from $\{\frac{1}{7}, \frac{1}{10}\}$ proportional to the image size.
- **Side length**: We follow the same policy for the diameter to choose the side length of the pentagons.
- **Line thickness**: We generate each image with {0.5, 1.0} line width of *matplotlib* standard. This results in {1px, 2px} lines in 384px images, {2px, 3px} lines in 769px images and {3px, 5px} lines in 1155px images.

We center the shape collection on the center of the image in two rows, and generate 480 images (see Fig. F27 and Tab. T12), 240 images of circles, and 240 images of pentagons.

Code The code is available at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/CountingCircles/OlympicCircles.ipynb` for circles and at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/CountingCircles/OlympicPentagons.ipynb` for pentagons.

Prompts

1. *How many* `{shapes}` *are in the image? Answer with only the number in numerical format.*
2. *Count the* `{shapes}` *in the image. Answer with a number in curly brackets e.g. {3}.*

where `{shapes}` = *circles* or *pentagons.*

Groundtruth answers are $\in \{5, 6, 7, 8, 9\}$ (random-baseline accuracy: 20%).

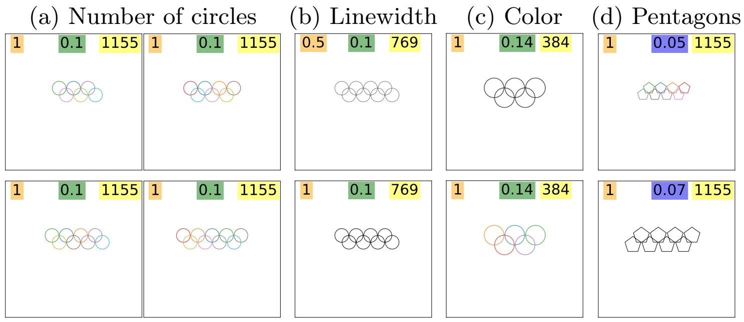


Fig. F27: We generate images of (a) different numbers of circles with various parameter changes, *e.g.*, the diameter, (b) the linewidth (in points) (c) colorings, and the image size (in pixels). For the pentagons, we vary the side length instead of the diameter.

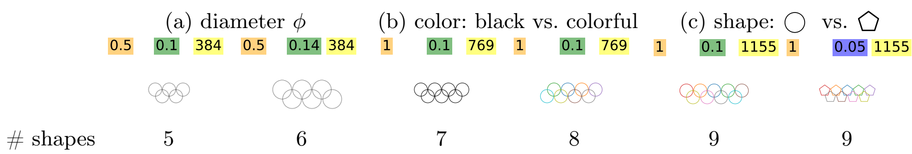


Fig. F28: Images span across three sizes and shapes span across two diameters (and two side lengths for ⬠), two color options (black vs. colored), and two line widths.

Table T12: We create 480 images containing circles and pentagons that are uniformly distributed over various parameters.

Parameter	Values	Samples	Total Samples
Shape	Circles	240	480
	Pentagons	240	
Image size (C)	384px	80	240
	769px	80	
	1155px	80	
Line thickness	$0.0025 \times C$ pixels	120	240
	$0.005 \times C$ pixels	120	
Number of shapes	5	48	240
	6	48	
	7	48	
	8	48	
	9	48	
Diameter/Side	$\frac{C}{7}$	120	240
	$\frac{C}{10}$	120	

I.2 Finding: different resolutions have no impact on most VLMs' performance

Fig. F29-a shows that VLMs are invariant to the resolution when asked to count the overlapping shapes. This suggests that the image quality has almost no effect on the performance, and VLMs cannot see the shapes.

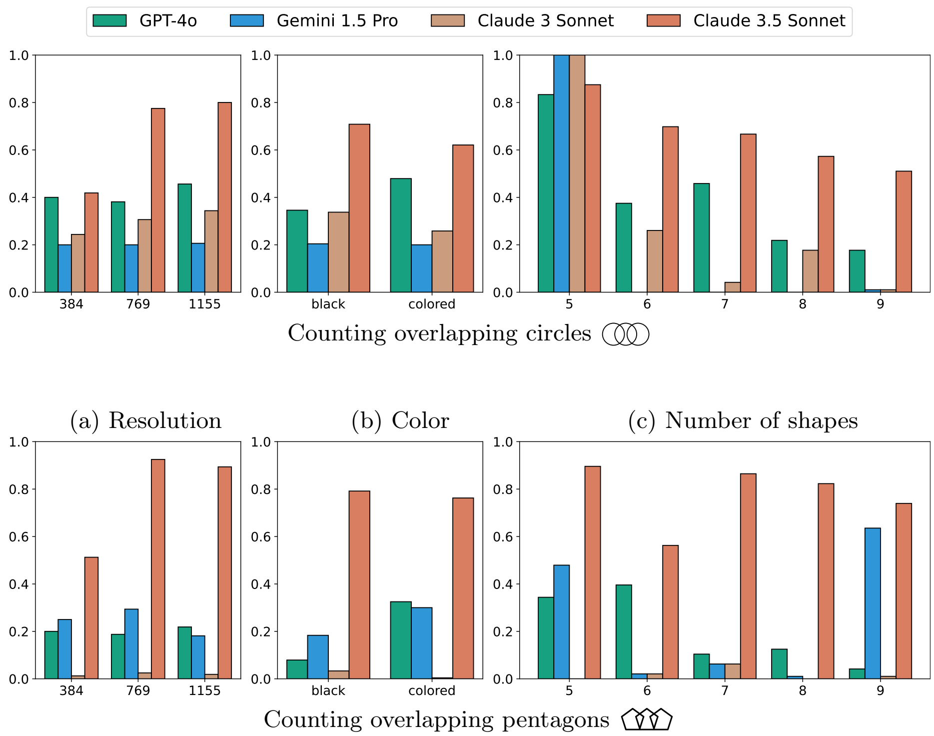


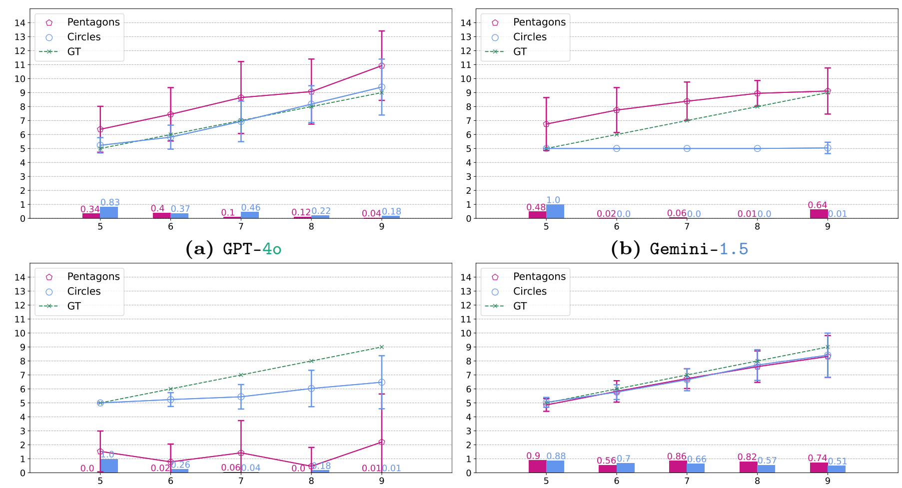
Fig. F29: VLMs perform better on counting overlapping circles (top) than overlapping pentagons (bottom). For most models, resolution (a) and colors (b) have minimal impact on performance. `Sonnet-3.5` performs better as the image size increases (a). `GPT-4o` performs better on colored shapes than on black shapes.

I.3 Finding: color-coding does not generally help the VLMs

While we expect the color-coding to make the shapes more distinct for the models, Fig. F29-b suggests that, except for `GPT-4o`, coloring the shapes has an opposite effect on the performance of the models.

I.4 Finding: Gemini-1.5 has the most biased predictions to the Olympic logo

Fig. F30 shows the overall trend of the predictions among SOTA VLMs. Gemini-1.5 (see Fig. F30b) tends to predict "5" significantly more often when asked to count the circles, while its predictions are more random for pentagons. This suggests the model's bias toward the Olympic logo.



(a) GPT-4o (b) Gemini-1.5 (c) Sonnet-3 (d) Sonnet-3.5

Fig. F30: Prediction trend for each VLM shows (a) GPT-4o has less variance in counting circles versus pentagons, (b) Gemini-1.5 is biased to predicting the number of circles to be 5, (c) Sonnet-3 tends to under count the number of shapes, and (d) Sonnet-3.5 has the least relative variance over both shapes.

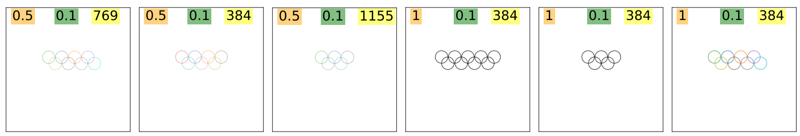
Table T13: Frequency (%) of predicting "5" when there are more than 5 circles (◯) or pentagons (⬠), *i.e.*, $N = 6, 7, 8, 9$ shapes in the image. For example, Gemini-1.5 predicts "5" circles 99.74% of the time but this tendency disappears in the case of ⬠(10.94%), showing a strong bias towards the 5-circle Olympic logo (among four models).

◯	⬠	◯	⬠	◯	⬠	◯	⬠
13.54	5.47	99.74	10.94	64.32	5.73	15.89	10.16

I.5 Additional Examples

We provide examples of VLMs' responses in Fig. F31.

Count the circles in the image. Answer with a number in curly brackets e.g. {3}

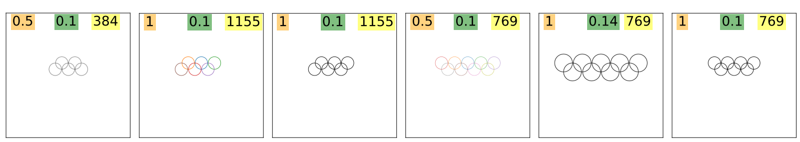


GPT-4o	9	✗	6	✗	5	✓	8	✗	5	✓	8	✓
Gemini-1.5	5	✗	5	✗	5	✓	5	✗	5	✗	5	✗
Sonnet-3	5	✗	5	✗	5	✓	8	✗	5	✓	8	✓
Sonnet-3.5	8	✓	5	✗	5	✓	5	✗	5	✓	5	✗

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F31: VLMs tend only count the correct number of shapes when the image includes 5 circles. This shows that their predictions are biased toward the actual Olympic logo.

How many circles are in the image? Answer with only the number in numerical format.



GPT-4o	5	✓	5	✗	7	✗	12	✗	11	✗	5	✗
Gemini-1.5	5	✓	5	✗	5	✗	5	✗	5	✗	5	✗
Sonnet-3	3	✗	5	✗	5	✗	10	✗	10	✗	5	✗
Sonnet-3.5	4	✗	6	✓	6	✓	10	✗	9	✓	7	✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F32: Counting overlapped circles is not easy to VLMs regardless of circle colors, line widths, and resolutions. `Gemini-1.5` often predicts "5" regardless of the actual circle count, suggesting a strong bias towards the well-known Olympic logo.

J Counting the rows and columns of a grid task

J.1 Benchmark Construction Details

Our benchmark specifications consist of various parameters for the grid generation process.

- **Image size**: We include three different sizes {500, 1250, 2000} to create the grids on the image.
- **Number of rows/columns**: We choose a base size $N \in \{3, 4, 5, 6, 7, 8, 9\}$, and initialize the sizes to $N \times N$, $N \times N'$, and $N' \times N$ where $N' = N + 1$. We also create 10×10 grids to balance the square grids.
- **Line thickness**: We use a line thickness of 1% of the image size and 0.5% of the image size.
- **Entry**: Each table is generated in two versions, one that includes blank entries, and the second with random text entries.

We divide the image size by the number of rows and columns to find the coordinates for drawing the borderlines. Then, we draw the lines on the four edges of the image and draw the remaining lines in between. Our benchmark comprises 264 images of blank and text-containing grids (see Tab. T14 and Fig. F33).

Code The code is available at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/CountingRowsAndColumns/Grids.ipynb`.

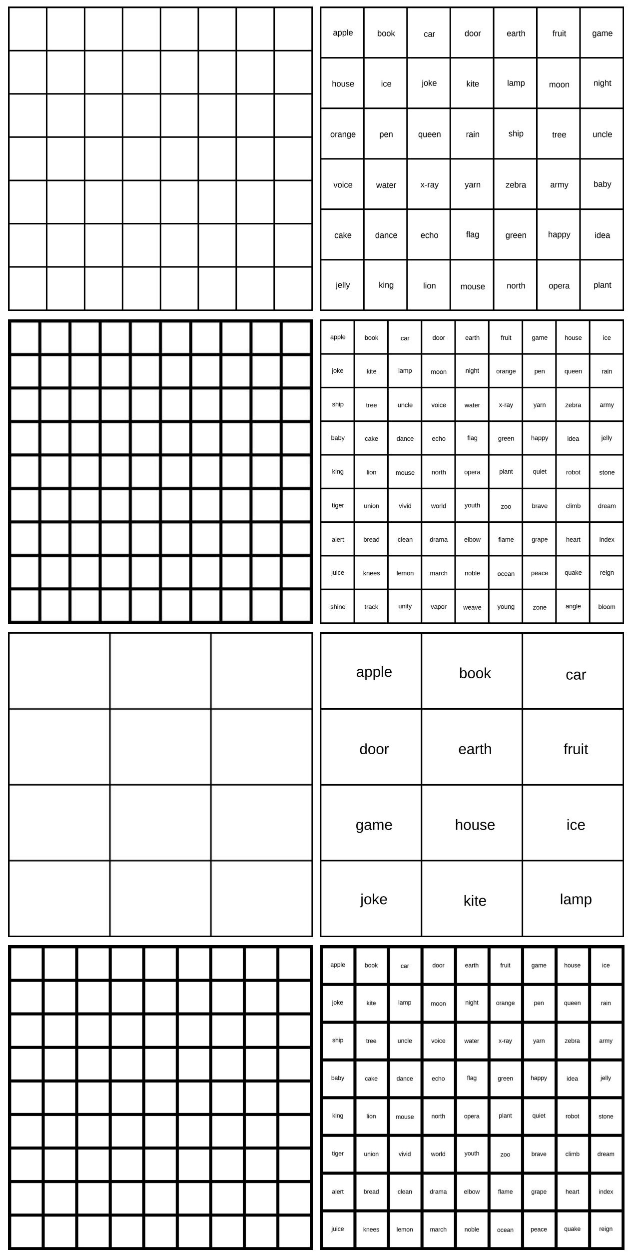
Prompts

1. *Count the number of rows and columns and answer with numbers in curly brackets. For example, rows={5} columns={6}*
2. *How many rows and columns are in the table? Answer with only the numbers in a pair (row, column), e.g., (5,6).*

Groundtruth answers include both the number of rows and columns. An answer is correct when both column and row counts are correctly predicted (random-chance accuracy is 1/22, *i.e.*, 4.55%).

Table T14: `BlindTest` consists of 264 empty and text-containing grids. We create equal number of images for each subcategory.

Parameter	**Values**	**Samples**	**Total Samples**
Cell	Empty	132	264
	Text	132	
Image size (C)	500px	44	132
	1250px	44	
	2000px	44	
Line thickness	$0.005 \times C$	66	132
	$0.01 \times C$	66	
Dimensions	3×3	6	132
	3×4	6	
	4×3	6	
	4×4	6	
	4×5	6	
	5×4	6	
	5×5	6	
	5×6	6	
	6×5	6	
	6×6	6	
	6×7	6	
	7×6	6	
	7×7	6	
	7×8	6	
	8×7	6	
	8×8	6	
	8×9	6	
	9×8	6	
	9×9	6	
	9×10	6	
	10×9	6	
	10×10	6	



(a) blank (b) text

Fig. F33: We create grids with various sizes and line thicknesses. Each grid has a blank (a) and text (b) version.

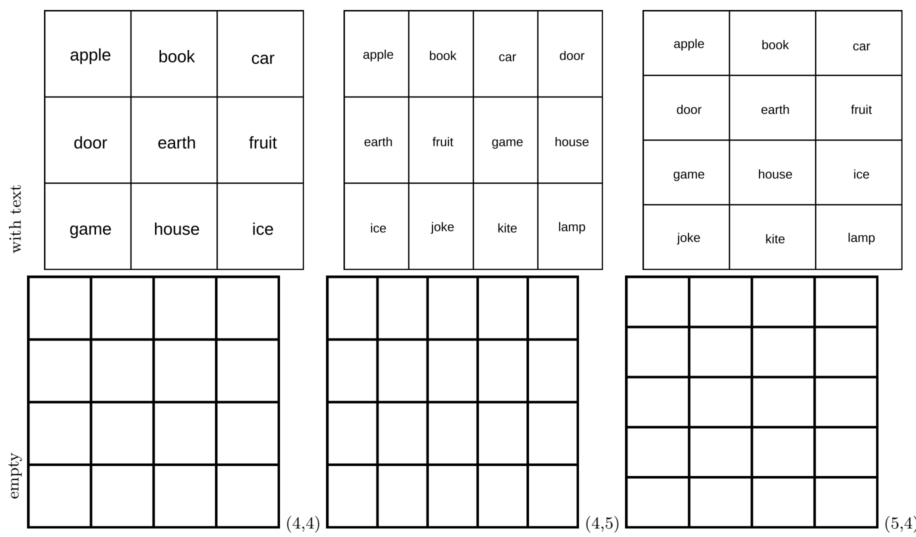


Fig. F34: Empty and text-containing grids are generated with various image sizes.

J.2 Finding: VLMs cannot reliably count either rows or columns

We expect counting rows and columns separately to be hard for the VLMs, thus we analyze counting the rows and grids individually to see how VLMs perform. As shown in Tab. T16, VLMs cannot count either rows or columns alone.

Table T15: Including text inside grids improves all model accuracies. `Sonnet-3.5`, yet, outperforms other models on both empty and text-containing grids.

Grid					Mean
Empty	26.13	26.51	25.00	59.84	34.37
Text	**53.03**	**52.27**	**47.34**	**88.68**	60.33
Mean	39.58	39.39	36.17	74.26	47.35

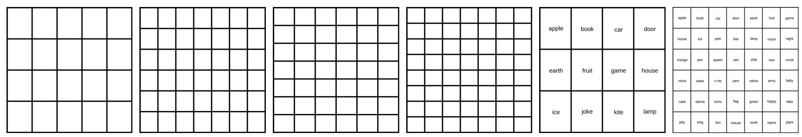
Table T16: Average row and column counting accuracy (%) of VLMs. VLMs perform better at counting columns (70.53% accuracy) than counting rows (60.83%)—both of which are far from the expected 100% accuracy.

Axis					Mean
Rows	65.54	52.95	42.19	82.64	60.83
Columns	58.42	58.50	74.65	90.54	70.53

J.3 Additional Examples

We show examples of models' responses to the counting the number of rows and columns task in Figs. F36 and F37.

Count the number of rows and columns and answer with numbers in curly brackets. For example, rows={5} columns={6}.

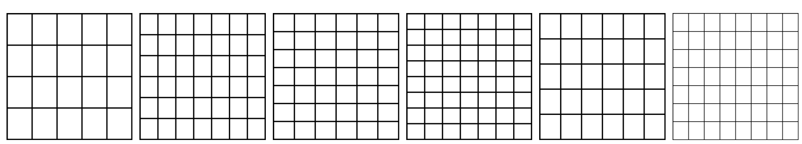


GT	4×5		6×7		7×6		8×7		3×4		6×7	
GPT-4o	4×4	✗	6×6	✗	7×7	✗	6×6	✗	3×4	✓	7×7	✗
Gemini-1.5	5×5	✗	6×6	✗	7×7	✗	10×10	✗	3×4	✓	7×8	✗
Sonnet-3	5×5	✗	7×8	✗	6×6	✗	9×9	✗	4×4	✗	7×7	✗
Sonnet-3.5	4×5	✓	6×7	✓	7×7	✗	8×7	✓	3×4	✓	7×7	✗

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F35: VLMs are often off by one or two in counting rows and columns in an empty grid. The same is true when a grid is small (*e.g.*, 3×4) and contains a word in each cell.

Count the number of rows and columns and answer with numbers in curly brackets. For example, rows={5} columns={6}



GT	4×5		6×7		7×6		8×7		5×6		7×8	
GPT-4o	4×4	✗	6×6	✗	7×7	✗	6×6	✗	6×6	✗	6×6	✗
Gemini-1.5	5×5	✗	6×6	✗	7×7	✗	10×10	✗	5×6	✓	10×10	✗
Sonnet-3	5×5	✗	7×8	✗	6×6	✗	9×9	✗	6×6	✗	9×12	✗
Sonnet-3.5	4×5	✓	6×7	✓	7×7	✗	8×7	✓	5×6	✓	8×8	✗

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F36: Examples from the benchmark show that models consistently fail at counting rows and columns of empty grids. Sonnet-3.5, however, is more consistent in the lower number of rows and columns.

How many rows and columns are in the table? Answer with only the numbers in a pair (row, column), e.g., (5,6).

GT	4×4		4×5		5×4		5×6		6×7		7×7	
GPT-4o	4×4	✓	4×5	✓	5×4	✓	5×6	✓	6×8	✗	7×8	✗
Gemini-1.5	4×4	✓	4×5	✓	5×4	✓	5×6	✓	6×8	✗	7×8	✗
Sonnet-3	4×4	✓	5×5	✗	5×4	✓	6×6	✗	7×7	✗	8×7	✗
Sonnet-3.5	4×4	✓	4×5	✓	5×4	✓	5×6	✓	6×7	✓	7×7	✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F37: When text is included in the cells of the grid, the performance of all VLMs improves, especially Sonnet-3.5.

K Following single-colored paths task

K.1 Benchmark Construction Details

Our subway-like graphs are generated using a set of parameters defining the characteristics of the plot.

- **Image size**: We use two different sizes {512, 1024}px for the images to include various resolutions.
- **Grid size**: We assume a hypothetical grid on the image that determines the position of the paths. We used an 18×18 grid, which means each path segment is $\frac{1}{18}$ of the image size.
- **Number of stations**: We use four station labels, {A, B, C, D}.
- **Starting points**: Each station in our maps has three different starting points which are exactly $\frac{1}{18}$ of the image size to one side of the stations.
- **Path thickness**: We use two line thicknesses, 10 and 20 pixels to have bold and light visualizations of the same path.
- **Number of paths**: Considering the number of starting points in our setup, each image can include stations from which exactly 1, 2, or 3 paths exit.

We keep generating the images until we have 15 samples for each number of paths which results in 180 images (see Tab. T17 and Fig. F38).

Code The code is available at `https://github.com/anguyen8/vision-llms-are-blind/blob/main/src/SubwayMap/SubwayMap.ipynb`

Prompts

1. *How many single-colored paths go from* ***{station 1}*** *to* ***{station 2}****? Answer with a number in curly brackets, e.g., {3}.*
2. *Count the one-colored routes that go from* ***{station 1}*** *to* ***{station 2}****. Answer with a number in curly brackets, e.g., {3}.*

where the two stations are different and sampled from { A, B, C, D }.
Groundtruth answers are $\in \{1, 2, 3\}$ (random-baseline accuracy: 33.33%). Note that, to make the task easier, 0 is excluded from the groundtruth set (*i.e.*, we never ask VLMs to count when no path exists between two given stations).

Table T17: We create 2 different image resolutions, and 3 various line widths to have balanced number of colored paths.

Parameter	Values	Samples	Total Samples
Image size	512px 1024px	90 90	180
Line thickness	10px 20px	90 90	180
Paths	1 2 3	60 60 60	180

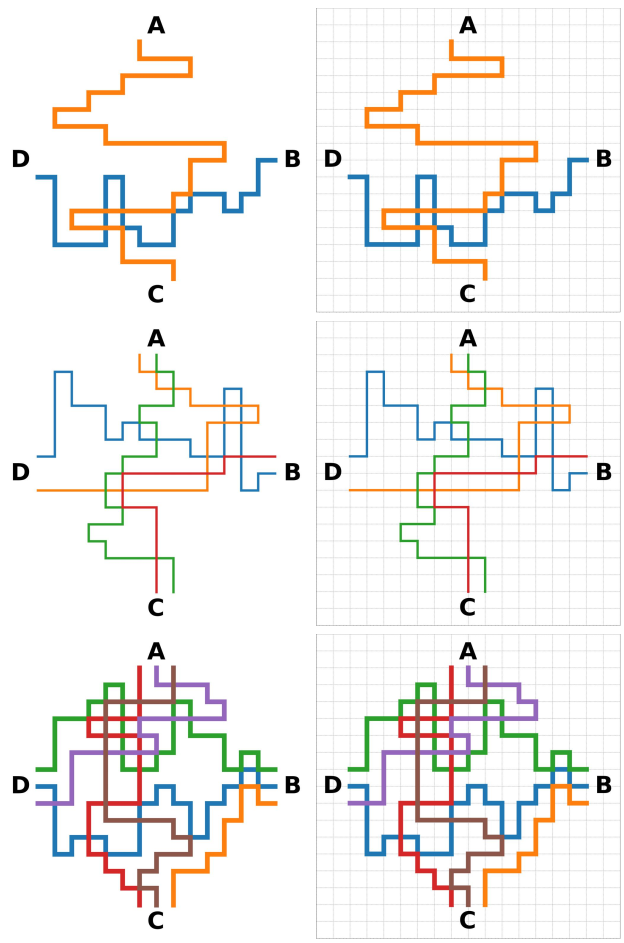


Fig. F38: Images in our benchmark (left) have exactly 1, 2, or 3 paths exiting each station. The hypothetical grids (right) are used when generating the paths.

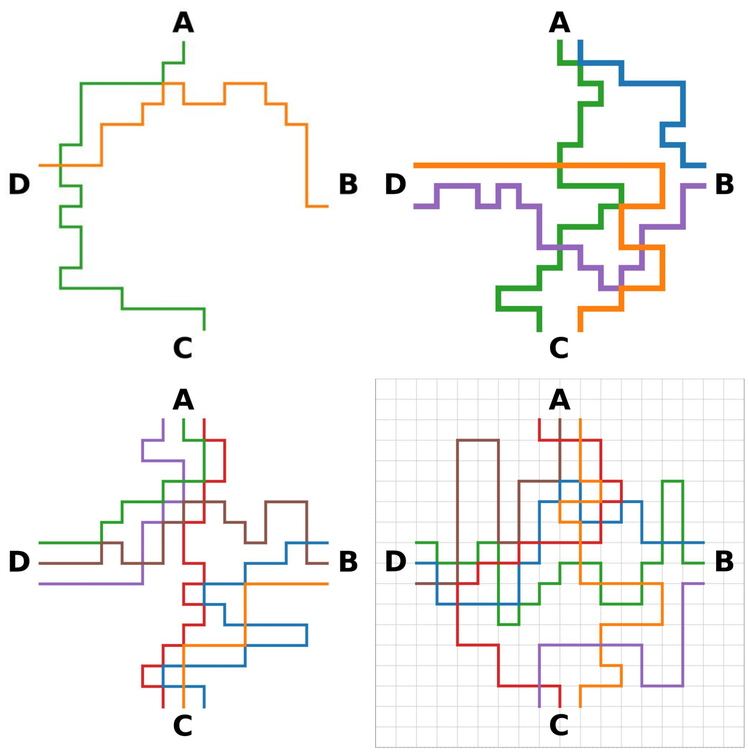


Fig. F39: On an invisible 18×18 grid (bottom right), we randomly generate random paths from one station to another. All stations have an equal $N = 1$, 2 or 3 outgoing paths.

K.2 Finding: VLMs fail to see the colored paths across various map complexities

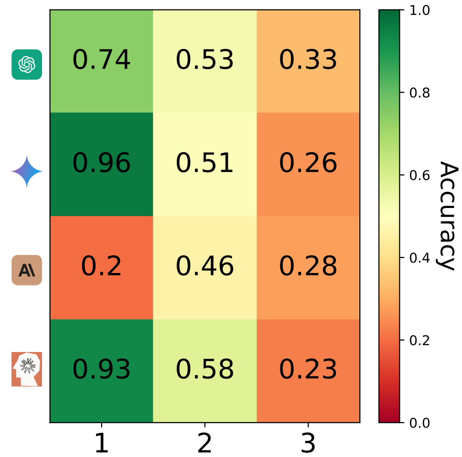
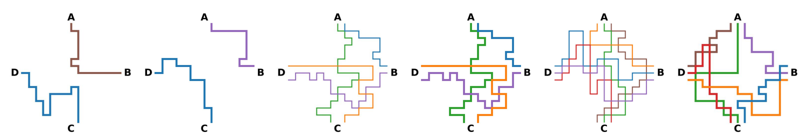


Fig. F40: VLM accuracy decreases as the number of paths increases from 1 to 3. Overall, counting colored paths poses a challenge to VLMs.

K.3 Additional Examples

We show examples of models' responses to the counting the number of single-colored connecting paths in Fig. F41.

How many single-color paths go from A to B? Answer with a number in curly brackets e.g. {3}

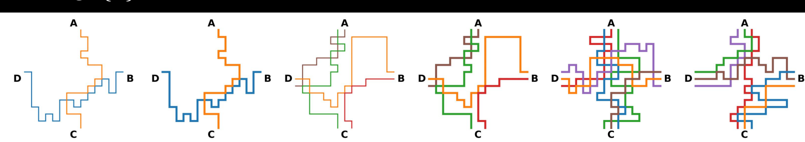


GPT-4o	1	✓	1	✓	1	✓	2	✗	2	✗	0	✗
Gemini-1.5	2	✗	2	✗	3	✗	3	✗	1	✓	6	✗
Sonnet-3	2	✗	2	✗	2	✗	2	✗	4	✗	2	✗
Sonnet-3.5	1	✓	2	✗	1	✓	3	✗	1	✓	3	✗

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F41: VLMs generally fail at images with more intersecting paths. As the number of paths increases, models tend to make an educated guess, especially `Gemini-1.5` and `Sonnet-3.5`.

How many single-color paths go from A to C? Answer with a number in curly brackets e.g. {3}.



GT	1		1		1		1		2		1	
GPT-4o	2	✗	0	✗	2	✗	3	✗	3	✗	1	✓
Gemini-1.5	2	✗	2	✗	4	✗	1	✓	2	✓	5	✗
Sonnet-3	2	✗	2	✗	3	✗	2	✗	3	✗	3	✗
Sonnet-3.5	1	✓	1	✓	3	✗	3	✗	2	✓	1	✓

GPT-4o Gemini-1.5 Sonnet-3 Sonnet-3.5

Fig. F42: Some VLMs (, ,) surprisingly fail in even extremely easy cases across both line widths (leftmost). VLMs tend to perform worse as the number of paths connecting stations increases.

L VLM failures on real-world data

List out stations on a subway path.

User: This is a subway map. List out the stations between Chungmuro and Wangsimni.

`Gemini-1.5`: Here are the stations between Chungmuro and Wangsimni on this map, assuming you're traveling along the dark blue line:

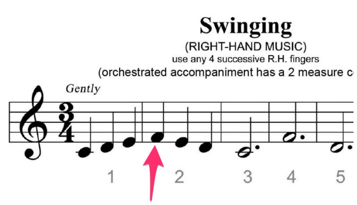
- Dongdaemun History & Culture Park ✗
- Sinseol-dong ✗
- Jegi-dong ✗

`Sonnet-3.5`: Based on the subway map shown, the stations between Chungmuro and Wangsimni are:

1. Chungmuro
2. Dongdaemun History & Culture Park
3. Sindang
4. Cheonggu
5. Sangwangsimni ✗
6. Wangsimni

Fig. F43: Examples of how VLMs fail on real-world data.

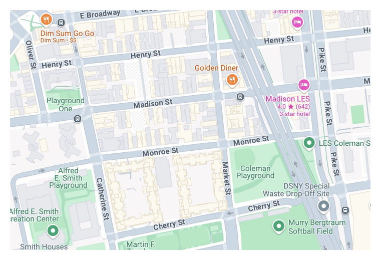
Reading music sheets



User: What is the name of this note?
`Sonnet-3.5`: The note indicated by the pink arrow is F. ✓ This can be determined by its position on the musical staff - it sits on the top line of the treble clef, which corresponds to the note F.

User: Is the note on a line?
`Sonnet-3.5`: Yes, this note is on a line. ✗ Specifically, it's on the third line from the bottom of the treble clef staff. ✗

Reading NYC street map



User: Is Catherine St intersecting with Market St?

`GPT-4o`: Yes, Catherine St does intersect with Market St as shown in the map ✗. They intersect near the Alfred E. Smith Playground and close to the Monroe St and Cherry St intersections. ✗

`Gemini-1.5`: No, Catherine St and Market St do not intersect in this map. ✓

Fig. F44: Examples of how VLMs fail on real-world data.

References

1. He, M., Liu, Y., Wu, B., Yuan, J., Wang, Y., Huang, T., Zhao, B.: Efficient multimodal learning from data-centric perspective. arXiv preprint arXiv:2402.11530 (2024)
2. McGuinness, P.: Gpt-4 system prompt revealed - by patrick mcguinness. https://patmcguinness.substack.com/p/gpt-4-system-prompt-revealed, (Accessed on 06/06/2024)
3. Mirza, M.J., Karlinsky, L., Lin, W., Doveh, S., Micorek, J., Kozinski, M., Kuhene, H., Possegger, H.: Meta-prompting for automating zero-shot visual recognition with llms. arXiv preprint arXiv:2403.11755 (2024)
4. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. Advances in neural information processing systems **35**, 27730–27744 (2022)