

# SketchAgent: Language-Driven Sequential Sketch Generation

## Supplementary Material

Yael Vinker<sup>1</sup> Tamar Rott Shaham<sup>1</sup> Kristine Zheng<sup>2</sup> Alex Zhao<sup>1</sup> Judith E Fan<sup>2</sup> Antonio Torralba<sup>1</sup>

<sup>1</sup>MIT

{yaelvink,tamarott,alexzhao,torralba}@mit.edu

<sup>2</sup>Stanford University

{jefan,kxzheng}@stanford.edu

<https://sketch-agent.csail.mit.edu/>

## Contents

<b>1. Technical Details</b>	<b>1</b>
<b>2. More Results and Analysis</b>	<b>2</b>
2.1. Quantitative Text-Conditioned Analysis . . .	2
2.2. Sequential sketching . . . . .	7
2.3. Human-Agent Collaborative Sketching . . .	12
2.4. Chat-Based Editing . . . . .	13
<b>3. Ablation Study</b>	<b>15</b>
<b>4. Prompts and More Results</b>	<b>16</b>

The supplementary material describes in detail the experiments from the main paper, as well as provides additional analysis results. **Please refer to the provided video for additional visual results.**

**IRB Disclosure** We received IRB approvals for all user studies, from all of the institutions involved. Accordingly, we took measures to ensure participant anonymity and refrained from showing them potentially offensive content.

## 1. Technical Details

We will publicly release the full source code, including our interactive platform. Our default backbone model is Claude3.5-Sonnet (version 20240620) [1]. We use the official API of Anthropic, with an average cost of \$0.05 per sketch. We employ CairoSVG [9] for rendering the SVG onto the canvas. Our output sketches are also provided in SVG format to facilitate further editing if needed. SketchAgent generates a complete sketch in approximately 20 seconds, with individual strokes in collaborative mode taking about 3.5 seconds each. For the CLIP zero-shot classification, we use the clip-vit-large-patch14 model from Hugging Face [4]. Our canvas is defined as a  $50 \times 50$  grid with numbers labeled on the bottom and left edges. Each cell corresponds to a patch of size  $12 \times 12$  pixels, chosen to ensure a clear display of the grid numbers along the edges. This configuration results in a  $612 \times 612$  pixels grid, with the drawing area confined to a  $600 \times 600$  pixel range. All



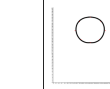








A smooth curve that starts at x8y6, passes through x6y7 and x6y10, ending at x8y11	To close this curve into an ellipse shape, you can add another curve:	To draw a large circle that starts at x25y44 and ends at x25y44, passing through the cells x32y41, x35y35, x31y29, x25y27, x19y29, x15y35, x18y41:		To draw a rectangle with four corners at x13y27, x24y27, x24y11, x13y11:
				
draw an upside-down "V" shape that starts at x13y27, ends at x24y27, with a pick at x18y37:	and then close it with a straight line from x13y27 to x24y27 to form a triangle: Points = ["x13y27", "x24y27"] t_values = [0.00,1.00]		To draw a small square with four corners at x26y25, x29y25, x29y21, x26y21:	To draw a single dot at x15y31 use:
				To draw a straight linear line that starts at x18y31 and ends at x35y14 use:
				

Figure 1. Visualization of single-stroke primitives used in the system prompt to introduce the grid and sketching language to the agent.

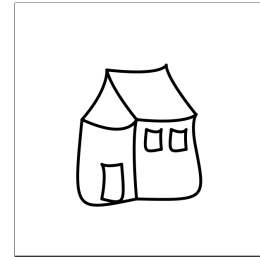


Figure 2. Visualization of the simple sketch of a house provided as an in-context example, represented with our sketching language through the user prompt.

prompts used in our method are provided in Figs. 37, 38 and 41. The examples provided to the agent in the system and user prompts are visualized in Fig. 1 and Fig. 2 respectively.

We use Claude3.5-Sonnet in its default settings, which results in significant variability in results, given the highly diverse nature of LLMs. For example, in Fig. 3, we present 12 sketches produced by our method for the concept “rabbit”, demonstrating high diversity in pose, structure, and quality. To generate variations in the experiments described in Section 5.1 of the main paper (where we applied our method 10 times per category), we use the default

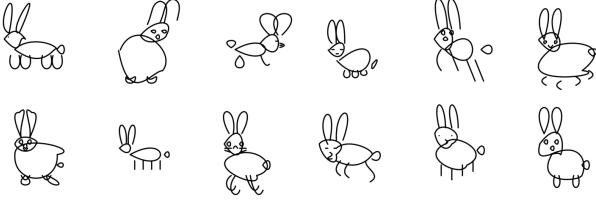


Figure 3. Sketch variability. Example of twelve different sketches produced for the concept “rabbit” by SketchAgent, with the same settings.

settings of Claude3.5-Sonnet. However, during controlled experimental conditions, we reduce variability by setting the temperature to 0 and top.k to 1, ensuring deterministic outputs. For general use, we recommend the stochastic version to encourage more varied and creative outputs.

## 2. More Results and Analysis

As described in Section 5.1 of the main paper, SketchAgent is capable of generating sketches for a wide range of concepts that extend beyond standard categories. Here we provide additional results to support this claim. We define three unique categories that require general knowledge: Scientific Concepts, Diagrams, and Notable Landmarks, and utilize ChatGPT-4o to produce 10 random textual concepts for each category, resulting in the following random concepts:

- **Scientific Concepts:** *Double-slit experiment, Pendulum motion, Photosynthesis, DNA replication, Newton’s laws of motion, Electromagnetic spectrum, Plate tectonics, Quantum entanglement, Cell division (mitosis), Black hole formation.*
- **Diagrams:** *Circuit diagram, Flowchart, Organizational chart, ER diagram (Entity-Relationship), Venn diagram, Mind map, Gantt chart, Network topology diagram, Pie chart, Decision tree.*
- **Notable Landmarks:** *Taj Mahal, Eiffel Tower, Great Wall of China, Pyramids of Giza, Statue of Liberty, Colosseum, Sydney Opera House, Big Ben, Mount Fuji, Machu Picchu.*

We generate five sketches for each concept (producing 50 sketches per category) by applying our method five times using its default settings. Figures 4 to 6 present the results for Scientific Concepts, Diagrams, and Notable Landmarks, respectively. The resulting sketches generally depict the concepts well, demonstrating diversity in the outputs. As can be seen, our method can generate a diverse set of different types and instances per concept (see *double-slit experiment*, *pendulum motion*, *Electromagnetic spectrum* in Fig. 4 and *Flowchart*, *Network typology diagram* in Fig. 5). Naturally, within each set, some concepts were depicted very successfully, while some outputs were less successful (e.g., Statue of Liberty, photosynthesis, pie chart).

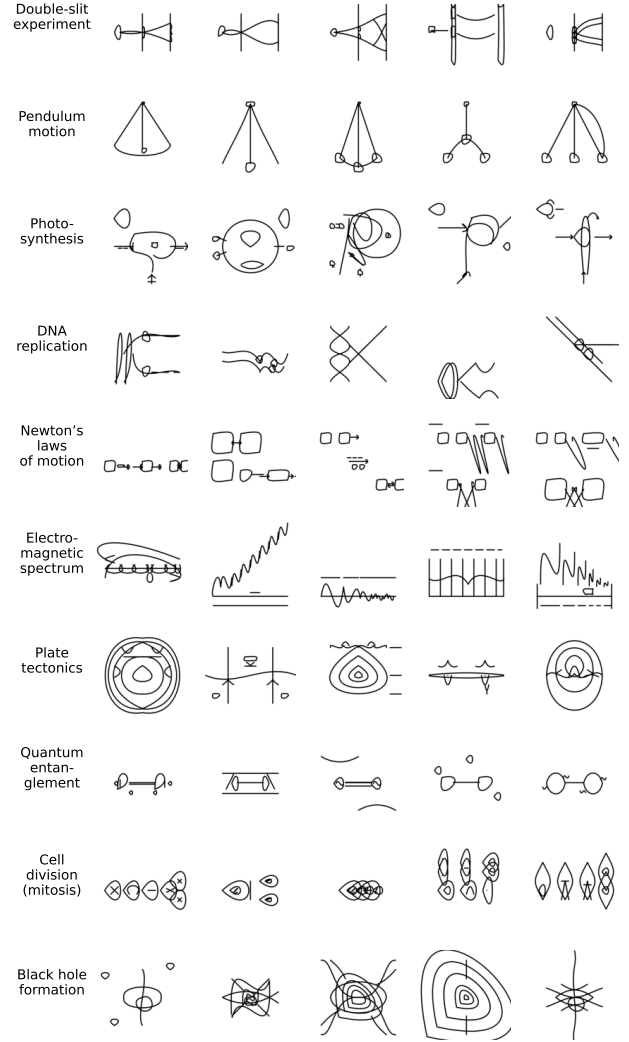


Figure 4. Randomly selected sketches of scientific concepts. Ten textual concepts were randomly selected using GPT-4o. Five sketches were generated per concept, showcasing the variability and diversity of the outputs.

### 2.1. Quantitative Text-Conditioned Analysis

In Section 5.1 of the main paper, we presented a quantitative analysis of text-conditioned sketch generation across 50 selected categories from the QuickDraw dataset [7]. Here, we provide additional details, visual examples, and further analysis of the experiment. We begin by providing further analysis of the CLIP classification rates of our default settings (Claude3.5-Sonnet) to explore recognition patterns. Figure 7 shows the confusion matrix (top 10 confused categories out of 50) for our set of 500 sketches. The most commonly confused classes are: “shark”, which was often misclassified as a “fish”, “octopus”, which was frequently identified as a “spider”; and “snake”, which was misclassified as a “squiggle”. These confused classes often fall within highly related classes (such as a fish and a shark, or a school bus and a

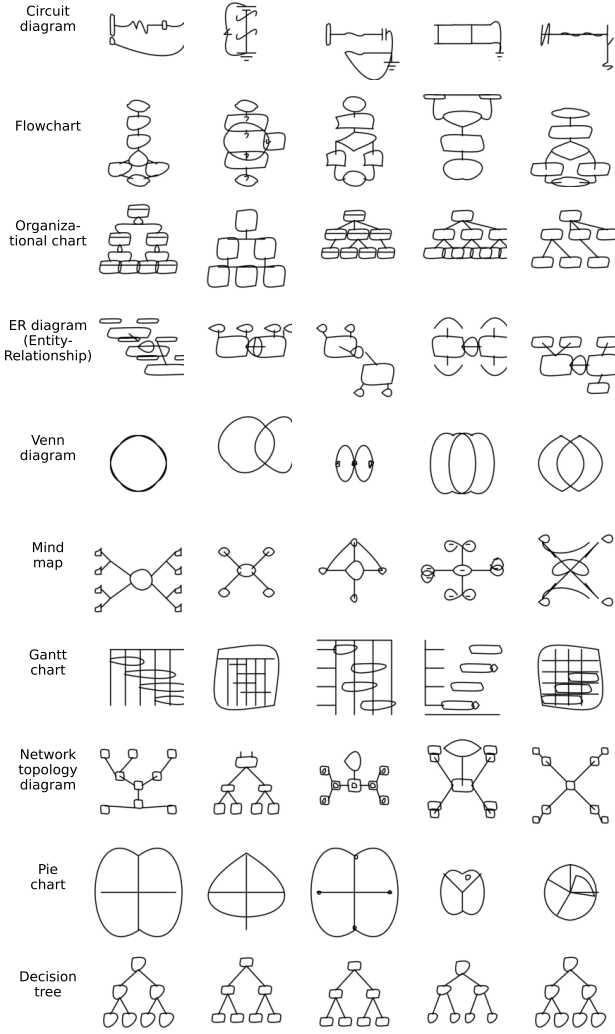


Figure 5. Randomly selected sketches of diagrams across fields. Ten textual concepts were randomly selected using GPT-4o. Five sketches were generated per concept, showcasing the variability and diversity of the outputs.

bus), suggesting that our method struggles with emphasizing distinctive features, likely due to its inherently abstract style. In Fig. 8, we visualize sketches from the six most confused classes with the correct class shown in green and the misclassified class shown in red. Figure 9 visualize the 10 top recognized classes. The class “fish” was correctly identified across all seeds, followed by “house”, “umbrella”, and “table”, which were correctly recognized in 90% of trials (9 out of 10). Recognition rates for other classes ranged from 70% to 20%. In Section 5.1 of the main paper, we compared the performance of different multimodal LLMs (GPT-4o-mini, GPT-4o, and Claude3-Opus) using our default prompts and settings. Figure 10 visualizes the eight most recognized classes across all backbone models. For this analysis, we select the top two recognized categories from each model and display the sketches with the highest classification probability for each.

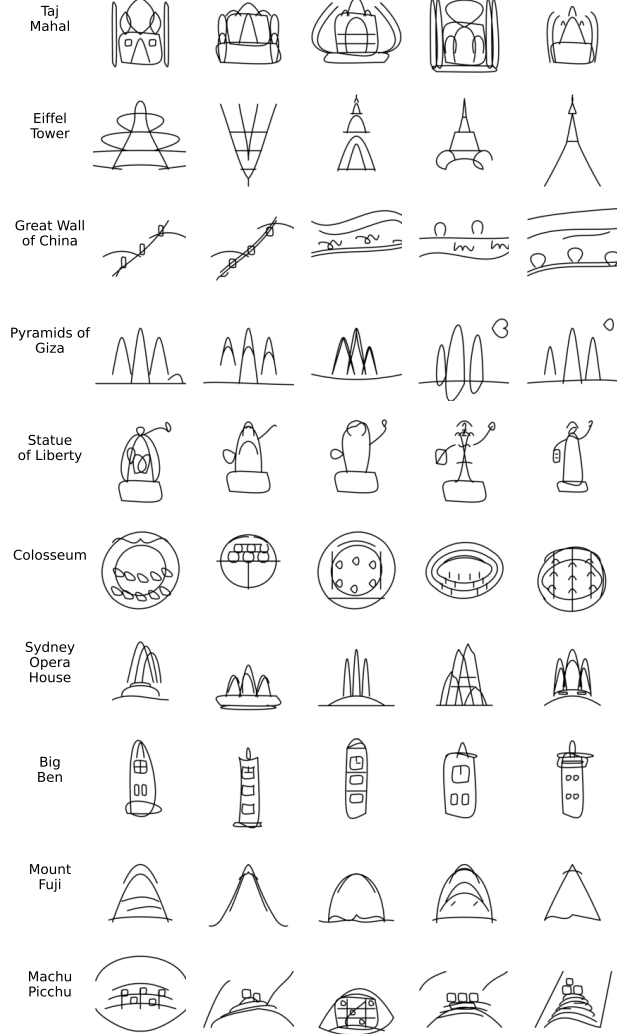


Figure 6. Randomly selected sketches of notable landmarks. Ten textual concepts were randomly selected using GPT-4o. Five sketches were generated per concept, showcasing the variability and diversity of the outputs.

ity for each. Note that some categories were at the top two of multiple models (such as house, fish, and eye), in that case, we select the next top recognized category. The chosen top two categories for each model are: GPT-4o: house and eye, GPT-4o-mini: table and goatee, Claude3-Opus: fish and airplane, Claude3.5-Sonnet: umbrella and bus. Similarly, Figure 11 highlights the least recognized categories, chosen using the same selection criteria. The chosen worst two categories for each model are: GPT-4o: snake and school bus, GPT-4o-mini: saxophone and raccoon, Claude3-Opus: octopus and dolphin, Claude3.5-Sonnet: shark and watermelon. Note that snake, octopus, and shark, were all confused under at least three of the four backbones. The visualizations align well with the quantitative results presented in Table 1 of the main paper. Among the Anthropic models, Claude3.5-Sonnet produces better sketches than Claude3-Opus, and among

True Label	ant	binoculars	bus	camel	dog	eye	fish	giraffe	headphones	mailbox	octopus	onion	pear	potato	school bus	shark	smiley face	snake	spider	squiggle
ant	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
binoculars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
camel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dog	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eye	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fish	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
giraffe	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
headphones	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mailbox	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
octopus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
onion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pear	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
potato	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
school bus	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shark	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
smiley face	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
snake	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
spider	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
squiggle	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Predicted Label	ant	binoculars	bus	camel	dog	eye	fish	giraffe	headphones	mailbox	octopus	onion	pear	potato	school bus	shark	smiley face	snake	spider	squiggle

Figure 7. Confusion matrix (showing top 10 confused classes) for the set of 500 sketches generated with SketchAgent default settings (Claude3.5-Sonnet) across 50 categories

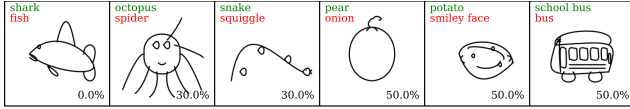


Figure 8. Visualization of sketches from the six most confused classes. The correct category is highlighted in green, while the misclassified category is highlighted in red.

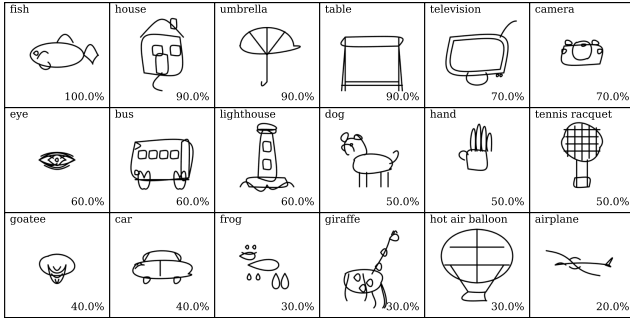


Figure 9. Visualization of the top recognized classes for the set of 500 sketches generated with our default settings (Claude3.5-Sonnet) across 50 categories.

the GPT models, GPT-4o outperforms GPT-4o-mini. Overall, the two best-performing backbone models are Claude3.5-Sonnet and GPT-4o. Interestingly, the sketching style differs between GPT-4o and Claude3.5-Sonnet. Although Claude3.5-Sonnet (our default backbone model) seems to yield the best results, this may be due to the fact that our method was primarily developed using this model. Consequently, the prompts we use were optimized for Claude3.5-Sonnet, and improved results for other models might be achievable with additional prompt engineering. We leave this exploration for future work.

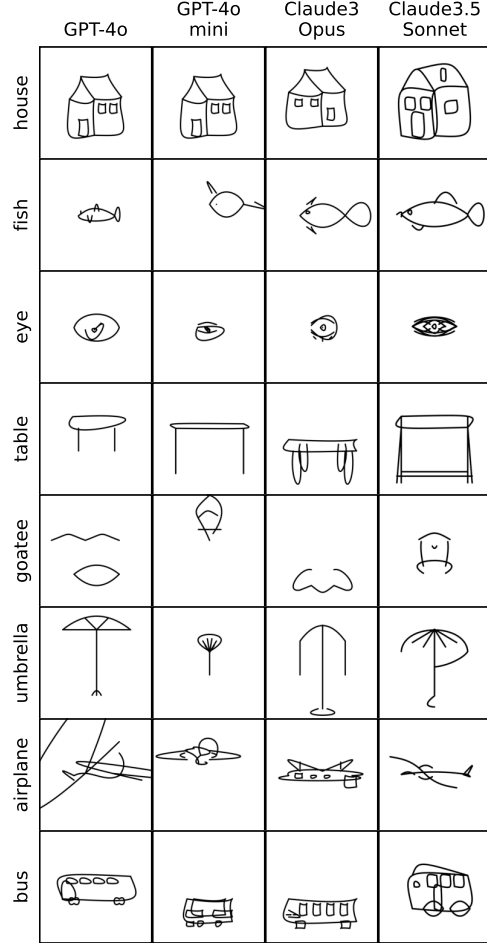


Figure 10. Visualization of sketches from the most recognized classes across all backbone models. The classes selected based on the two most recognizable classes in each model.

**SketchAgent using an open-source model** While open-source models currently lag behind commercial closed-source models, they are rapidly advancing in size and capability, showing significant potential for facilitating sketch generation.

We begin by experimenting with Llama-3.2-11B-Vision [3], a multimodal large language model developed by Meta AI, as SketchAgent’s backbone model. When used with our default prompts and framework, the model fails to generate meaningful sketches, frequently replicating the in-context example of a house provided in the user prompt (examples are shown in Fig. 12).

We therefore turn into exploring a larger available open-source model, Llama-3.1-405B-Instruct [3]. This model resulted in better sketches that manage to generalize well beyond the in-context example. We generated 500 random sketches and computed their classification rates using CLIP, as described in Section 5.1 of the main paper. The results yielded lower scores compared to commercial models, with an average Top-1 recognition accuracy of  $0.052 \pm 0.03$  and a Top-5 recognition accuracy of  $0.1 \pm 0.03$ . Visualizations of the top eight correctly classified classes are shown in Fig. 13, and the top eight most confused classes are presented



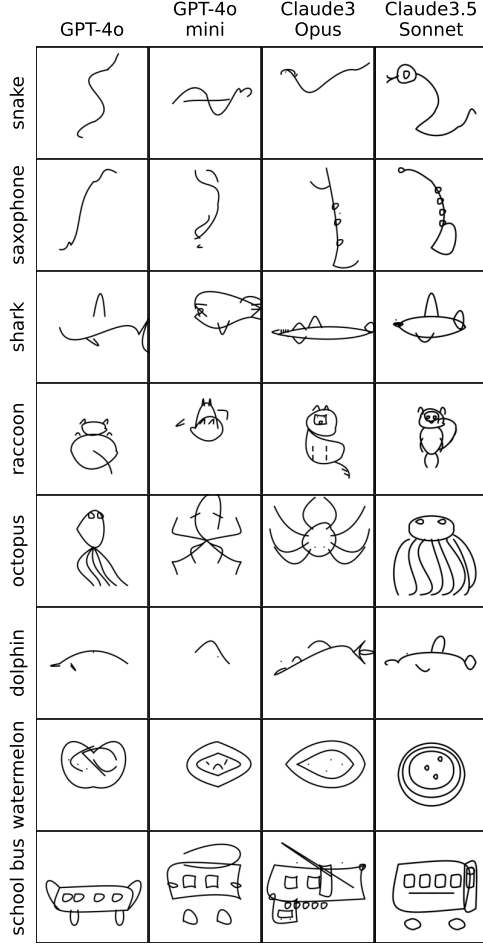


Figure 11. Visualization of sketches from the least recognized classes across all backbone models. The classes selected based on the two least recognizable classes in each model.



Figure 12. Sketches generated using Llama-3.2-11B-Vision as our backbone models. The model frequently replicates the in-context example of a house provided in the user prompt.

in Fig. 14. Despite the lower recognition rates, the generated sketches are reasonable and visually coherent, showing promise as open-source models continue to improve. This experiment demonstrates the potential for SketchAgent to be implemented using publicly available models. While its performance does not match that of our default backbone, SketchAgent can still function effectively with open-source models, albeit with a slight compromise in performance.

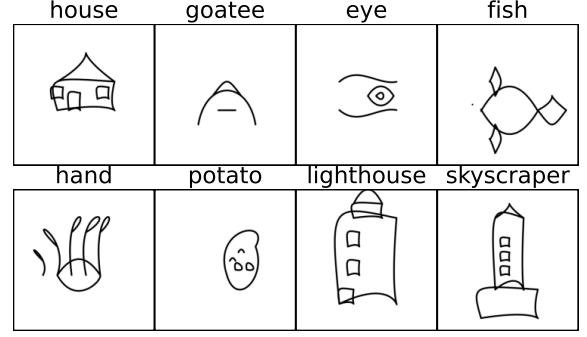


Figure 13. Visualization of the eight top recognized classes for the set of 500 sketches generated with Llama-3.1-405B-Instruct as our backbone model.

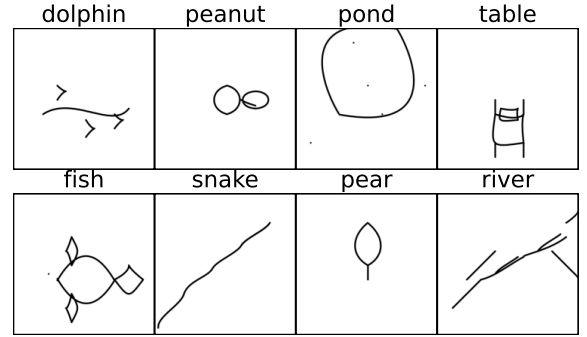


Figure 14. Visualization of sketches from the eight least recognized classes for the set of 500 sketches generated with Llama-3.1-405B-Instruct as our backbone model.

**Direct Prompting Analysis** In Section 5.1 of the main paper, we compared our method to directly prompting Claude3.5-Sonnet for generating SVGs with a sketch-like appearance. In Fig. 15, we extend this analysis by visualizing the results of direct prompting with the other backbone models used in the quantitative experiment. This demonstrates how different models respond to direct SVG generation prompts. We present examples for the concepts “giraffe” and “lighthouse”, using the following SVG generation prompt: “Write an SVG string of a <concept>.” For sketch-like SVGs, we used the same prompt as in the main paper (“Write an SVG string that draws a sketch of a <concept>. Use only black and white colors”). As shown, the outputs across all methods often feature uniform and precise geometric shapes (e.g., ellipses, triangles), which diverge from the natural variability and expressiveness characteristic of hand-drawn sketches. Interestingly, the SVGs generated by GPT-4o and Claude3.5-Sonnet appear more expressive and visually appealing compared to those produced by GPT-4o-mini and Claude3-Opus, aligning well with the performance differences observed in sketch generation.

**2AFC experiment** In section 5.1 of the main paper, we also presented a 2AFC experiment to evaluate how “human-like” our agent’s sketches appear compared to sketch-like SVGs generated with direct prompting and human sketches from the QuickDraw dataset. We utilize 50 sketches from 50 classes per method. We recruited a total of 150 workers through Amazon Mechanical Turk, each participating in 50 test sessions, as presented in Fig. 16. Before starting the test, workers were presented with instructions (Fig. 17). We filtered participants with a Mturk approval rate of 99.9% or higher and with a record of more than 1,000 surveys. Workers were paid \$0.5 for completing the full test.

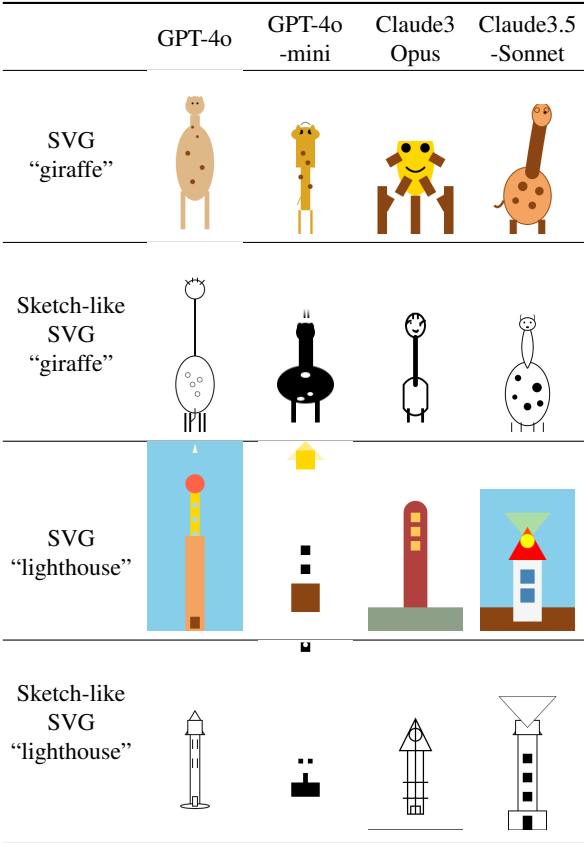


Figure 15. Direct prompting for SVG generation across different backbone models. The SVGs generated by GPT-4o and Claude3.5-Sonnet appear more expressive and visually appealing compared to those produced by GPT-4o-mini and Claude3-Opus, aligning well with the performance differences observed in sketch generation.

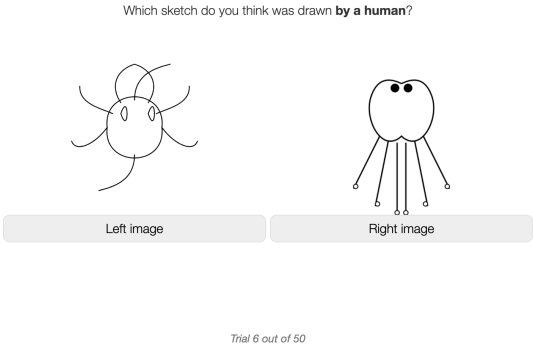


Figure 16. An example of our 2AFC session.

About this HIT:

- Please only participate in this HIT if you have normal vision.
- It should take about 5 minutes.
- You will take part in an experiment involving visual perception. You'll see a series of pairs of sketches. In each pair, one sketch was drawn by a human, and the other was drawn by a computer. Your task is to determine which sketch was drawn by a human.

Start!

EXAMPLE CONSENT TEXT: By making judgments about these images, you are participating in a study being performed by scientists at MIT. Your participation in this research is voluntary. You may decline further participation, at any time, without adverse consequences. Your anonymity is assured; the researchers who have requested your participation will not receive any personal information about you.

Figure 17. 2AFC instructions to users.

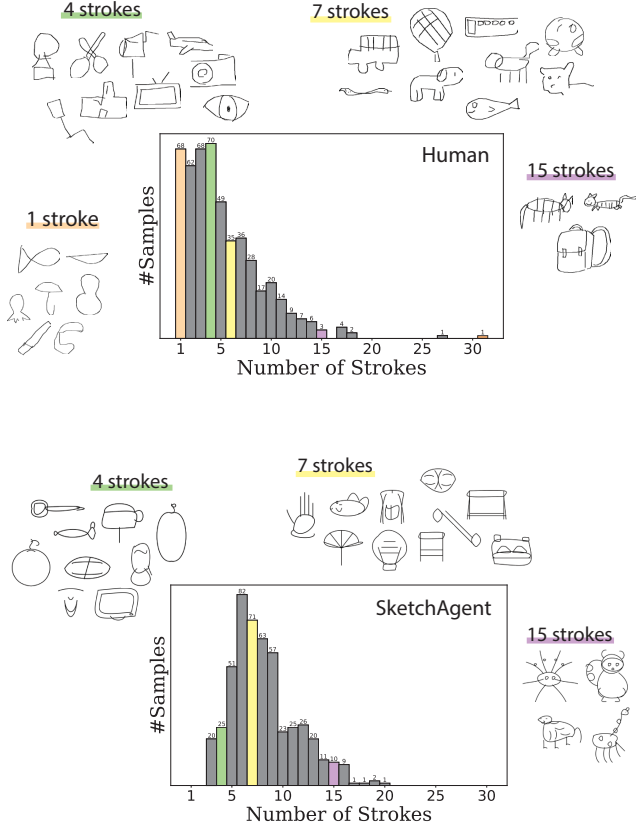


Figure 18. Distribution of human sketches [7] (top) and SketchAgent’s sketches (bottom) based on the number of strokes per sketch. Representative examples are shown for sketches drawn with 1, 4, 7, and 15 strokes. Notably, in the QuickDraw dataset, single-stroke sketches often consist of a single long continuous line.

## 2.2. Sequential sketching

In Section 5.2 of the main paper, we analyze the sequential nature of our generated sketches. In Figs. 23 to 25, we present additional visualizations of annotated sequential sketches of 48 randomly selected animals, with the presented sketches also chosen randomly. As illustrated, due to the extensive prior knowledge of the backbone model, SketchAgent provides meaningful textual annotations for each stroke and sketches in a logical order. Typically, more significant body parts, such as the head and body, are drawn first. We next provide more details and visualizations of the quantitative analysis shown in Figure 11 of the main paper. Figure 18 displays histograms of the number of strokes in QuickDraw sketches (top) and our generated sketches (bottom), as shown in the main paper. Alongside these histograms, we include visualizations of sketches drawn with 1, 4, 7, and 15 strokes. Notably, in the QuickDraw dataset, single-stroke sketches often consist of a single long continuous line, making them recognizable after the first stroke. In contrast, sketches with a larger number of strokes rarely feature long continuous lines. For such cases, the sequential process of adding strokes gradually makes the sketches recognizable

after several strokes. Figures 19 to 22 also demonstrates the sequential sketching process for both QuickDraw sketches and those generated by our method, providing a visual context for the trends observed in Figure 11.

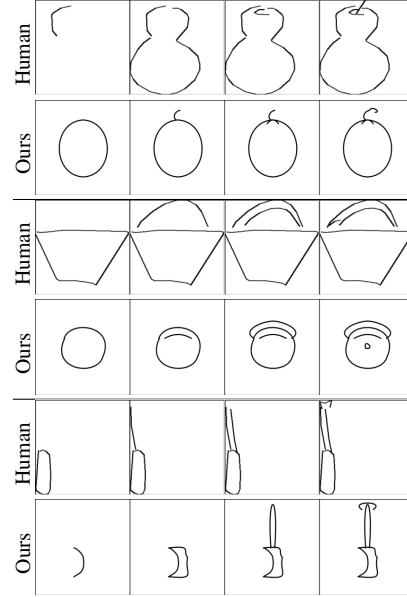


Figure 19. Sequential four-stroke sketches of a pear, purse, and screwdriver, created by humans [7] and by SketchAgent.

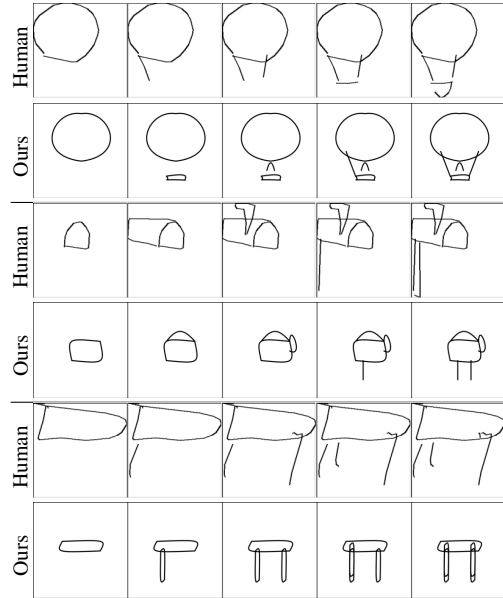


Figure 20. Sequential five-stroke sketches of a pear, purse, and screwdriver, created by humans [7] and by SketchAgent.

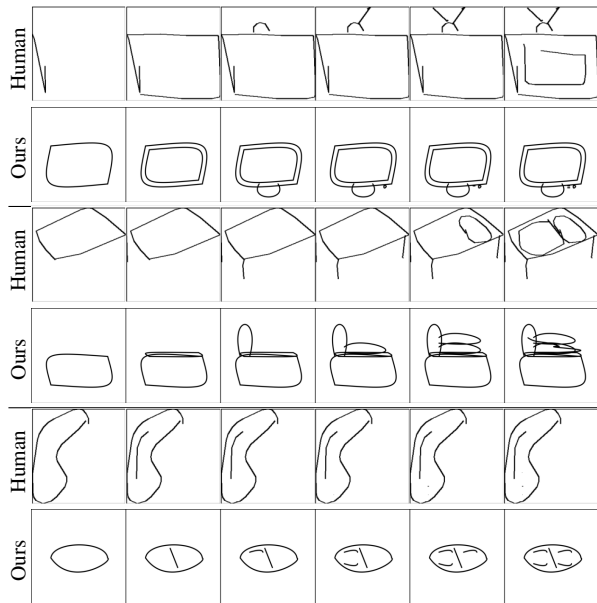


Figure 21. Sequential six-stroke sketches of a television, bed, and peanut, created by humans [7] and by SketchAgent.

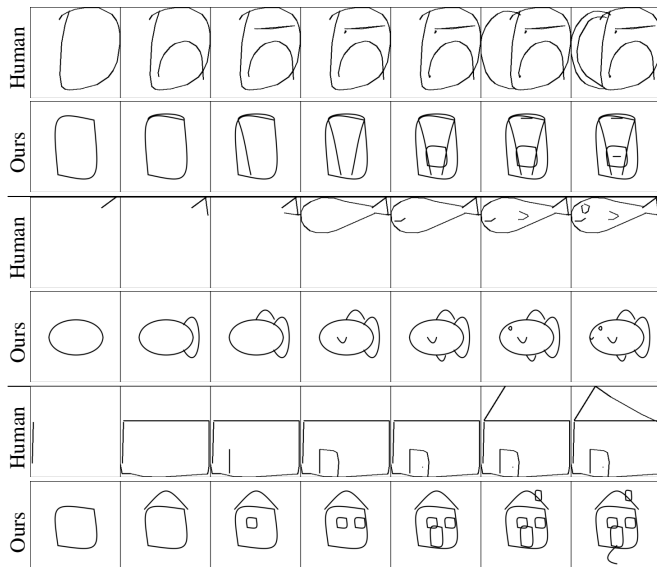


Figure 22. Sequential seven-stroke sketches of a backpack, fish, and house, created by humans [7] and by SketchAgent.

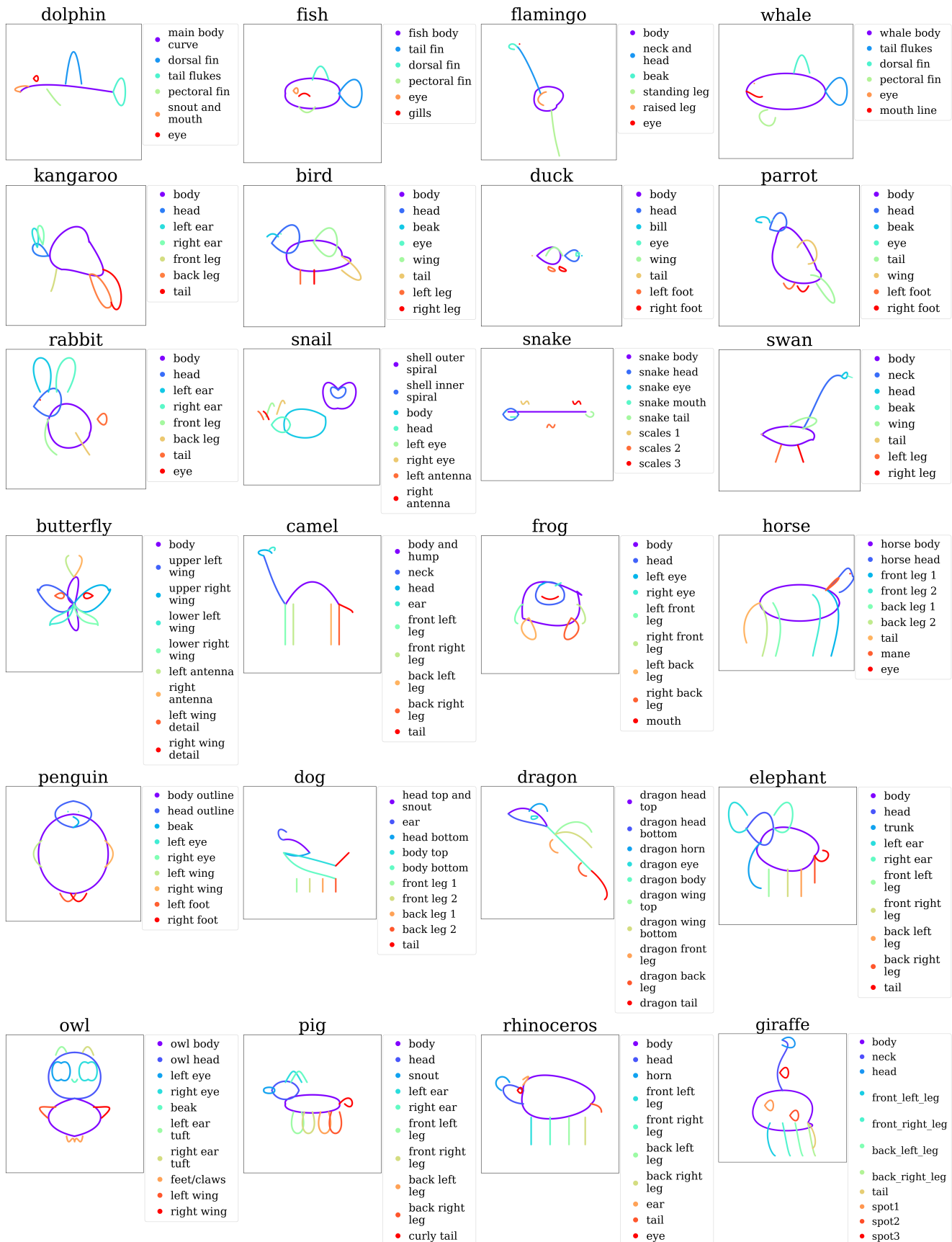


Figure 23



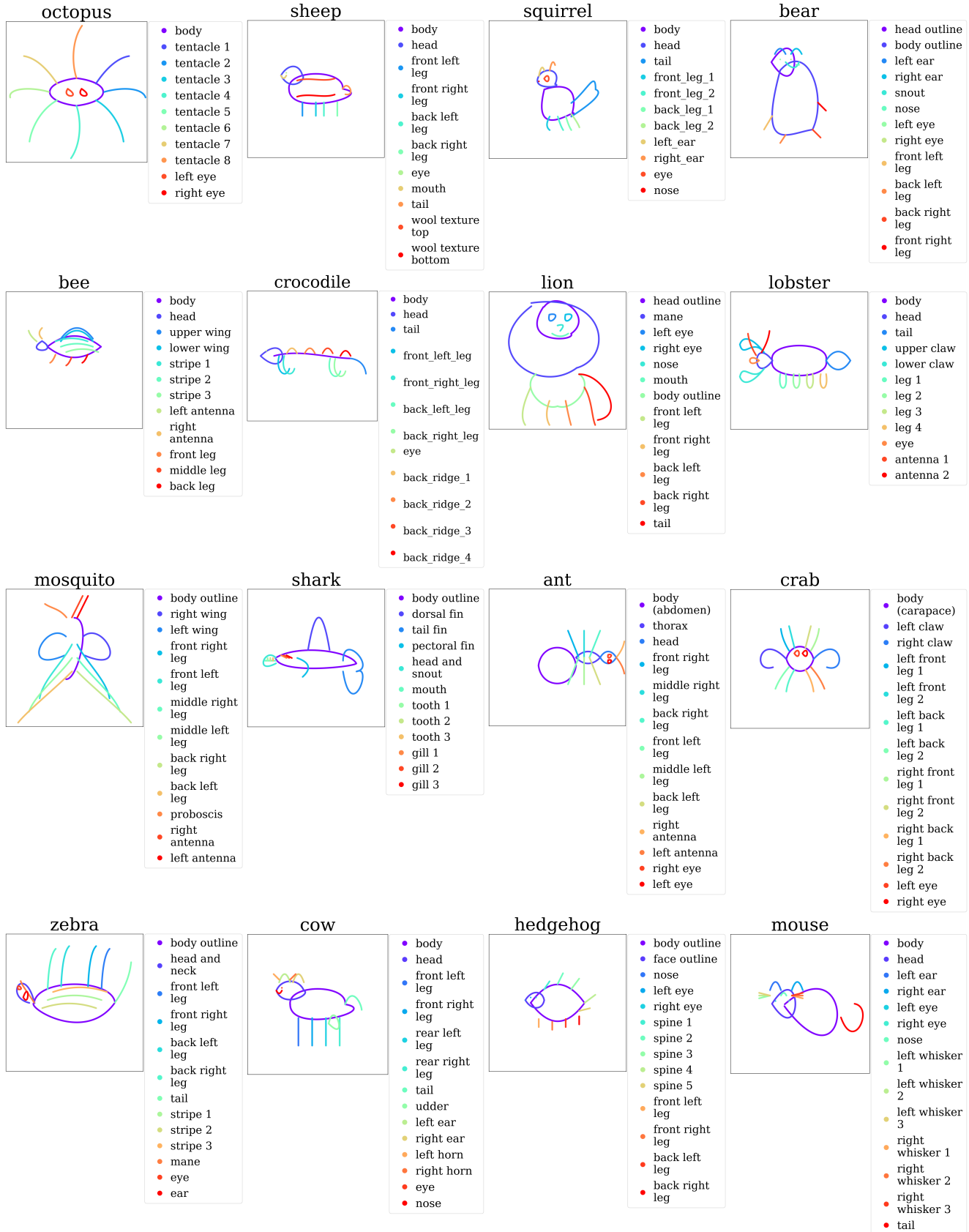


Figure 24

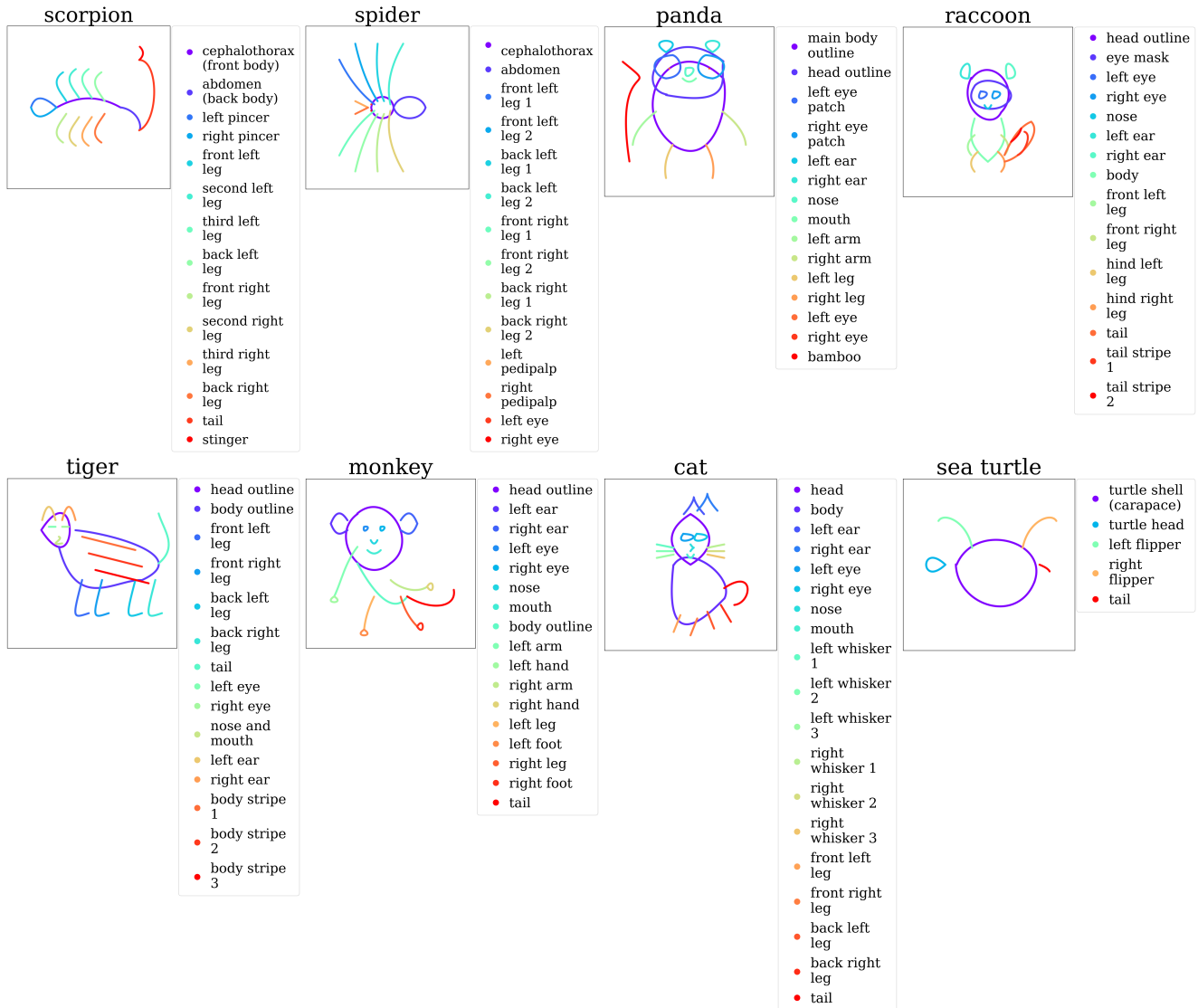


Figure 25

### 2.3. Human-Agent Collaborative Sketching

In Section 5.3 of the main paper, we demonstrate that humans and SketchAgent can effectively collaborate to produce meaningful sketches through genuine interaction. The sketching interface (Fig. 26) consists of a  $400 \times 400$  plain canvas shared between the user and the agent. It highlights the current concept to be sketched and displays the active sketching mode, which can be either *solo* or *collab*. Additionally, the interface includes a submit button that allows users to finalize the sketch when they consider it complete. In *solo* mode, users independently sketch the given concept using green strokes. In *collab* mode, users and SketchAgent take turns adding strokes, with user strokes displayed in green and agent strokes in pink. At the beginning of each session, users are provided with general instructions about the experiment and the types of sketches they will be asked to draw (Fig. 27). Specifically, they are instructed to create recognizable sketches, stroke by stroke, while minimizing the number of strokes by planning ahead. Next, users begin by sketching two warm-up concepts in both “solo” and “collab” modes to familiarize themselves with the web environment. Each session includes all eight primary concepts in a randomized order, resulting in a total of 10 sketches per user (including the two warm-up sketches). The concepts are as follows:

- **Warm up concepts:** *jellyfish, house*
- **Text concepts:** *butterfly, fish, rabbit, duck, sailboat, coffee mug, eyeglasses, car*

For each concept, participants sketched in both solo and collaboration modes, with the order of these modes randomized to mitigate potential biases. The 30 users are counterbalanced: 15 users produced the first stroke in collaboration with the agent (and all odd-numbered strokes thereafter), while the other 15 users produced the second stroke in collaboration with the agent (and all even-numbered strokes). In total we collected responses from 32 users, however, two users were excluded from the analysis due to incomplete sketching sessions, leaving a total of 30 users. In Fig. 32 we present examples of sketches from each mode, focusing on those with high recognition rates across categories. Solo sketches are shown in green, agent-only sketches in pink, and collaborative sketches are depicted with a combination of both colors. To analyze “collab” and “solo” sketches, we rendered all complete and partial sketches (agent-only and user-only strokes) from SVG to pixel images. We then utilized a CLIP zero-shot classifier, as described in the main paper, to evaluate how effectively each sketch represented the intended concept. Tab. 1 summarizes the results (as shown in the graph in Fig. 12B of the main paper). These results highlight that both users and the agent contributed meaningfully to the final “collab” sketches. Variants of collaborative sketches containing only the user’s strokes or only the agent’s strokes were found to contain substantially less semantic information about the intended concept compared to the complete collaborative sketches. Additionally, the average number of strokes per completed sketch was consistent across modes, indicating similar levels of complexity. Specifically, the average stroke counts were as follows: collaborative full sketches: 7.333; solo agent sketches: 7.321; solo user sketches: 7.708. This suggests that collaboration produces sketches with a level of detail comparable to those created independently.

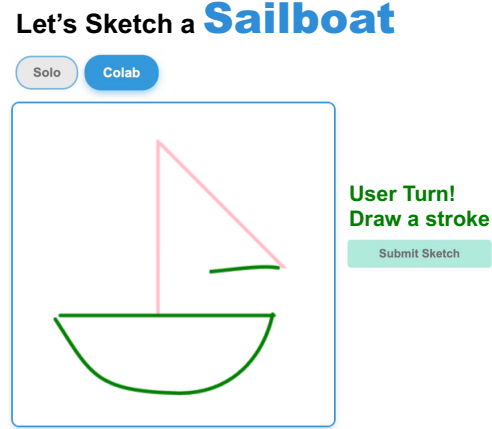


Figure 26. Screenshot of our web interface.

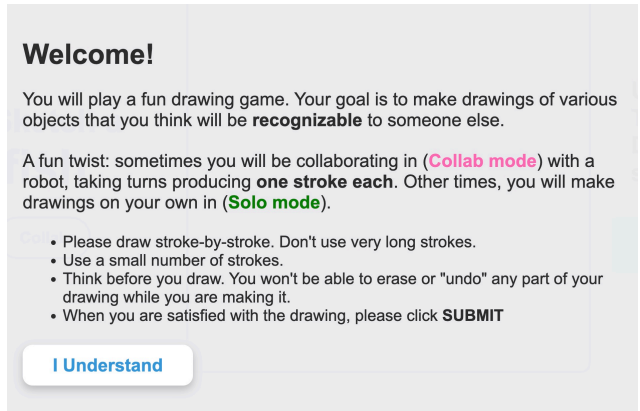


Figure 27. User instructions in the sketching interface.

We analyze the classification confusion patterns for collaborative and solo sketches (240 sketches each) in Figs. 29 and 30, revealing similar trends. For instance, a “coffee cup” was often misclassified as a “teapot”, a “car” was frequently identified as a “turtle”, and a “duck” was misclassified as a “bird”. Additionally, “car” sketches were sometimes mistaken for a “pickup truck”. In most cases, the misclassifications occur within closely related categories (e.g., “car” to “truck” or “pickup truck”) or among categories sharing similar visual structures (e.g., the rounded dome and four base components of a “car” resembling a “turtle”). This highlights a challenge in emphasizing distinctive features within specific categories, likely stemming from the inherently abstract nature of our sketches.

Figure 28 presents the recognition rates with 95% confidence interval (CI) error bars for each concept across all three sketching conditions: “collab” (blue), “solo-user” (green), and “solo-agent” (pink). Overall, the recognition rates for collaborative sketches are comparable to those produced by users alone or the agent alone for each unique category. Notably, sketches of “car” exhibit the lowest recognition rate across all conditions. This is likely due to confusion with semantically similar categories, such as “truck”, “pickup truck”, “airplane”, and “speedboat”, as indicated by the

Variation	Recognition Rate	95% CI
Collab full sketch	0.75	[0.61, 0.85]
Collab agent-only strokes	0.10	[0.06, 0.19]
Collab user-only strokes	0.13	[0.07, 0.23]

Table 1. Recognition rate and 95% CI across collaborative full and partial sketches. In collaborative sketches, keeping agent-only strokes or user-only strokes significantly reduces recognizability.

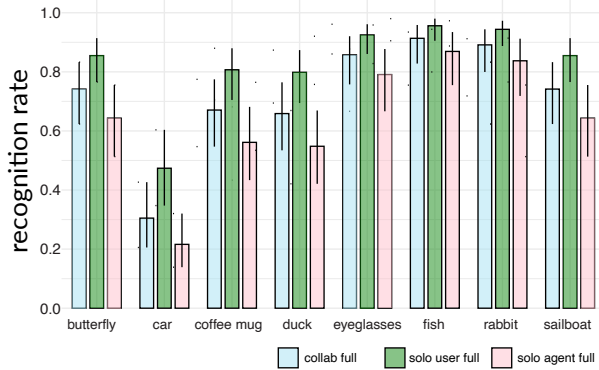


Figure 28. CLIP recognition rate by class for collaborative, solo user, and solo agent full sketches.

confusion matrices. Similarly, as discussed earlier, “coffee cup” and “duck” are frequently misclassified as related categories with overlapping visual features.

We observe that in some cases of collaborative sketching (14 out of 240 sketches), the agent-human pair faces challenges in interpreting each other’s intentions and the meanings of strokes. Achieving effective collaboration and communication between different parties [5] is a challenge that often requires prior planning, social reasoning, and repeated interactions to establish shared intentions and representations. These complex processes continue to be studied across various contexts, including in interactions between humans [6, 8, 10], between humans and agents [2], and between agents [11]. Fig. 31 highlights the few collaborative sketches where the CLIP classification is correct, but the agent and user appear to lack a shared understanding of different stroke groups, resulting in the conflicting creation of duplicate concept components (i.e. two heads).

## 2.4. Chat-Based Editing

In section 5.4 of the main paper we demonstrate chat-based editing using SketchAgent. Below, we provide more details about the implementation of the experiment we performed. To enable chat editing, we use the following prompt: “<editing instruction>. Describe the location of the added concepts first in <thinking> tags. Only provide the added strokes. Respond in the same format as before. Be concise.”, where <editing instruction> contains the desired edit such as “Add glasses to the given cat”. The chosen objects per category, as well as the editing prompts, are provided:

- **Animals:** *fish, bird, cat*. Editing instruction: “Add glasses”, “Add a hat”, “Add a skirt”.

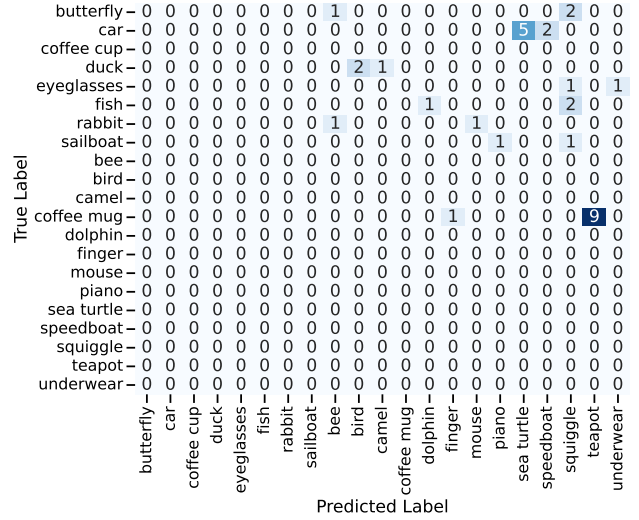


Figure 29. Confusion matrix from CLIP classification with categories from the QuickDraw dataset for 240 collaborative sketches across 8 categories.

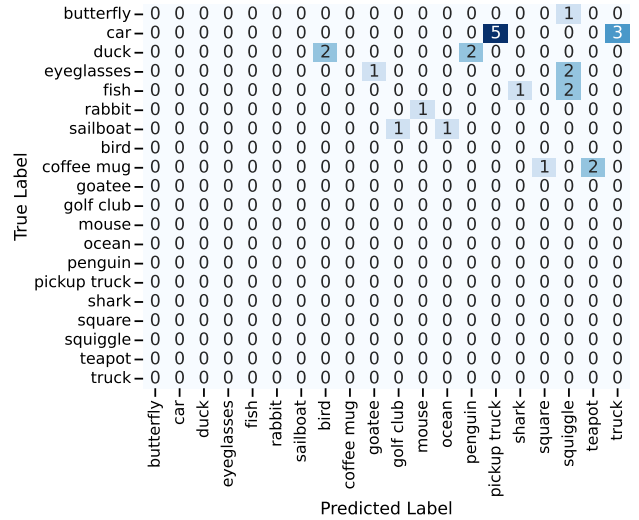


Figure 30. Confusion matrix from CLIP classification with categories from the QuickDraw dataset for 240 solo user sketches across 8 categories.

- **Outdoor:** *bus, building, boat*. Editing instruction: “Add a tree to the left of the <concept>”, “Add a sun on the top right, above the <concept>”, “Add another smaller <concept> to the right of this <concept>”.
- **Indoor:** *shelf, nightstand, table*. Editing instruction: “Add a coffee mug on the top of the <concept>”, “Add a lamp on the top of the <concept>”, “Add an indoor plant to the left of the <concept>”.

The resulting edited sketches are shown in Figure 33.

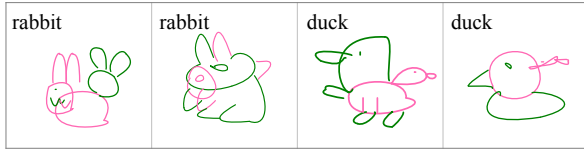


Figure 31. Examples of sketches created in “collab” mode that were correctly classified by CLIP but considered unsuccessful as collaborations due to conflicting agent-user interpretations of sub-components.

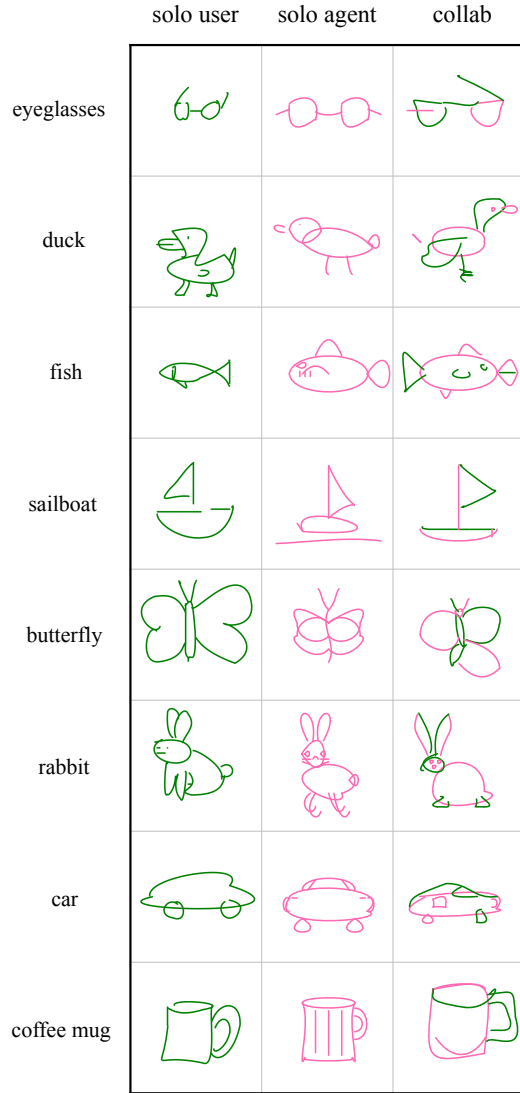


Figure 32. Examples of sketches from our collaborative human study that received high recognition rates. From left to right are sketches drawn in “solo” mode by users, “solo” mode by the agent, and collaboratively by both.

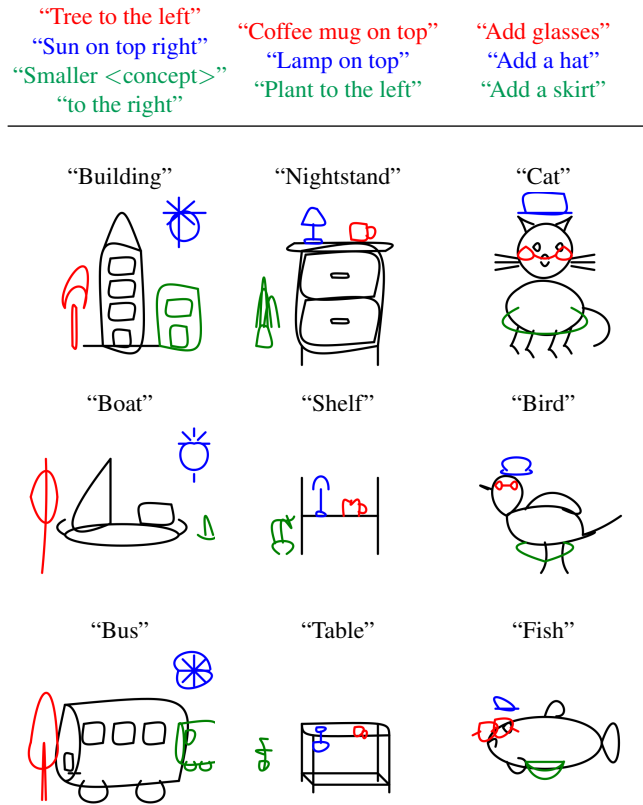


Figure 33. Chat-based sketch editing. We iteratively prompt SketchAgent to add objects to sketches through chat dialogues.



### 3. Ablation Study

In Section 6 of the main paper, we presented an ablation study by systematically removing key components of our method and computing the resulting classification rates. Here, we provide further analyses and discussions on the ablation study.

Table 2 in the main paper shows the CLIP classification rates for 500 sketches (across 50 categories) per experiment. In Fig. 34 we include a visualization of six sketches from six different concepts, covering both structures and animals. As shown, incorporating chain-of-thought reasoning and our in-context example of a house significantly enhances the quality of the results.

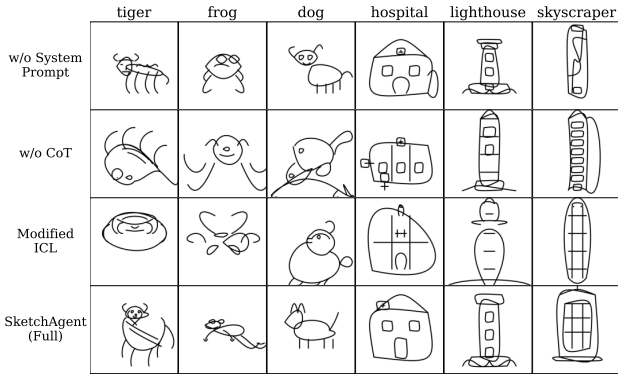


Figure 34. Visualization of sketches produced in different cases of our ablation study.

We find that the examples used in in-context learning (ICL) can influence both the quality and appearance of the generated sketches, suggesting an interesting direction for future research. Here, we analyze the impact of different types of in-context examples. To investigate whether the theme of the in-context example affects the output (e.g., whether using a house example aids in sketching related concepts like a hospital or if using a cat example helps with sketching other animals), we constructed an alternative sketch of a cat. This sketch used the same number of strokes as the house example to isolate the effect of the theme from complexity. We then applied our method using this alternative example in ICL. In Fig. 35 we illustrate the influence of different ICL examples on related concepts. The example used in each experiment is shown on the left, with the top figure presenting the effect on animal concepts and the bottom figure depicting the effect on structures. The results indicate that animal sketches are generally more influenced by an animal-based in-context example. For instance, the eyes in the generated sketches tend to resemble the eyes of the cat example more closely, while they vary more when a house example is used. However, there is no definitive conclusion regarding the overall quality or recognizability of these results. Conversely, for structures (bottom), the use of the cat example seems to result in smoother and more rounded shapes, while sketches generated using the house example generally appear more refined and cohesive.

We also examine the impact of example complexity, specifically how using a more detailed sketch with additional strokes affects the output. To test this, we enhanced the cat example by

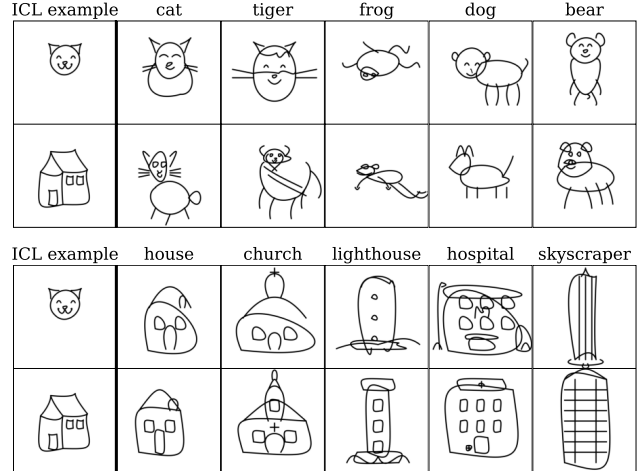


Figure 35. ICL example ablation study. We examine the impact of changing the concept in the ICL example (e.g., from a house to a cat) on the generation of related concepts. The example used in each experiment is shown on the left, with the top figure illustrating the effect on animal concepts and the bottom figure showing the effect on structural concepts.

adding more details and then applied our method with the new, more complex example. The results are presented in Fig. 36. As shown, when a more detailed example is used, the generated sketches tend to overfit, closely replicating the original cat sketch. In contrast, using a simpler example leads to greater variation in the output.

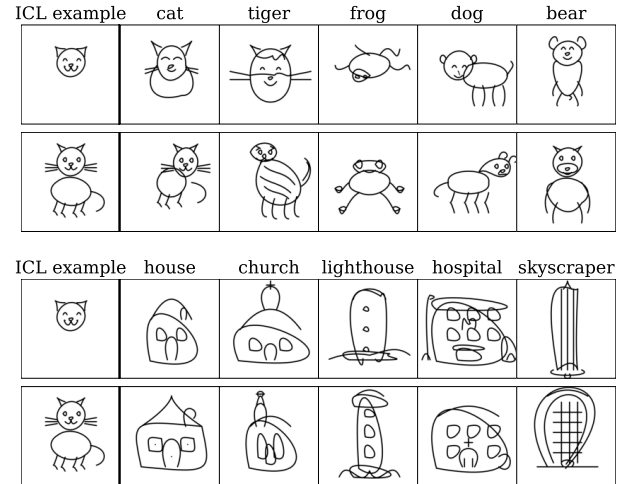


Figure 36. ICL example ablation study. We examine the impact of varying the complexity of the sketch presented in the ICL example while keeping the semantic concept (a cat) constant. The example used in each experiment is shown on the left, with the top figure illustrating the effect on animal concepts and the bottom figure showing the effect on structural concepts.

## 4. Prompts and More Results

We present the full prompts used in our system, as well as our randomly generated sketches used for the quantitative evaluation presented in Section 5.1 of the main paper, and the full set of sketches made by users and in collaborative mode from our human study.

## References

- [1] Anthropic. Claude. <https://www.anthropic.com/claude>, 2023. 1
- [2] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 13
- [3] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 4
- [4] Hugging Face. clip-vit-large-patch14. <https://huggingface.co/openai/clip-vit-large-patch14>. 1
- [5] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996. 13
- [6] Robert D. Hawkins, Megumi Sano, Noah D. Goodman, and Judith E. Fan. Visual resemblance and communicative context constrain the emergence of graphical conventions, 2021. 13
- [7] Jongejan Jonas, Rowley Henry, Kawashima Takashi, Kim Jongmin, and Fox-Gieg Nick. The Quick, Draw! - A.I. Experiment, 2016. 2, 7, 8
- [8] Günther Knoblich, Stephen Butterfill, and Natalie Sebanz. Chapter three - psychological research on joint action: Theory and data. In *Advances in Research and Theory*, pages 59–101. Academic Press, 2011. 13
- [9] Kozea. Cairosvg. <https://cairosvg.org/>, 2023. 1
- [10] William P. McCarthy, Robert D. Hawkins, Haoliang Wang, Cameron Holdaway, and Judith E. Fan. Learning to communicate about shared procedural abstractions, 2021. 13
- [11] Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):1504–1509, 2010. 13

You are an expert artist specializing in drawing sketches that are visually appealing, expressive, and professional.

You will be provided with a blank grid. Your task is to specify where to place strokes on the grid to create a visually appealing sketch of the given textual concept. The grid uses numbers (1 to res) along the bottom (x axis) and numbers (1 to res) along the left edge (y axis) to reference specific locations within the grid. Each cell is uniquely identified by a combination of the corresponding x axis numbers and y axis number (e.g., the bottom-left cell is 'x1y1', the cell to its right is 'x2y1'). You can draw on this grid by specifying where to draw strokes. You can draw multiple strokes to depict the whole object, where different strokes compose different parts of the object.

To draw a stroke on the grid, you need to specify the following:

Starting Point: Specify the starting point by giving the grid location (e.g., 'x1y1' for column 1, row 1).

Ending Point: Specify the ending point in the same way (e.g., 'xresyres' for column res, row res).

Intermediate Points: Specify at least two intermediate points that the stroke should pass through. List these in the order the stroke should follow, using the same grid location format (e.g., 'x6y5', 'x13y10' for points at column 6 row 5 and column 13 row 10).

Parameter Values (t): For each point (including the start and end points), specify a t value between 0 and 1 that defines the position along the stroke's path. t=0 for the starting point. t=1 for the ending point.

Intermediate points should have t values between 0 and 1 (e.g., "0.3 for x6y5, 0.7 for x13y10").

Examples:

To draw a smooth curve that starts at x8y6, passes through x6y7 and x6y10, ending at x8y11:

Points = ['x8y6', 'x6y7', 'x6y10', 'x8y11'] t\_values = [0.00,0.30,0.80,1.00]

To close this curve into an ellipse shape, you can add another curve:

Points = ['x8y11', 'x11y10', 'x11y7', 'x8y6'] t\_values = [0.00,0.30,0.70,1.00]

To draw a large circle that starts at x25y44 and ends at x25y44, passing through the cells x32y41, x35y35, x31y29, x25y27, x19y29, x15y35, x18y41: Points = ['x25y44', 'x32y41', 'x35y35', 'x31y29', 'x25y27', 'x19y29', 'x15y35', 'x18y41', 'x25y44'] t\_values = [0.00, 0.125, 0.25, 0.375, 0.50, 0.625, 0.75, 0.875, 1.00]

To draw non-smooth shapes (with corners) like triangles or rectangles, you need to specify the corner points twice with adjacent corresponding t values. For example, to draw an upside-down "V" shape that starts at x13y27, ends at x24y27, with a pick (corner) at x18y37: Points = ['x13y27', 'x18y37', 'x18y37', 'x24y27'] t\_values = [0.00,0.55,0.5,1.00]

To draw a triangle with corners at x10y29, x15y33, and x9y35, start with drawing a "V" shape that starts at x10y29, ends at x9y35, with a pick (corner) at x15y33:

Points = ['x10y29', 'x15y33', 'x15y33', 'x9y35'] t\_values = [0.00,0.55,0.5,1.00]

and then close it with a straight line from x13y27 to x24y27 to form a triangle:

Points = ['x13y27', 'x24y27'] t\_values = [0.00,1.00]

Note that for a triangle, the start and end points should be different from each other.

To draw a rectangle with four corners at x13y27, x24y27, x24y11, x13y11:

Points = ['x13y27', 'x24y27', 'x24y11', 'x24y11', 'x24y11', 'x13y11', 'x13y11', 'x13y27'] t\_values = [0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00]

To draw a small square with four corners at x26y25, x29y25, x29y21, x26y21:

Points = ['x26y25', 'x29y25', 'x29y25', 'x29y21', 'x29y21', 'x26y21', 'x26y21', 'x26y25'] t\_values = [0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00]

To draw a single dot at x15y31 use: Points = ['x15y31'] t\_values = [0.00]

To draw a straight linear line that starts at x18y31 and ends at x35y14 use: Points = ['x18y31', 'x35y14'] t\_values = [0.00, 1.00].

If you want to draw a big and long stroke, split it into multiple small curves that connect to each other. These instructions will define a smooth stroke that follows a Bezier curve from the starting point to the ending point, passing through the specified intermediate points. To draw a visually appealing sketch of the given object or concept, break down complex drawings into manageable steps. Begin with the most important part of the object, then observe your progress and add additional elements as needed. Continuously refine your sketch by starting with a basic structure and gradually adding complexity. Think step-by-step.

Figure 37. System prompt.

I provide you with a blank grid. Your goal is to produce a visually appealing sketch of a {concept}.  
Here are a few examples:

<examples>  
{gt-sketches}  
</examples>

You need to provide x-y coordinates that construct a recognizable sketch of a concept.  
You will receive feedback on your sketch and you will be able to adjust and fix it. Note that you will not have access to any additional resources. Do not copy previous sketches.

Think before you provide the x-y coordinates in <thinking> tags.  
First, think through what parts of the concept you want to sketch and the sketching order.  
Then, think about where the parts should be located on the grid.  
Finally, provide your response in <answer> tags, using your analysis.

Provide the sketch in the following format with the following fields:

<formatting>  
<concept>The concept depicted in the sketch.</concept>  
<strokes>This element holds a collection of individual stroke elements that define the sketch.  
Each stroke is uniquely identified by its own tag (e.g., <s1>, <s2>, etc.).  
Within each stroke element, there are three key pieces of information:  
<points>A list of x-y coordinates defining the curve. These points define the path the stroke follows.</points>  
<t\_values>A series of numerical timing values that correspond to the points. These values define the progression of the stroke over time, ranging from 0 to 1, indicating the order or speed at which the stroke is drawn.</t\_values>  
<id>A short descriptive identifier for the stroke, explaining which part of the sketch it corresponds to.</id>  
</strokes>  
</formatting>

Figure 38. User prompt. This prompt contains the specific sketching task as well as details about the expected format.

<example>

To draw a house, start by drawing the front of the house:

<concept>House</concept>

<strokes>

<s1>

<points>'x13y27', 'x24y27', 'x24y27', 'x24y11', 'x24y11', 'x13y11', 'x13y11', 'x13y27'</points>

<t\_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t\_values>

<id>house base front rectangle</id>

</s1>

<s2>

<points>'x13y27', 'x18y37', 'x18y37', 'x24y27'</points>

<t\_values>0.00,0.55,0.5,1.00</t\_values>

<id>roof front triangle</id>

</s2>

</strokes>

Next we add the house's right section:

<concept>House</concept>

<strokes>

<s1>

<points>'x13y27', 'x24y27', 'x24y27', 'x24y11', 'x24y11', 'x13y11', 'x13y11', 'x13y27'</points>

<t\_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t\_values>

<id>house base front rectangle</id>

</s1>

<s2>

<points>'x13y27', 'x18y37', 'x18y37', 'x24y27'</points>

<t\_values>0.00,0.55,0.5,1.00</t\_values>

<id>roof front triangle</id>

</s2>

<s3>

<points>'x24y27', 'x36y28', 'x36y28', 'x36y21', 'x36y21', 'x36y12', 'x36y12', 'x24y11'</points>

<t\_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t\_values>

<id>house base right section</id>

</s3>

<s4>

<points>'x18y37', 'x30y38', 'x30y38', 'x36y28'</points>

<t\_values>0.00,0.55,0.5,1.00</t\_values>

<id>roof right section</id>

</s4>

</strokes>

Now that we have the general structure of the house, we can add details to it, like windows and a door:

<concept>House</concept>

<strokes>

<s1>

<points>'x13y27', 'x24y27', 'x24y27', 'x24y11', 'x24y11', 'x13y11', 'x13y11', 'x13y27'</points>

<t\_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t\_values>

<id>house base front rectangle</id>

</s1>

<s2>

<points>'x13y27', 'x18y37', 'x18y37', 'x24y27'</points>

<t\_values>0.00,0.55,0.5,1.00</t\_values>

<id>roof front triangle</id>

</s2>

Figure 39. ICL example. This is the example of a sketch of a house we provide to the model.



```

<s3>
  <points>'x24y27', 'x36y28', 'x36y28', 'x36y21', 'x36y21', 'x36y12', 'x36y12', 'x24y11'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>house base right section</id>
</s3>
<s4>
  <points>'x18y37', 'x30y38', 'x30y38', 'x36y28'</points>
  <t_values>0.00,0.55,0.5,1.00</t_values>
  <id>roof right section</id>
</s4>
<s5>
  <points>'x26y25', 'x29y25', 'x29y25', 'x29y21', 'x29y21', 'x26y21', 'x26y21', 'x26y25'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>left window square</id>
</s5>
<s6>
  <points>'x31y25', 'x34y25', 'x34y25', 'x34y21', 'x34y21', 'x31y21', 'x31y21', 'x31y25'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>right window square</id>
</s6>
<s7>
  <points>'x17y11', 'x17y18', 'x17y18', 'x21y18', 'x21y18', 'x21y11', 'x21y11', 'x17y11'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>front door</id>
</s7>
</strokes>
and here is the complete example:
<concept>House</concept>
<strokes>
  <s1>
    <points>'x13y27', 'x24y27', 'x24y27', 'x24y11', 'x24y11', 'x13y11', 'x13y11', 'x13y27'</points>
    <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
    <id>house base front rectangle</id>
  </s1>
  <s2>
    <points>'x24y27', 'x36y28', 'x36y28', 'x36y21', 'x36y21', 'x36y12', 'x36y12', 'x24y11'</points>
    <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
    <id>house base right section</id>
  </s2>
  <s3>
    <points>'x13y27', 'x18y37', 'x18y37', 'x24y27'</points>
    <t_values>0.00,0.55,0.5,1.00</t_values>
    <id>roof front triangle</id>
  </s3>
  <s4>
    <points>'x18y37', 'x30y38', 'x30y38', 'x36y28'</points>
    <t_values>0.00,0.55,0.5,1.00</t_values>
    <id>roof right section</id>
  </s4>

```

Figure 40. ICL example. This is the example of a sketch of a house we provide to the model.

```

<s5>
  <points>'x26y25', 'x29y25', 'x29y25', 'x29y21', 'x29y21', 'x26y21', 'x26y21', 'x26y25'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>left window square</id>
</s5>
<s6>
  <points>'x31y25', 'x34y25', 'x34y25', 'x34y21', 'x34y21', 'x31y21', 'x31y21', 'x31y25'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>right window square</id>
</s6>
<s7>
  <points>'x17y11', 'x17y18', 'x17y18', 'x21y18', 'x21y18', 'x21y11', 'x21y11', 'x17y11'</points>
  <t_values>0.00,0.3,0.25,0.5,0.5,0.75,0.75,1.00</t_values>
  <id>front door</id>
</s7>
</strokes>
</example>

```

Figure 41. ICL example. This is the example of a sketch of a house we provide to the model.

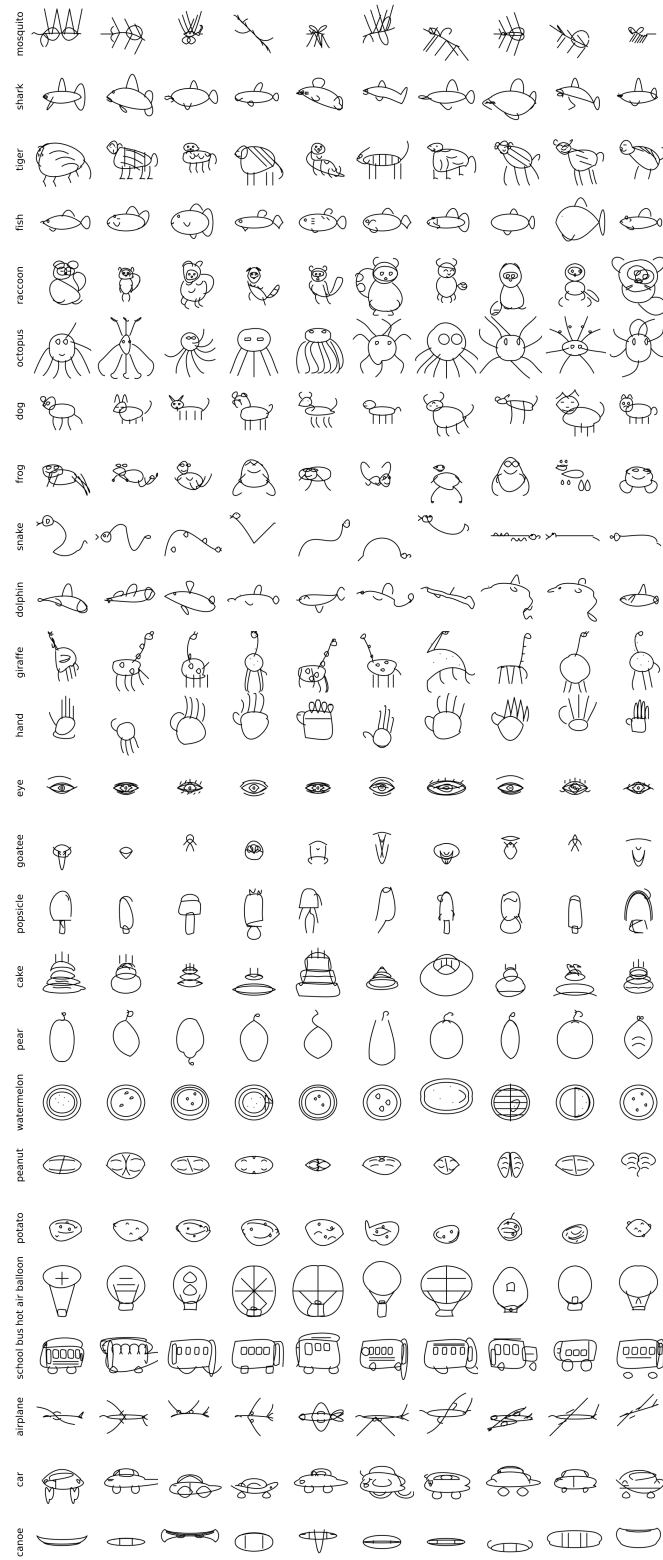


Figure 42. Randomly generated sketches used in the quantitative analysis (ten sketches per category).

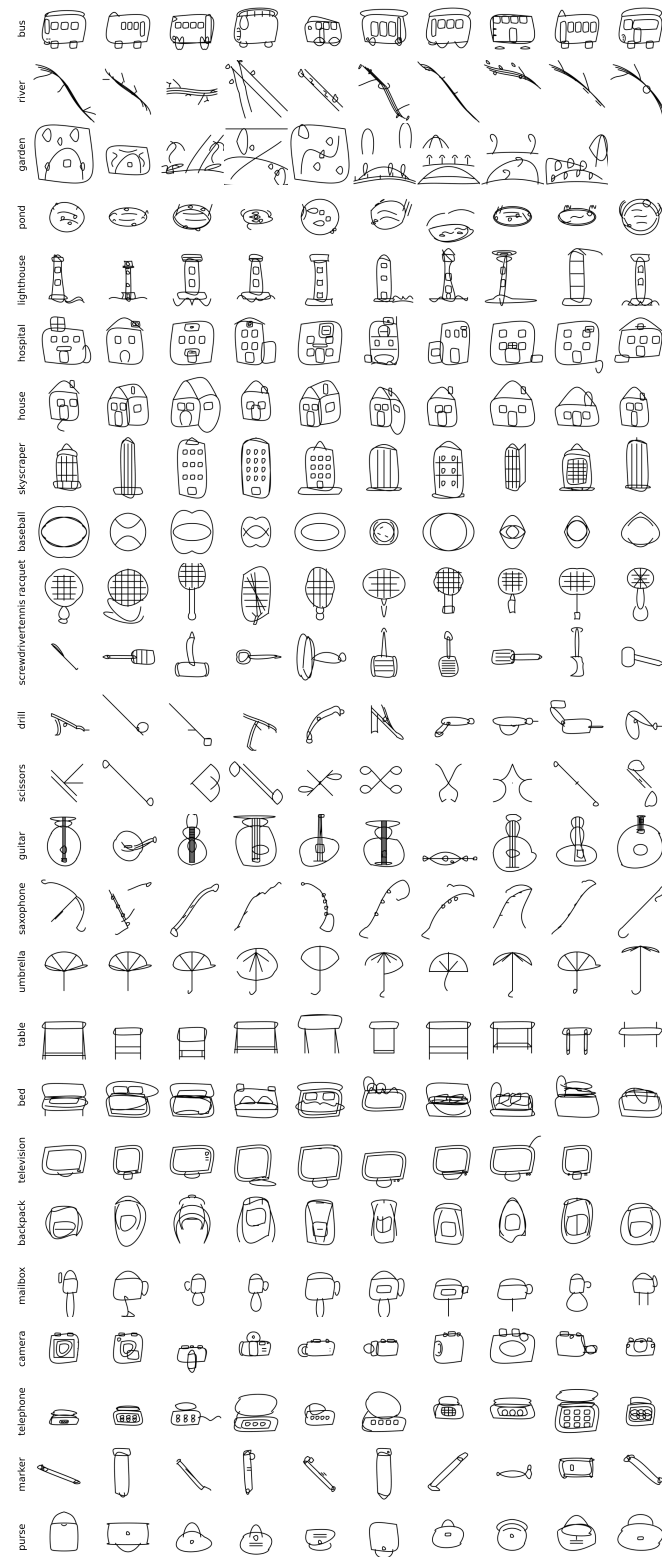


Figure 43. Randomly generated sketches used in the quantitative analysis (ten sketches per category).

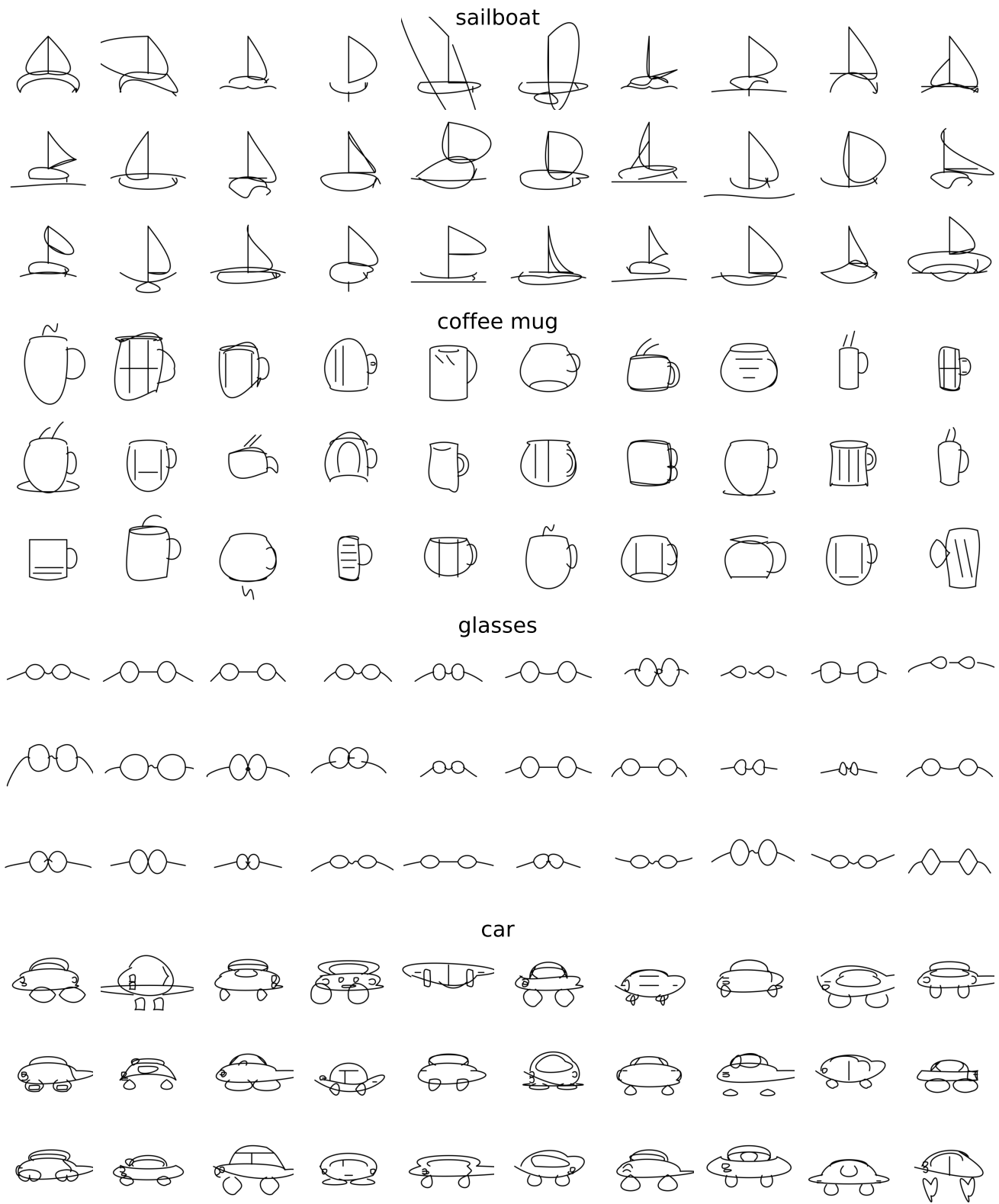


Figure 44. Sketches generated by SketchAgent for the eight categories of our human-agent collaborative study.



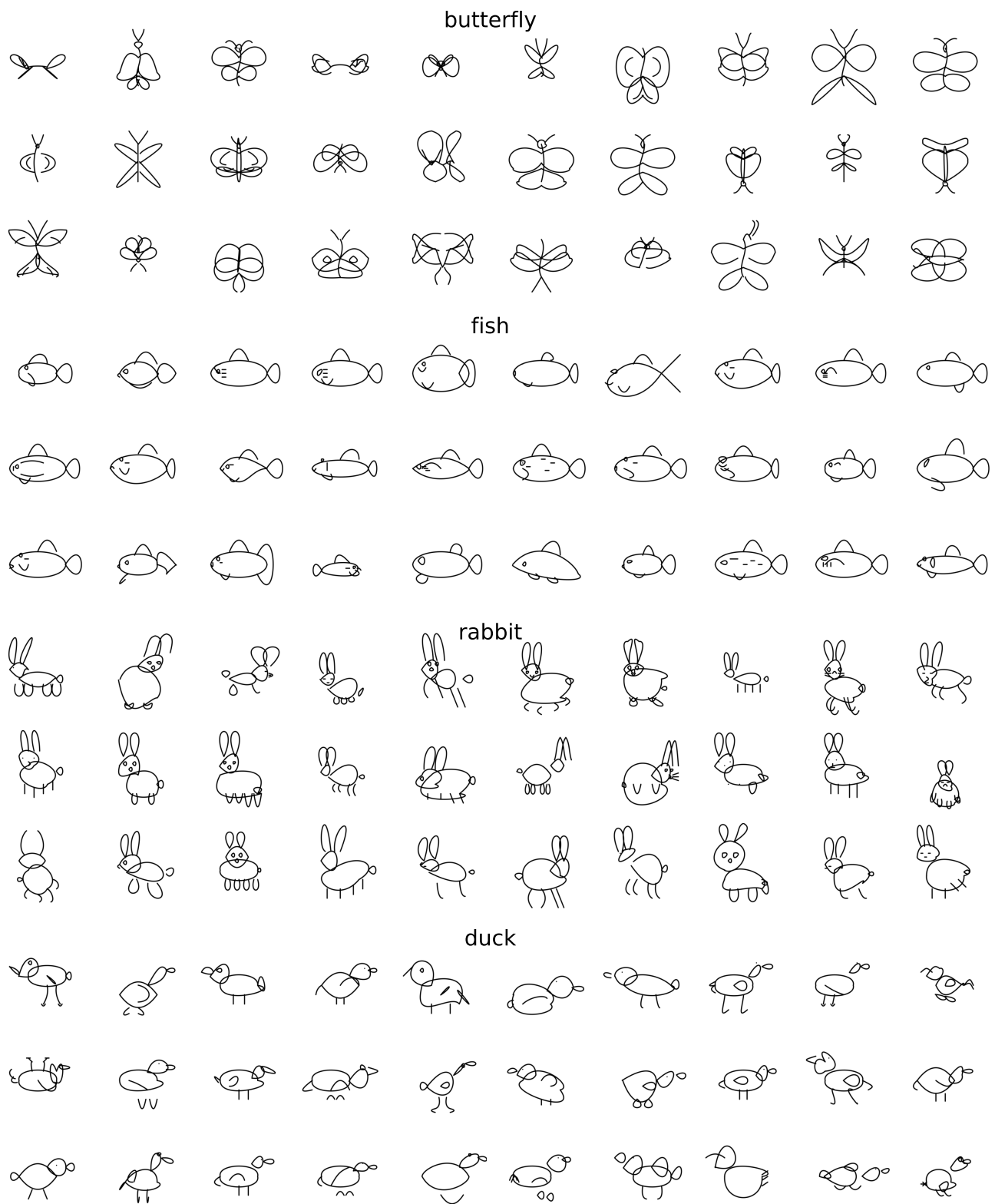


Figure 45. Sketches generated by SketchAgent for the eight categories of our human-agent collaborative study.



Figure 46. Sketches created collaboratively by users and SketchAgent as part of our collaborative human study.



Figure 47. Sketches created collaboratively by users and SketchAgent as part of our collaborative human study.

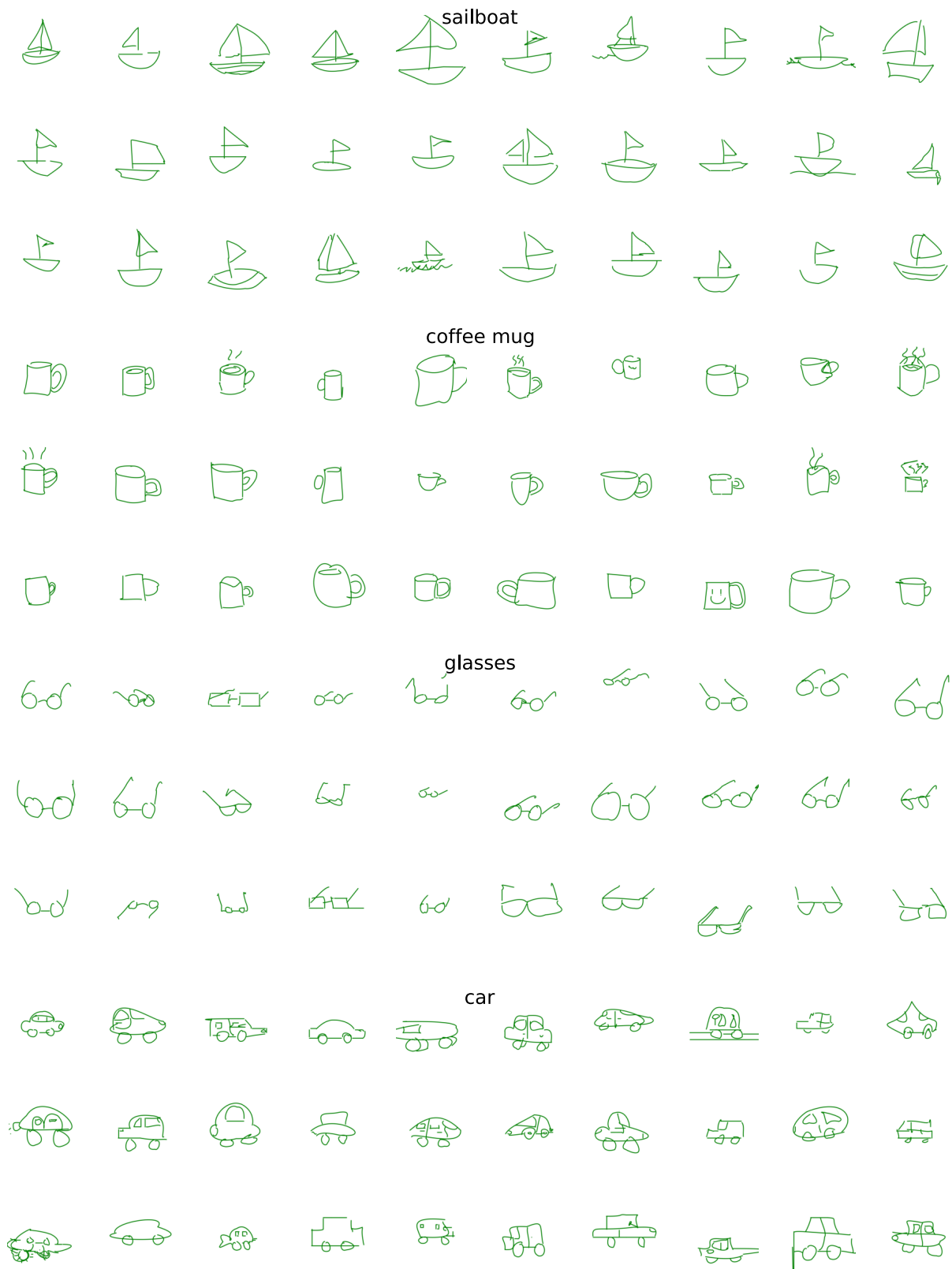


Figure 48. Sketches created by users in “solo” mode as part of our collaborative human study.

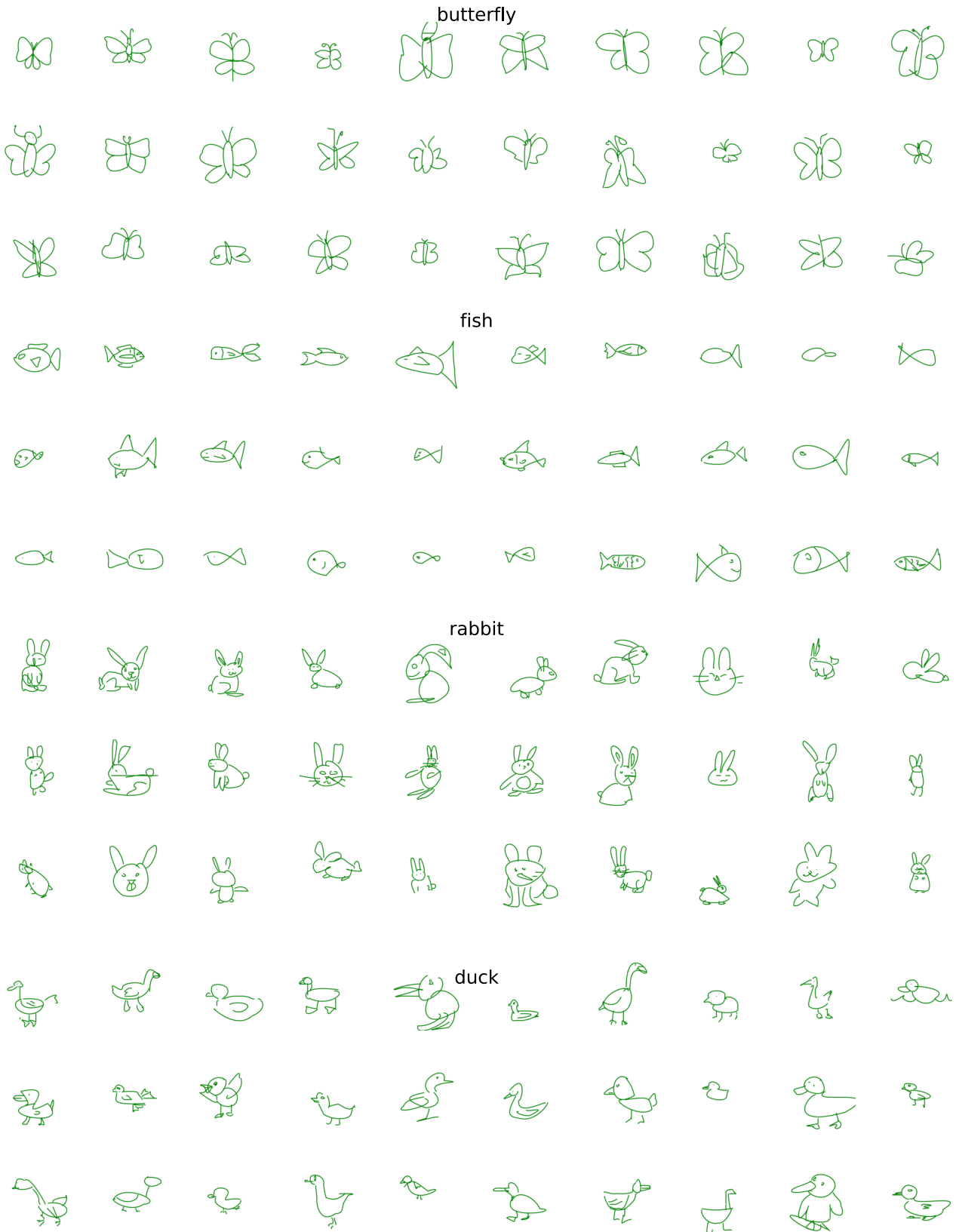


Figure 49. Sketches created by users in "solo" mode as part of our collaborative human study.