THE LANGUAGE OF SKETCHES

Xiaoyu Zhang

Carnegie Mellon University
Pittsburgh, PA 15213
April 2022

Thesis Committee
Oliver Kroemer, advisor, chair
Yonatan Bisk, advisor, co-chair
Jean Oh

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Abstract

Creative AI has seen much progress in recent years. Works like DALL-E 2 can generate inspiring art pieces from text descriptions. Instead of synthesizing realistic art works from language, we approach creativity from a different angle and investigate composition of semantic parts and visual concepts in sketches. For example, people can draw a circle to represent the moon, a scoop of ice-cream, or the face of a cat. In a similar way, language descriptors can be composed to create new concepts. People can draw a large round cat face or a narrow oval cat face.

In order to study this reuse of abstract concepts, we construct a dataset of language annotated sketches. We examined current sketch datasets and found that they either lack language annotations or semantic part annotations. Therefore, we collect a dataset of 11,150 (sketch part, text) pairs for 572 face sketches and 787 angel sketches.

To understand the limits of current vision-language models, we fine-tuned CLIP, a model pre-trained with a contrastive objective on 400 million (image,text) pairs and can map (image,text) pairs into a joint vision-language embedding space. We observed that (1) CLIP cannot easily generalize to an unseen category on the task of pairing sketches with their descriptions even though similar shapes and descriptions have occurred in training; (2) through fine-tuning, average cosine distance has increased between a pair of descriptors used by annotators to differentiate two sketches. With insights gained about how language and sketches interact in the CLIP embedding space, our aim is to facilitate research into models that can generate sketches in a part-based manner satisfying descriptions given by users of the pictures they have on their minds.

Acknowledgements

I first and foremost thank my advisors, Professor Oliver Kroemer and Professor Yonatan Bisk, for advising me on this project. I am very grateful that I had the opportunity to explore the creative sketching domain with them during my Master's. Through working with them on this project, I have learned how humans use language creatively to describe sketches, and I am excited about building systems that can interact seamlessly with humans in the future. I want to thank Professor Jean Oh for being on my committee.

I also want to thank my lab mates in the IAM and COMEDY lab (the name *COMEDY* was coined in April 2022 😆). Thank you guys for testing the numerous beta versions of my data collection interface and providing feedback without which the official deployment would not have been as successful.

I thank my parents for encouraging me to pursue research topics that I enjoy. Thank you guys for listening to all my crazy ideas.

Contents

Abstract ii

Acknowledgements iii

1 Introduction 1

2 Related Work 4

2.1 Sketch Dataset 4

2.2 Collaborative Drawing 6

3 Data Collection 11

3.1 Prompt-Guided Sketch Text Dataset 12

3.1.1 Overview 12

3.1.2 Interface Design 12

3.1.3 Deployment Results 16

3.2 Contrasting Sketch Text Dataset 17

3.2.1 Overview 17

3.2.2 Interface Design 18

3.3 Dataset Summary 19

4 Modeling **29**

4.1 Task Definition 29

4.2 Method 30

4.2.1 Contrastive Objective 30

4.2.2 Text Encoder 32

4.2.3 Vision Transformer 34

4.2.4 Fine-Tuning CLIP 35

5 Results & Analysis **37**

5.1 Classification Experiment 37

5.2 Word Similarity Experiment 39

6 Future Work **42**

A Designing the Requirement Section for the Prompt-Guided Sketch Text Dataset **44**

Chapter 1

Introduction

Creative AI, such as using deep learning models to generate paintings and music, has been a popular research domain. Since creative activities are hallmarks of human intelligence, numerous works have attempted to replicate the creative process on machines. For example, Pharmako-AI (Allado-McDowell & Okojie, 2020) is a book co-written by K Allado-McDowell and GPT-3 (Brown et al., 2020) through exchanges between the human and the language model. Works like DALL-E, GLIDE, and DALL-E 2 tackle the problem of synthesizing images from short language descriptions, and these generative models have produced many imaginative and inspiring art pieces (Ramesh et al., 2021; Nichol et al., 2021; Ramesh et al., 2022). These work are often motivated by the vision to create machines that can interact with people and augment human creativity. Our work is driven by a similar vision: how can we build systems that can be creative like humans so that AI agents and people can participate in creative activities together and inspire each otherN.

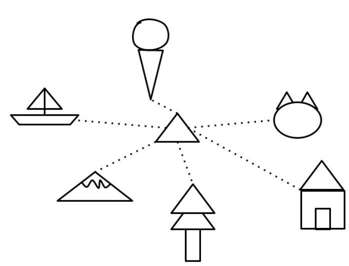


Figure 1.1: Triangle is adapted and composed with other shapes to create sketches of different objects. The objects are, from top and in clockwise order: *ice-cream, cat, house, pine tree, mountain*, and *boat*.

Instead of generating realistic art works from a wide variety of language like DALL-E, we approach creativity from a different angle and investigate composing basic visual concepts in object sketches. An example is illustrated in Figure 1.1: a triangle can be the sail of a boat, the cone of an ice-cream, the left/right ear of a cat, the roof of a house, the canopy of a tree, or the body of a mountain. The same triangle is adapted to become different parts in sketches of different objects. While sketches contain much fewer details compared to their natural images counterparts, people are not any less creative when sketching. One could say that the abstractness allows for more creativity.

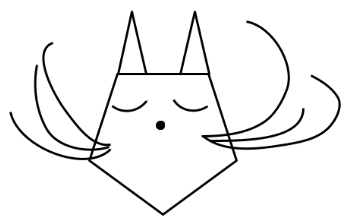


Figure 1.2: People are likely to provide a variety of descriptions when asked, *"What features does this cat sketch have?"*.

Similarly, language descriptors can be composed to form new concepts applicable to different objects. For example, combining *large* and *round* or *narrow* and *oval* to describe different kinds of cat faces. Many other objects can be *large round* or *narrow oval*: angel halo, glasses, mirror, plate, necklace charm, table, etc. Moreover, there are multiple ways to describe the same sketch, depending on the person's viewpoint. *What visual features does this cat sketch have?* (Figure 1.2) Some people might pick up on its usual size; others might notice its slightly edged chin, and there are still others that focus on the long whiskers curving upwards. People are likely to provide a variety of responses.

We want to build systems that can compose basic shapes in a creative manner and understand how words are combined and adapted to describe different objects. In this thesis, we are interested in semantic parts in sketches and how people describe them. For example, a face sketch may contain four semantic parts: eyes, nose, mouth, and face contour, that can be drawn using various geometric shapes with different features. After examining existing sketch datasets, we found that they do not contain either language annotations or semantic part annotations. We explain the details of related sketch datasets in Chapter 2. To fill the gap and study this reuse of abstract concepts, we construct a dataset of language annotated face and angel sketches. We elaborate on the data collection process in Chapter 3.

Moreover, we hope to understand the limits of current vision-language models, such as CLIP, or Contrastive Language-Image Pre-Training, which is trained on millions of image-text pairs. CLIP has been used extensively to provide a loss function for optimizing text-to-image synthesis models

(Frans et al., 2021; Ramesh et al., 2022; Gal et al., 2021; Patashnik et al., 2021). Despite the abundance of knowledge captured by CLIP's embedding space, we observed that CLIP, trained to associate natural images with their captions, has difficulty matching sketches from unseen categories with their part descriptions, suggesting that they might not understand that sketches from different categories are composed of parts different in semantics but share similar shapes and descriptions. We explain the details of CLIP and the fine-tuning process in Chapter 4, and we analyze CLIP's performance and changes in its text embeddings after fine-tuning on our dataset in Chapter 5.

Chapter 2

Related Work

change here

[X] not very satisfied with this sentence.

We survey two areas of related work: text-to-image synthesis and sketch representation learning. There is a long line of works on generating realistic images by learning their underlying distributions, especially on employing generative adversarial networks (GAN). Related to our work is a set of works on generating images from text descriptions or use texts to manipulate image styles, but they often work with a constraint set of language, with the exception of industrial-scale generative models with billions of parameters trained from text-image dataset on the hundred million scale. Most of these work focus on datasets of photo-realistic images, and our work instead centers around sketches, which are abstract since certain features of the objects in the sketches are illustrated figuratively and contain sparse information, as most part of the image is empty. Therefore, we also look into the field of sketch representation learning: what methods have been employed and what datasets are constructed.

2.1 Sketch Dataset

Since our work seeks to collect a dataset of (sketch parts, text description), we survey exisiting sketch datasets and find that they either lack language descriptions, for the entire sketch or for parts in a sketch, or they do not contain semantic object annotations.

TU-Berlin Eitz et al. (2012) is one of the first works to investigate the characteristics of free-hand sketches and attempt to extract local features based on orietation, which are later used in the task of sketch recognition. It also provides the TU-Berlin sketch dataset that contains $20,000$ sketches spanning 250 object categories with 80 samples in each. The TU-Berlin dataset is then extended for sketch-based 3D object retrieval to contain 1814 new sketches for 130 common household object categories. This additional set also contains hierarchical category labels; for example, the *animal* category contains *arthropod, biped, human, flying creature, quadruped, underwater creature* categories if going one level down the hierarchy. The TU-Berlin contains some of the highest quality sketches for a wide range of categories, but they lack both text descriptions and semantic part annotations, and our dataset contains both types of annotations, although only for simpler sketches in the face and angel category.

Datasets with Text Descriptions The QMUL dataset contains sketches of shoes and chairs: 419 shoe sketches of various types of shoes, such as high heels, boots, and ballerina flats, and 297 chair sketches of different kinds (Yu et al., 2016). Although QMUL annotates for attributes of the sketches, they come from a fixed set of descriptions and are obtained from product tags, so these descriptions do not reflect how humans would creatively describe the drawings on their mind, which is a feature of our collected dataset. The Sketchy dataset contains $75,000$ sketches of 125 categories, and each sketch is paired with an image; Sketchy is additionally annotated with sketch validity metrics and short descriptions from annotators (Sangkloy et al., 2016). However, these descriptions are more like comments given by the participants, and the collecting process is not carefully designed, unlike our data collection process, where obtaining the text description is the main focus, ensuring the quality of the language. Moreover, the short texts in Sketchy do not describe individual semantic object in the sketches.

Quick,Draw! Dataset Quick,Draw! collects the largest sketch datasets containing 50 million sketches distributed across 345 object categories, each containing around 100,000 sketch (Ha & Eck, 2017). Annotators for the TU-Berlin dataset have 30 minutes to draw one sketch, and annotators for the QMUL dataset have reference photos for the sketches they would draw later; on the other hand, Quick,Draw! participants had 20 seconds to create the sketches for a randomly assigned category without time to look for reference, so the Quick,Draw! sketches are the simplest; however, since sketches that failed to be recognized by a classification model are filtered, they are in general of good quality and are representative of the general population drawing skills. Moreover, many works on sketch representation learning use the Quick,Draw! dataset, so utilize these sketches allow us to potentially adapt some of the pre-trained generative models. These features of the Quick,Draw! dataset make it very favorable to be used to learn an collaborative drawing robot that can interact with a wide range of users, including children. However, Quick,Draw! dataset contains neither semantic part annotation nor language descriptions, and our collected dataset contains both in order to build collaborative drawing robots.

Datasets with Semantic Part Annotations The sketches in our dataset come from the Quick,Draw! dataset, but the original Quick,Draw! dataset does not provide labels for semantic objects in the sketches. The Sketch Perceptual Grouping (SPG) dataset (Li et al., 2018) and the SketchSeg dataset (Qi & Tan, 2019) contain annotations for semantic segmentation, meaning that each stroke in a sketch is paired with a semantic label; for example, strokes in a sketch for the *face* category are assigned one of the following labels: *eyes, nose, mouth, ear, hair, moustache, outline of face*. The SPG dataset annotates for $25,000$ Quick,Draw! sketches, belonging to 25 (out of 345) categories, and it selects 800 out of the original $100,000$ sketches in each category for annotation. The SketchSeg dataset builds upon the SPG dataset by using a recurrent neural work to generate additional sketches stemmed from one sketch in SPG; since each stroke in the generated sketch corresponds to a stroke in the original sketch, it automatically has part annotations. However, since the quality of the generated sketch is dependent upon the generative model, we choose the original SPG dataset, which is also publicly available. The SPG dataset does not contain text descriptions like our dataset, so although they can be used to build a collaborative drawing system, such system would not have the capacity to respond to natural language commands from users, and our work is interested in learning collaborative robots that can communicate freely with people.

2.2 Collaborative Drawing

Sketch-RNN Along with the Quick,Draw! dataset, Ha & Eck (2017) introduces Sketch-RNN and provides a web demo of collaborative drawing, where users can select a category and draw the first few strokes of a sketch, and the model will complete the rest of the sketch[1]. Sketch-RNN uses a variational autoencoder (VAE) with bi-directional RNN encoder and RNN decoder. Ha & Eck (2017) uses a vector format to represent the sketches: each sketch is a set of strokes, and each stroke is a sequence of points, so the smooth curve is approximated with piecewise linear splines created by connecting adjacent points in the sequence. Each point is a 5-component vector, $(\delta x, \delta y, p_1, p_2, p_3)$: the first two components represent changes in the (x, y) coordinate of the current point compared to the last one; the last three make up a one-hot vector representing current state of the sketch. Let $s = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}$, $s = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, if the current point will be connected with the next point;

[1]You can access the demo from this website `https://magenta.tensorflow.org/sketch-rnn-demo`

$s = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$, if the current point is the last point in the current stroke, so the pen is expected to be lifted next; $s = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$, if the current point is the last point in the sketch. The RNN encoder take this sequence as input and uses the final hidden state h of the RNN to parameterize a Gaussian distributions of size N_z, from which a latent random variable $z \in \mathbb{R}^{N_z}$ is sampled. The latent z will be used to (1) initialize the hidden state h_0 of the RNN decoder and (2) be concatenated with the vector at each time step to be fed as input into the decoder. The RNN decoder predicts parameters for distributions (Gaussian mixture model for sampling $\delta x, \delta y$ and categorical distribution for sampling p_1, p_2, p_2) at each time step from the hidden states of the RNN. Since the decoder works in an autoregressive manner, meaning that points in a sketch are passed one by one into the RNN, the decoder can *encode* the first few strokes provided by the users into the hidden vector, which is conditioned on for later steps to produce distributions of points that make up the rest of the sketch. Subsequent works on sketch representation learning can be grouped into those that represent the input in this stroke-point vector format and those that treat the entire sketch as a raster image.

Although our work uses the Quick,Draw! sketches, we render the vectors into raster images in order to use CLIP and its vision-language joint embeddings, since compared to Sketch-RNN, we need to incorporate text descriptions of the sketch parts. It is challenging to align semantic information in text with distributions of the x, y coordinates. While language gives high-level guidance on how strokes are organized on canvas to illustrate a particular concept, the coordinate information is too low level if directly used without an intermediate level of abstraction to aggregate these low-level sequence.

DoodlerGAN DoodlerGAN (Ge et al., 2020) is a more recent work on collaborative drawing and represents sketches as raster images; it uses Generative Adversarial Network (GAN) to model the sketches [2]. Compared to Sketch-RNN, which can only complete the missing parts of sketches once users are done with their parts, DoodlerGAN generates sketches using a part-based approach so that users and the system can take multiple turns to create sketches more collaboratively. Moreover, as indicated by Ge et al. (2020), compared to vector inputs, raster images can take better advantage of information from spatial structure of the parts and ignore effects of shifting coordinates by focusing more on relationships among the sketch parts. The part-based GAN has two networks: a part selector and a part generator. Given an incomplete sketch with some parts drawn (for example, a bird with its eyes and beak), the part selector determines what kind of part to draw next (for example, the bird wings). Then given the partial sketch and the category of the next part, the part generator produces a raster image of the part and its location on canvas. The part generator is a conditional GAN derived from StyleGAN2, and it takes as input an image with number-of-parts+1

[2] Try the DoodlerGAN demo here: `http://doodlergan.cloudcv.org/`

channels: one channel for each part and an additional channel for the whole partial sketch. The part selector uses the encoder of the part generator with an extra linear layer at the end as the classification head.

While the part-based generation aspect of DoodlerGAN aligns with our goal to create collaborative drawing agent, there is no language associated with each part, and the generation process is not guided by language supervision. Therefore, we

SketchBirds

Past works have learned sketch representations from a wide variety of tasks: sketch recognition, sketch generation from image, image generation from sketches, sketch retrieval of 3D objects, sketch retrieval of images, semantic segmentation of sketches, etc. The survey paper by P. Xu et al. (2020) provides a comprehensive overview and systematic taxonomy of . There is a wide range of tasks that can be done on sketches, both unimodal and multimodal, and, for each task, a large reservoir of deep learning methods used to solve the tasks. P. Xu et al. (2020) gives a comprehensive review of the task taxonomy, summarized the unique challenges associated with each individual tasks, and evaluated the different deep learning methods on sketch recognition through a library `TorchSketch` the authors wrote, and it contains implementation for CNN, RNN, GNN, and TCN. The sections that are most relelvant to us are: sketch generation, sketch segmentation. Sketch generation because we are trying to learn a generative model. Sketch segmentation because we are trying to gain insight about how are semantically meaningful units discovered in sketches and what relationships do the parts have with the whole sketch. Similar to images, sketches have hierarchical structure, and we

The hope is that we can leverage previous work on sketch representation learning to gain insights about sketches and how to learn good representations of them. What is unique about sketches compared to regular RGB images from, for example, ImageNet is that (1) sketches are abstract characterisation of the objects, and although humans can recognize and understand a sketch perfectly, they do not necessarily bear big resemblance to their image counterpart; therefore, methods that work well on RGB images, especially generative models like GAN that have successfully generated wide range of images from texts, a realm that we care about, it is not necessarily the case that they can generalize well to our dataset.

In terms of exploring the multimodal sketch generation realm (text-to-sketch synthesis), a recent work is SketchBird (Yuan et al., 2021). This work, similar to ours, deal with the unique challenge of generating sketches from textual descriptions. They setup the task to mimic or as a counterpart to the classic text to image generation on the CUB dataset (Wah et al., 2011). This work is also representative of a line of work that is based on GAN, unique in the way that it is outputting sketches, closer to the domain that we are interested in. The line of text-to-image synthesis work begins with conditional GAN (Reed et al., 2016), which also reports results on the CUB dataset. But what is slightly in lack for the dataset that SketchBirds collected But to examine the line of GAN work, we can see that AttnGAN (T. Xu et al., 2017) (what SketchBirds is based upon or) One thing we are especially interested in is how these models are able to extract the text features, and how they fuse text features with image features. Moreover what loss is used to encourage the alignment between the image and text domain. In SketchBirds, a bidirectional long short-term memory (Bi-LSTM) network is used as the text encoder. Inspired by AttnGAN, to extract text vectors that are visually aware, SketchBrids trains the text encoder with image-text pairs while minimizing the Deep Attentional Multimodal Similarity Model (DAMSM) loss, proposed in AttnGAN. This loss is calculated based on attention-driven text-image matching score, where matching is between two vectors, one is the vector representing a word in the sentence, and the other is a weighted sum of vectors of image regions, where the weight comes from a matrix of size $T \times 289$ (T being the number words in the sentence, and 289 being the number of image regions), calculated using dot-product similarity between word in the sentence and sub-region in the image. It seems like from quite a few papers, such as G. Xu et al. (2021), fuse the visual and textual space by combining the visual features using weights calculated by dot-similarity between the two modality, or vice versa to achieve cross attention. G. Xu et al. (2021) uses a LSTM+GloVE setup for the unimodal text embeddings.

The SketchCUB dataset collected by SketchBird contains sketches that are more similar to still-life portrait sketches and are very realistic, but sketches in the Quick,Draw! dataset are more similar to icons. This is due to how SketchCUB is transferred from RGB images in the CUB dataset by using open-source holistically-nested network (HED). The SketchCUB dataset contains 200 bird categories with 10,843 images. It includes a training set with 8,326 images in 150 categories and a test set with 2,517 images in the remaining 50 categories.

What are some other ways that we can extract visually informed text embeddings.

StyleGAN-NADA: CLIP Guided Domain Adaptation of Image Generators

Of course, there are other techniques to generate images from texts, namely, leverage large pretrained model such as GPT-3. GPT-3 and DALL-E are particular nowadays for researchers to replicate on their own and try to query the immense feature space for creative art pieces. However, the abstract art style work is not our focus, and while creativity is an interesting future direction, we emphasize the collaborative aspect more than creativity.

In the larger realm of RGB images: Therefore, our dataset will be a good benchmark for how well these models work at capturing the individual semantic components of an object. The reason that we claim this is that some work on GAN's have try to look at how to manipulate certain regions in the images by manipulating the latent space. While this line of work also try to look at how .This area of the work is around facial feature editting. Work such as Semantic Photo Manipulation with a Generative Image Prior (Bau et al., 2019), has an interactive interface where the user can use stroke to indicate where in the image they would want a certain object, and the GAN will generate the objects in that location. "semantic image editing tasks, including synthesizing new objects consistent with background, removing unwanted objects, and changing the appearance of an object". semantic edit on an object. They would apply a semantic vector space operation in the latent space. How our work is different from this work is that: how well the methods can work on sketches and how well can the edits can done through language. [?] Moreover, it seems like we need to have an image already in order to do the manipulation, but for our ideal tasks, we start from a blank canvas.

Chapter 3

Data Collection

To study the composition of visual concepts, we need to collect a sketch dataset that contain both semantic part and language annotations. We tested two ways of collecting such dataset. Initially, in order to create a dataset with creative composition of shapes, such as the use of triangle in different sketches illustrated in Figure 1.1, we designed text prompts like *happy angel* and asked annotators to draw sketches in response to the prompts while annotating each step in their drawings. After deployment of pilot, although we collected many creative sketches, we saw major problems related to misalignment between the drawn parts and the text descriptions. We explain this prompt-guided design and its associated problems in details of in Section 3.1.

After re-design, we leverage existing sketch datasets, QuickDraw and SPG (both explained in details in Section 2.1), to solve the problems of low-quality part and text annotations unusable for model learning. In the new version, we present two sketches at a time to annotators, where the sketches contain semantic parts that are visually contrasting to aid annotators in coming up with creative descriptions. We walk through the process of designing this *contrasting* design in Section 3.2. After receiving satisfactory pilot results, we deployed the design full scale on Amazon Mechanical Turk (AMT); we present statistics of the dataset and explain how it allows us to study creative composition of abstract concepts in Section 3.3.

3.1 Prompt-Guided Sketch Text Dataset

3.1.1 Overview

When we first started designing our data collection interface, we wanted to collect sketches for prompts similar to the ones used in DALL-E (Ramesh et al., 2021): creative composition of attributes and objects that are not commonly associated. For example, *an evil cup of bubble tea* and *happy moon*. In addition to sketches, we also require annotators to decompose their drawings into steps and provide descriptions for each step. We hope to discover combination of simple shapes like examples in Figure 1.1, and because the imaginative prompts would result in creative sketches, the part descriptions would also be interesting, like.

We deploy the data collection interface on Amazon Mechanical Turk (AMT), which is a crowdsourcing website that hosts different machine learning annotation tasks. In the remaining text, we use the word *turker* to refer to annotators we recruit on AMT; we will also use the word *HIT*, Human Intelligence Task, to refer to a task hosted on AMT. Please refer to AMT FAQs for official definition and answers to questions related to AMT.

For each HIT, we need to design: (1) instruction and requirements explaining the dos and don'ts; (2) qualification task to train turkers to provide high-quality annotations; (3) the main interface for data collection. After deploying the pilot, we realized a few major problems with this design. Firstly, due to the subjective nature of sketching, although the sketches are very creative, it was hard to understand how some annotators are illustrating the given prompts. Moreover, turkers are taking more than 30 minutes for each task, and, most importantly, many descriptions do not align with the drawn objects, making the data difficult to use for model learning. For example, in one step, an annotator drew the entire cat face, but they only annotated *big eyes*.

3.1.2 Interface Design

Main Task Interface

We illustrate a typical annotation process in Figure 3.1. The annotator draws a step on the canvas, enter text description for this step in the *Annotation* column, and hit *Add* to display it as a new row in the annotation table. For the annotator's convenience, we include an *Undo* button and a *Clear* button for erasing strokes and clearing the entire canvas. If the annotator wants to remove

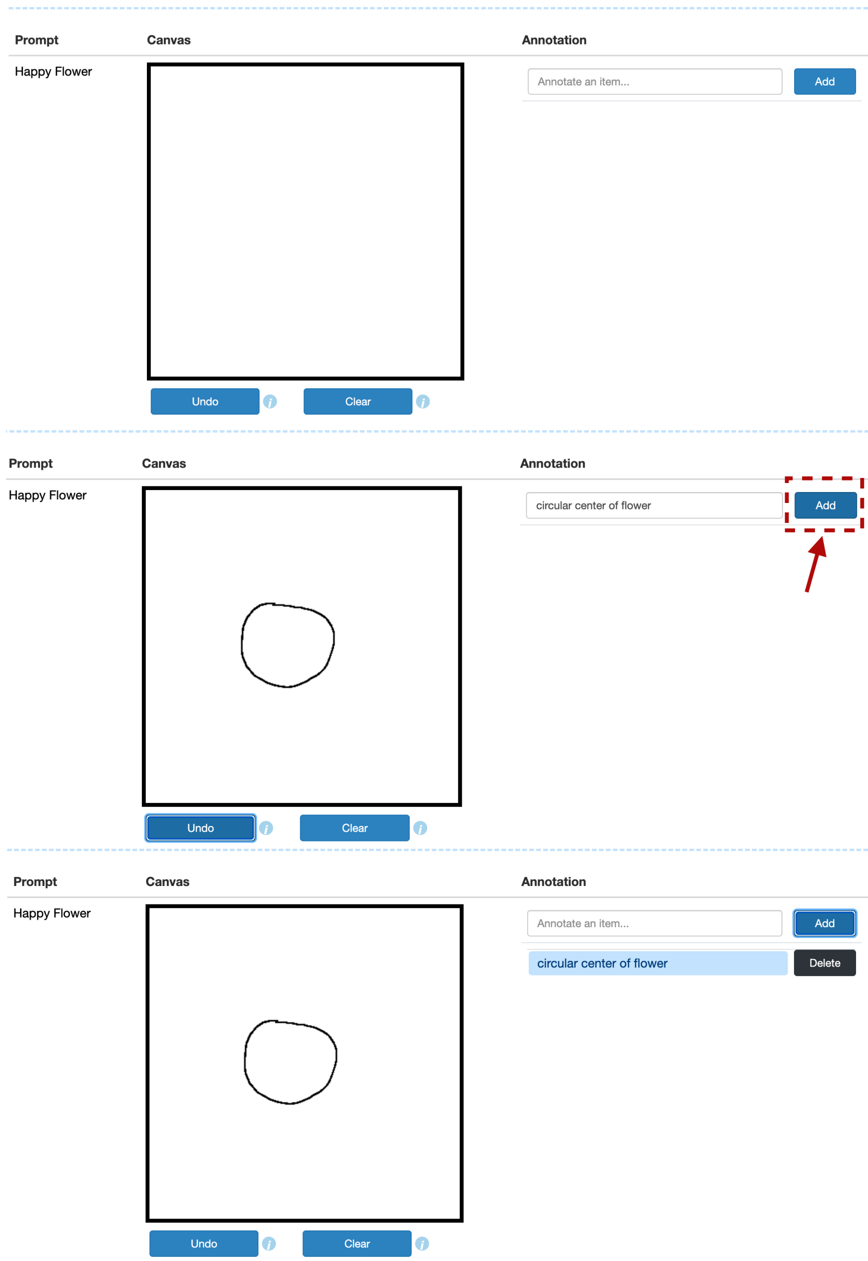


Figure 3.1: A typical annotation process. Top: interface at the start of annotation. Middle: before adding text descriptions for the drawing; red arrow and box show where to click to add text. Bottom: after adding text descriptions for the flower sketch.

an entire step, including the drawing and the text description, they can use the *Delete* button. An example is shown in Figure 3.2. Repeat the drawing-and-adding process until the drawing is done. This design encourages turkers to decompose their drawings into semantically meaningful parts.

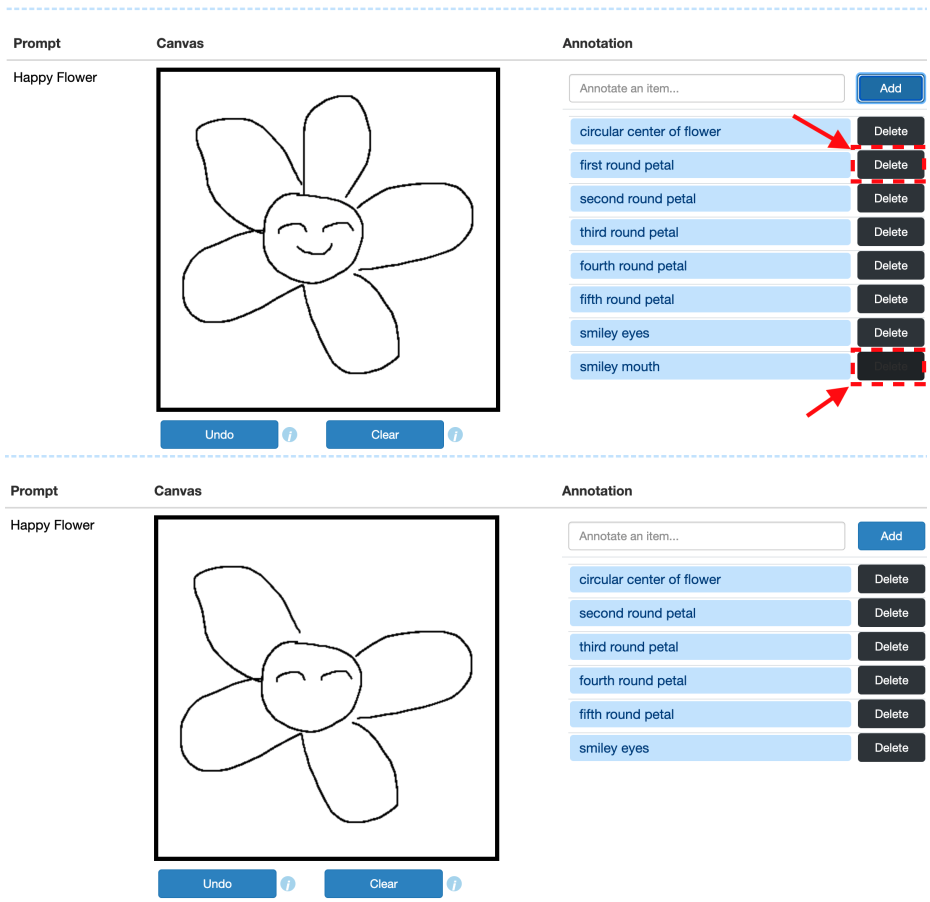
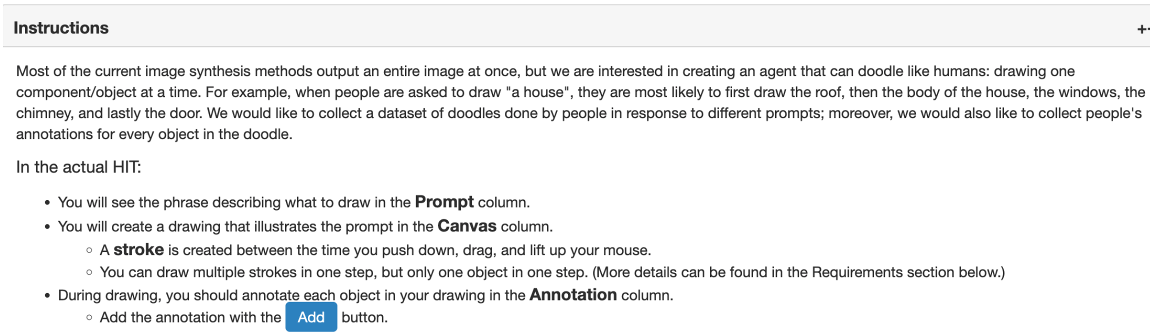


Figure 3.2: Top: the completed annotation for the prompt *Happy Flower*. Red arrows and boxes point to *Delete* buttons that can delete the text annotations along with their drawings. Bottom: after deleting the steps *first round petal* and *smiley mouth*.

We encountered some difficulties when implementing the *Delete* button. At the beginning, we treated erasing strokes as drawing the same strokes but in white; however, when strokes overlap each other, overwriting with white strokes breaks other strokes into segments. Therefore, we change the drawing canvas to use layers like Photoshop, so that deleting strokes would be the same as deleting an entire layer, leaving other strokes intact.

Instruction and Requirement

To ensure that turkers understand the purpose of collecting the dataset, the instruction begins with the motivation behind this project (Figure 3.3). What we struggled the most when drafting


Instructions +-

Most of the current image synthesis methods output an entire image at once, but we are interested in creating an agent that can doodle like humans: drawing one component/object at a time. For example, when people are asked to draw "a house", they are most likely to first draw the roof, then the body of the house, the windows, the chimney, and lastly the door. We would like to collect a dataset of doodles done by people in response to different prompts; moreover, we would also like to collect people's annotations for every object in the doodle.

In the actual HIT:

- You will see the phrase describing what to draw in the **Prompt** column.
- You will create a drawing that illustrates the prompt in the **Canvas** column.
  - A **stroke** is created between the time you push down, drag, and lift up your mouse.
  - You can draw multiple strokes in one step, but only one object in one step. (More details can be found in the Requirements section below.)
- During drawing, you should annotate each object in your drawing in the **Annotation** column.
  - Add the annotation with the Add button.

Figure 3.3: The instruction section used in the prompt-guided sketch text dataset.

the requirements was deciding what a single *step* in sketching was; how do we clearly explain this definition to the turkers? We considered providing a list of geometric shapes, such as rectangle, triangle, circle, etc., or asking annotations for each stroke, but these options could not reflect how people naturally sketch.

There is a wide spectrum of allowed annotations depending on how people sketch. For example, when drawing for the prompt *Happy Face*, one person might annotate 3 steps: *large u-shaped face*, *round eyes*, *big smiley mouth*. But for someone who likes to draw detailed eyes, they might describe the shape of the eye contour and the length of the eyelashes. The great variation in personal styles makes creative sketches fascinating to study but also challenging to collect a high-quality dataset.

We resorted to repeatedly testing the interface with lab mates to refine the requirements. The refinement process is explained in Appendix A. The requirements deployed in the final pilot is shown in Figure 3.4. Bad Example 2 is shown in Figure 3.5 as an instance of the examples used in the final requirements. To view all the examples, refer to: `https://erinzhang1998.github.io/portfolio/amazon_anno`.

Qualification

We set up a qualification test on AMT to (1) train turkers to have better understanding of the task and (2) to select turkers who can provide annotations that satisfy all the requirements. Similar to the process of writing the requirements, we went through several rounds of testing with students in the lab to come up with a set of questions correspond well with the requirements. The qualification test starts with the same instruction and requirements in the final HIT, thus allowing turkers to familiarize themselves with the requirements and ask clarification questions before completing the actual annotation task. The test leads with a navigation bar (Figure 3.6) to make it convenient for

Requirements +-

The entire drawing must satisfy:

1. Each drawing should contain at least 2 steps.
Do not complete the drawing in one step.
See details in: Bad Example 1

Every step must satisfy:

1. Do not draw multiple objects in one step, *unless* they are multiple copies of the same object.
For example, do not draw roof, window, and door of the house all in one step. *But*, you can draw two windows in one step.
See details in: Bad Example 1 Good Example 2 (Step 3)

2. Name the object in the annotation.
For example, if you draw a roof, annotation should at minimum include the word "roof".
See details in: Bad Example 2 (Step 1) Bad Example 2 (Step 2)

3. If you use the word "right", it should refer to this side: →. If you use the word "left", it should refer to this side: ←.
For example, do not draw a right window of the house and then annotate it with "left window".
See details in: Bad Example 2 (Step 3) Bad Example 2 (Step 4)

4. Differentiate between plural and singular.
For example, if you draw two windows, annotation should include "windows" instead of "window".
See details in: Bad Example 3 (Step 3) Good Example 2 (Step 3)

Note:

1. *Not* required to describe the geometric shape of the object, but preferred.
For example, if you draw a triangle as the roof of a house, you do not need to annotate "triangular roof"; annotating with "roof" is okay, since this satisfies "Name the object in the annotation".
We do not require describing the geometric shape, because in some cases it is difficult to do so, but you should try describing it when you can.

Ideally, when writing the annotation for each step in the drawing, imagine that you are teaching a kid how to draw. How would you describe to them about the object you are drawing?

Figure 3.4: Final version of the requirements. The Bad Example links to counter-examples of the requirements, and Good Example links to good examples. When turkers click on the links, they are directed to the examples illustrating the corresponding requirement. This design helps them to understand the requirements better and provides high-quality annotations

turkers to switch between questions; originally, we displayed all questions on one page, but some found it time-consuming to scroll from the later questions back up to the instructions, so we decided to display one question at a time. We show one question from the final qualification in Figure 3.7. Each question is a mock-up of the main task interface, and turkers need to determine whether every step satisfies all the requirements. We also include hints on which requirement the question is testing to encourage turkers to revisit the requirements and form better understanding of the task. To see the full test, refer to: `https://erinzhang1998.github.io/portfolio/amazon_qual`.

3.1.3 Deployment Results

In our first pilot, we used prompts in the forms of *adjective*$\times$*noun*. The list of adjectives includes: *happy, sad, surprised, sleepy, love-struck, evil*; the list of nouns includes: *person, kid, cat, bear, dog, sheep, jellyfish, cup of bubble tea, apple, burger, sun, moon, star*. We want to see what sketches and text descriptions annotators would provide for prompts that ask for imaginative beings not in this world and include novel compositions of unrelated concepts, such as *evil apple* or *love-struck moon*.

With these creative prompts, we hope to collect data that contain interesting compositions of the same geometric shapes and descriptions across different objects. We can then learn models that can, for example, generate circles to be different parts in different objects: eyes, moon, cherries, and angel halo.

Since we only collected 55 sketches, we were able to manually examine every sketch, and we found many creative sketches [FIGURE]. However, one issue was that turkers took a long time, on average 30 minutes, to complete one sketch and provide descriptions.

The second problem was that it was difficult to understand how some annotators interpreted the prompts through their sketches. [FIGURE] Indeed, sketching is by its nature very subjective, a common challenge in creative AI.

The third problem, the most concerning one, was that the part descriptions did not align well with the sketches: some annotators failed to describe every part they drew in a step, or they described parts not in the annotated step. [FIGURE]

3.2 Contrasting Sketch Text Dataset

3.2.1 Overview

In response to the pilot results, we reconsider the data collection pipeline. We examined existing sketch datasets to see how their annotations could facilitate our data collection (dataset details are explained in Section 2). To shorten the time spent on sketching, we no longer asked annotators to sketch and instead only asked them to describe sketches in the QuickDraw dataset. Although we were no longer able to design text prompts ourselves and collect creative sketches like the ones in the previous pilot, we solved the problem that it was difficult to understand how some sketches illustrate the give prompts.

The most important objective that this dataset should fulfill is allowing us to study how sketches share similar semantic parts and descriptions that are adapted to be used for different objects. Therefore, we must resolve the issue that some text descriptions we collected in the pilot either did not describe everything drawn in a step or described more than what was drawn. Therefore, we used semantic part annotations from the Sketch Perceptual Grouping (SPG) dataset, which provides semantic part labels for each stroke in $20,000$ sketches from the QuickDraw dataset. In this way, annotators did not need to spend time thinking about how to segment sketches into parts themselves.

Moreover, to help annotators coming up with creative ways to describe the semantic parts, in each task, we presented a pair of sketches with contrasting features, implicitly priming them to describe the visual differences.

3.2.2 Interface Design

Main Task Interface

When collecting the previous prompt-guided dataset, we relied on testing the interface with students in the lab to determine our design, but we observed performance differences between the students and turkers, such as amount of variety in the sketches, amount of time spent on the task, and common confusions related to understanding the requirements. Therefore, when collecting the contrasting sketch text dataset, we deployed several pilots on AMT to design the new interface. In Figure 3.8, we show how the main task interface progressed from the first pilot to the final version used to collect the entire dataset.

To better study how similar words are used differently across sketches, we changed to collect adjective phrases (Figure 3.8b, 3.8c) from collecting whole sentences (Figure 3.8a). We juxtaposed two sketches and highlighted the parts to be annotated in different colors to help annotators notice the contrasting features. Moreover, this design expedited the annotation process, since it was easier for people to perform contrasting tasks than to generate descriptions from a single sketch.

Instruction and Requirement

At the beginning, the instruction limited the annotators to provide three types of descriptions: shape, size, and position. However, in order to collect creative descriptions, we lifted restrictions on the type of words and only required annotators to fill in the blank with adjective phrases. We also provided some examples of adjective phrases in common sentences, unrelated to our task, for annotators to better understand their usage (Figure 3.9).

Since we simplified the HIT from 3 sub-tasks, sketching for the prompt, segmenting the sketches, and describing each step, to only asking for part descriptions, the requirements are much easier to write. We received less feedback from the annotators on being confused about the kind of sketches we wanted and the definition of semantic parts, which are now automatically highlighted in the sketches.

We relied on the examples in the instruction to give annotators an idea of what descriptions we wanted. Some examples that we used in the tasks are shown in Figure 3.11. However, the downside was that, primed by the examples, annotators described the parts using words in the examples instead of coming up with a variety of descriptors, and we observed this behavior in the pilots. Therefore, we emphasized an additional requirement that asked the annotators to not limit themselves to words in the examples, and they should use any words that could illustrate the parts well.

Since in the future we want to use our dataset to learn models that can generate sketch parts from text descriptions, the text should pertain to visual properties of the parts, so we required that *Do not use adjectives that fail to describe specific visual properties of the objects in the sketches.* A caveat was that some annotators might consider descriptions about emotions expressed in the sketches unrelated to visual properties, since they are abstract compared to words like *rectangular* and *large*. We wanted to collect annotations for face sketches, and parts like *eyes* and *mouth* can be *smiley* or *sad*, so we added that adjectives describing emotions were allowed.

The requirements used in the final version is shown in Figure 3.10.

Qualification

We prepared 10 qualification questions, all yes/no questions. We used the qualification test to train turkers to understand the requirements better. Each question had a hint that stated which requirement and examples were helpful for solving the questions. The purpose of the qualification test was not to trick annotators but to ensure speed and quality of the annotation. We show one question from the qualification in Figure 3.12. To see the full test, refer to: `https://erinzhang1998.github.io/portfolio/v2qual`. At the end, there are 88 annotators who worked on our tasks.

3.3 Dataset Summary

Our dataset contains sketches, their semantic part annotations, and descriptions for every part in a sketch. The sketches comes from the QuickDraw dataset (Ha & Eck, 2017), and the semantic part annotations come from the SPG dataset (Li et al., 2018); both datasets are explained in details in Section 2.

We annotated for 2 categories of sketches: face and angel. For angel sketches, we annotate for the parts *halo*, *eyes*, *nose*, *mouth*, *body*, *outline of face*, and *wings*. For face sketches, we annotate for the parts *eyes*, *nose*, *mouth*, *hair*, *outline of face*.

	Face	Angel
Number of contrasting pairs	2515	3060
Number of distinct words	833	1107
Number of sketches	572	787

Table 3.1: Dataset statistics by category.

	Face					Angel						
	eyes	nose	mouth	hair	face	halo	eyes	nose	mouth	face	body	wings
Number of sketches	334	572	572	104	572	558	114	8	80	732	781	779
Number of distinct words	228	360	325	152	314	365	112	21	88	379	425	534
Number of contrasting pairs	689	401	687	126	612	559	114	8	80	733	785	781

Table 3.2: Dataset statistics by sketch parts. The phrase *contrasting pair* refers to a pair of sketches with contrasting features that are presented to the annotators for descriptions of their parts.

In Table 3.2, we present dataset statistics broken down by semantic parts, and, in Table 3.1, we show the same statistics by sketch category. In Figure 3.13, we show 100 most frequently used words in our dataset.

The *Creative Birds* and *Creative Creatures* datasets collected by DoodlerGAN (Ge et al., 2020) contain 2 categories, like ours, but there are 9k sketches in each category, and ours contains one tenth as many. Although we fall short on the number of sketches and the variety of sketching styles, we approach creativity from a completely different angle: we focus on how people compose similar basic shapes to create sketches of different categories and adapt similar language to describe different sketch parts, as explained in Section 1 and Figure 1.1.

Our face sketches contain 5 different parts, and angel sketches has 7 parts; there are 7 parts in Creative Birds and 16 parts in Creative Creatures. Again, we do not annotate for as many parts as DoodlerGAN if compare directly with them. However, in our dataset, every part in every sketch has at least 2 different language annotations describing the visual features, while DoodlerGAN has no language annotation. With text descriptions, in addition to part-based generative model, we can study how people compose concepts (e.g. *large+round*) and apply the same concepts to different semantic parts (e.g. *round eyes* and *round halo*).

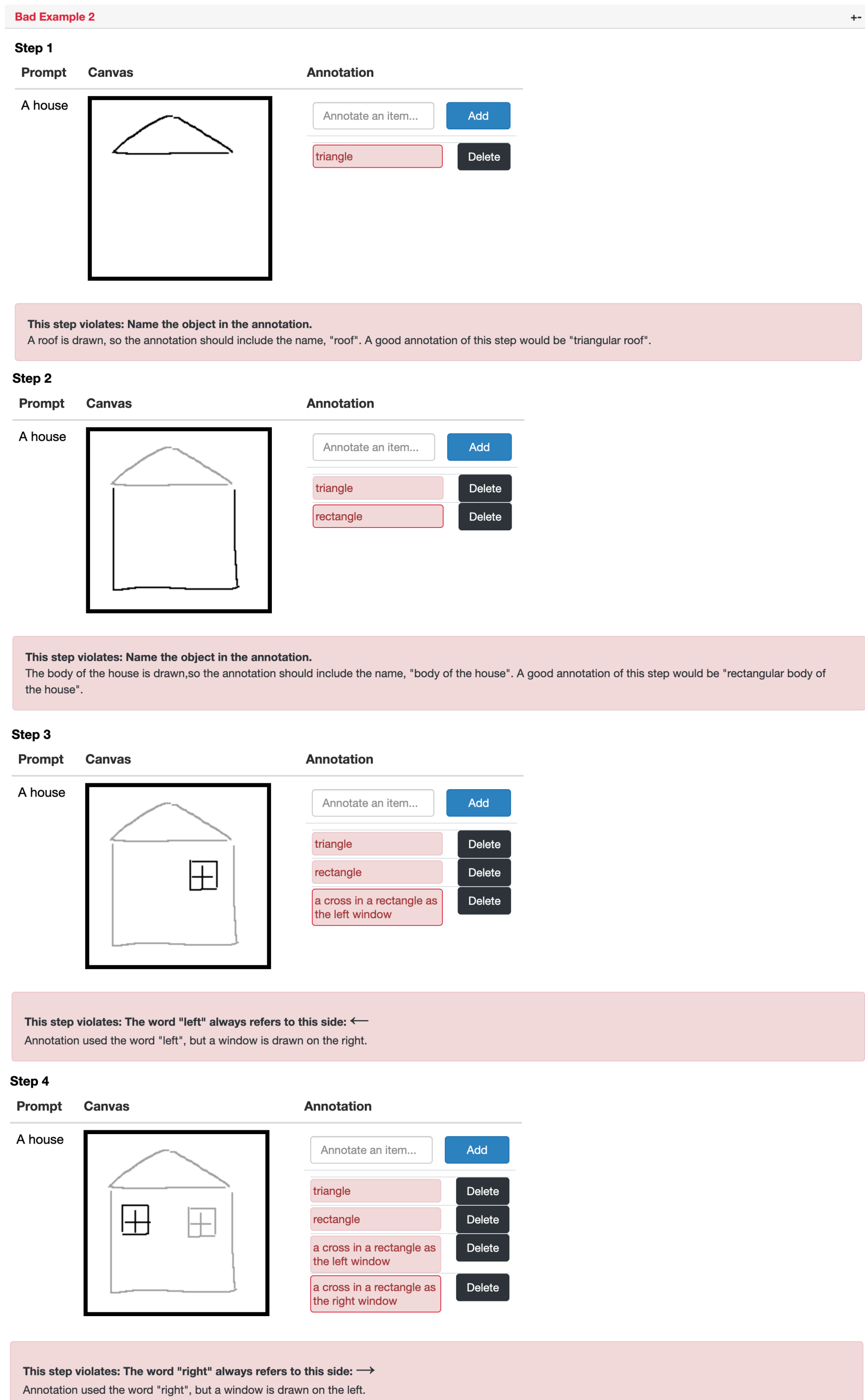


Figure 3.5: An example used to explain the requirements to turkers.

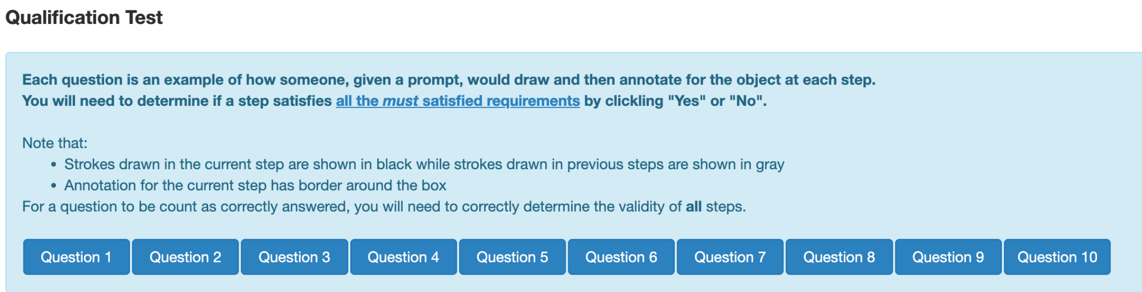


Figure 3.6: Navigation bar in the qualification test.

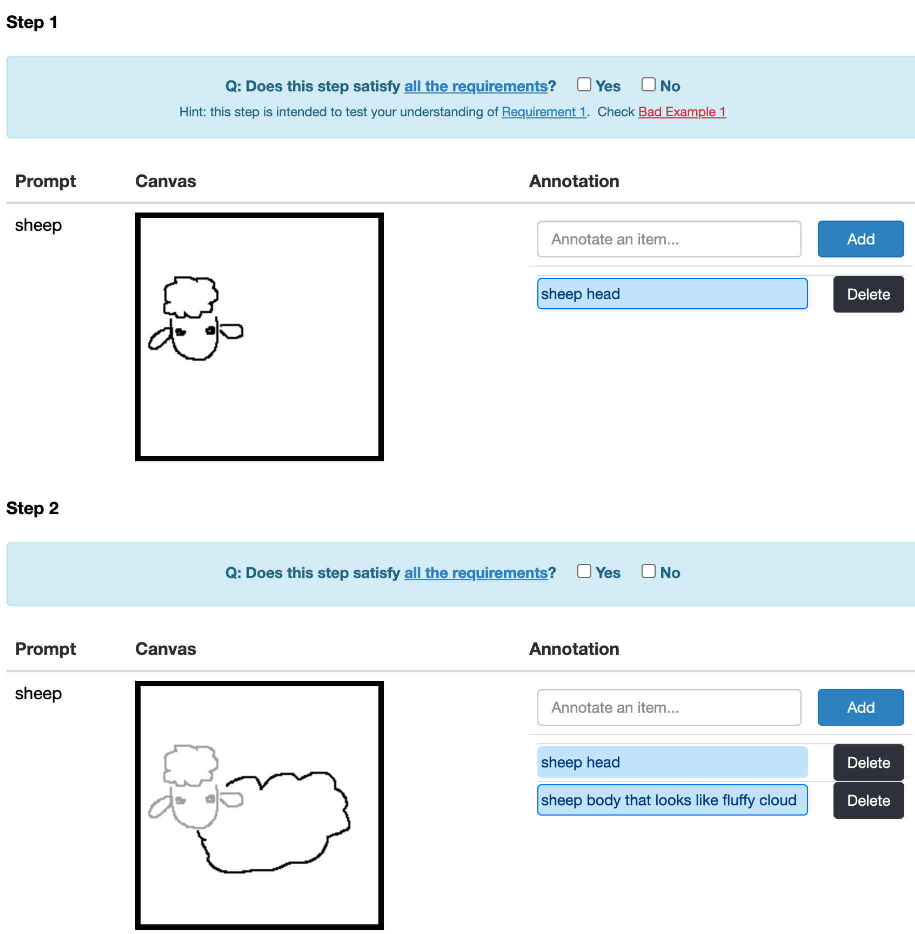
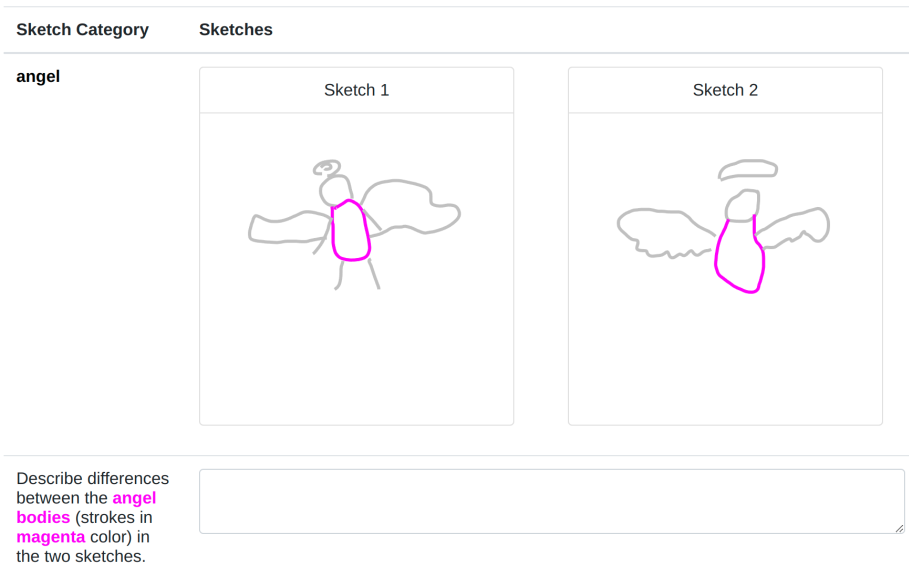
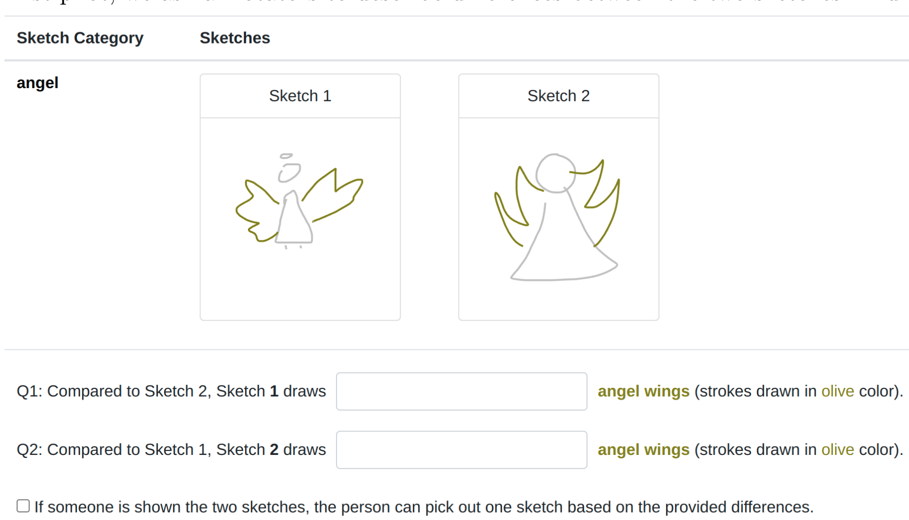


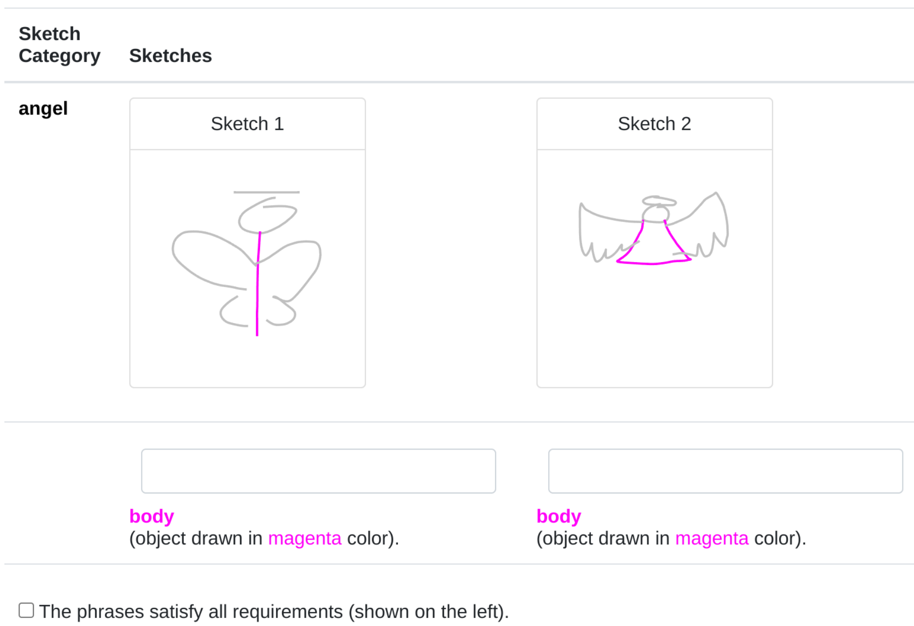
Figure 3.7: An example of the questions in the qualification test.



(a) In the first pilot, we ask annotators to describe differences between the two sketches in full sentence.



(b) Compared to the previous pilot (3.8a), we still ask explicitly the differences between the sketches but limit annotations to adjective phrases.



(c) Similar to the last pilot (3.8b), the final version asks for adjective phrases, but we do not state explicitly that annotators should describe the visual differences to allow for more creativity.

Figure 3.8: Different versions of the main task interface in chronological order. Final version used to collect the entire dataset is 3.8c.

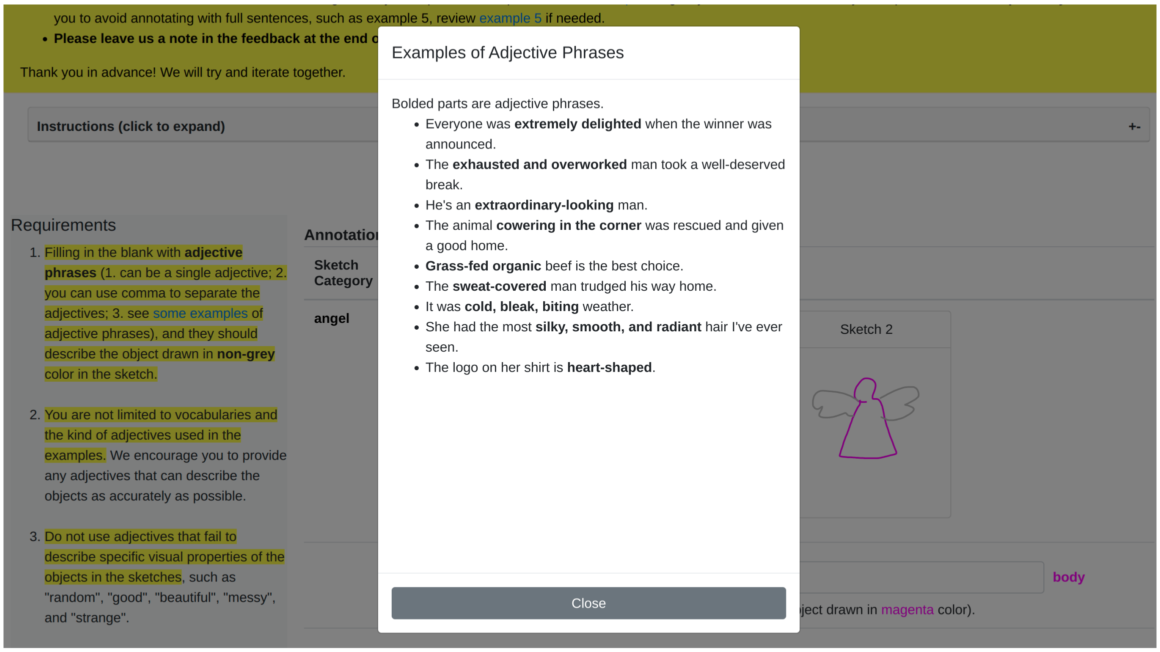
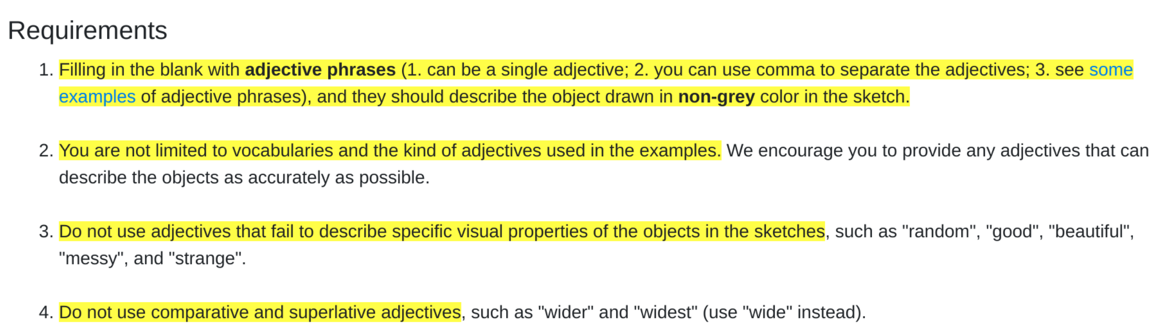


Figure 3.9: A pop-up window containing examples of adjective phrases, used in collecting contrasting sketch text dataset in Section 3.2. It can be opened from the side panel by clicking on *some examples*. Note that we used examples that are unrelated to our task on purpose, because annotators tended to repeat words in examples, so we tried to not bias them.



Requirements

1. Filling in the blank with **adjective phrases** (1. can be a single adjective; 2. you can use comma to separate the adjectives; 3. see some examples of adjective phrases), and they should describe the object drawn in **non-grey** color in the sketch.
2. You are not limited to vocabularies and the kind of adjectives used in the examples. We encourage you to provide any adjectives that can describe the objects as accurately as possible.
3. Do not use adjectives that fail to describe specific visual properties of the objects in the sketches, such as "random", "good", "beautiful", "messy", and "strange".
4. Do not use comparative and superlative adjectives, such as "wider" and "widest" (use "wide" instead).

Figure 3.10: Requirements used in collecting the contrasting sketch text dataset (Section 3.2).

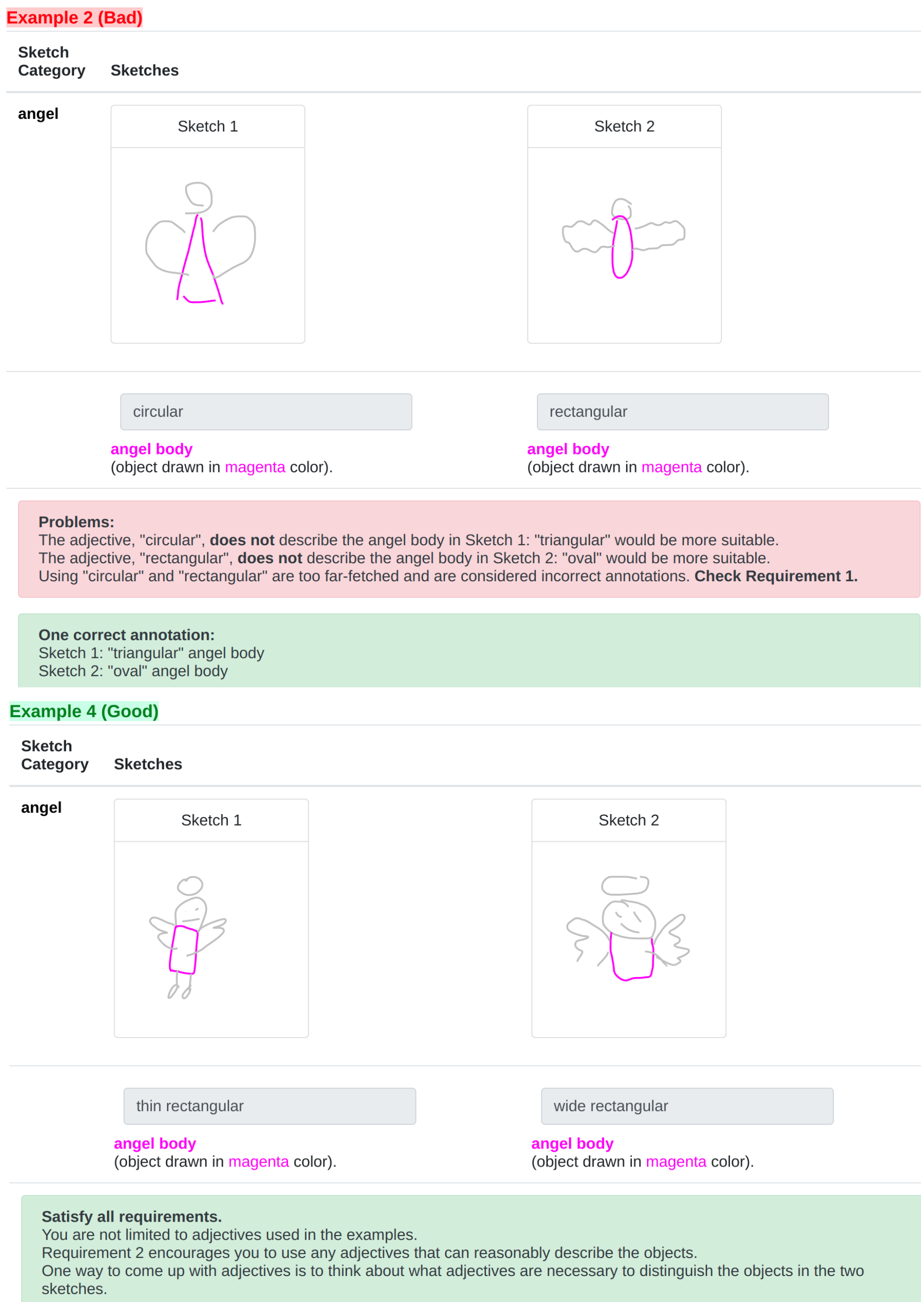


Figure 3.11: 2 examples used in the instructions to collect the contrasting sketch text dataset (Section 3.2).

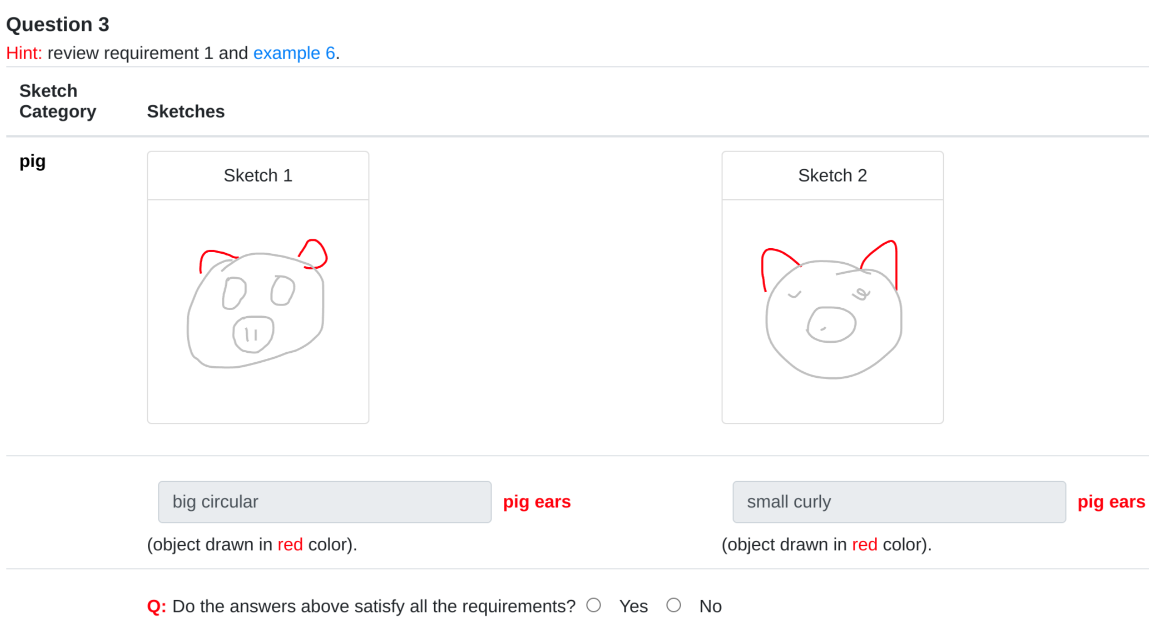


Figure 3.12: Question 3 from the qualification test used to collect the contrasting sketch text dataset (Section 3.2). We show a hint at the beginning of each question telling the annotators which requirement this question is testing. In this way, we encourage them to review the requirements so that they have a good understanding of the task and can provide high-quality annotations in the real HIT. The question interface is the same as main task interface that annotators will see when they annotate. The 1-to-1 mock-up helps them to be familiar with the workflow.

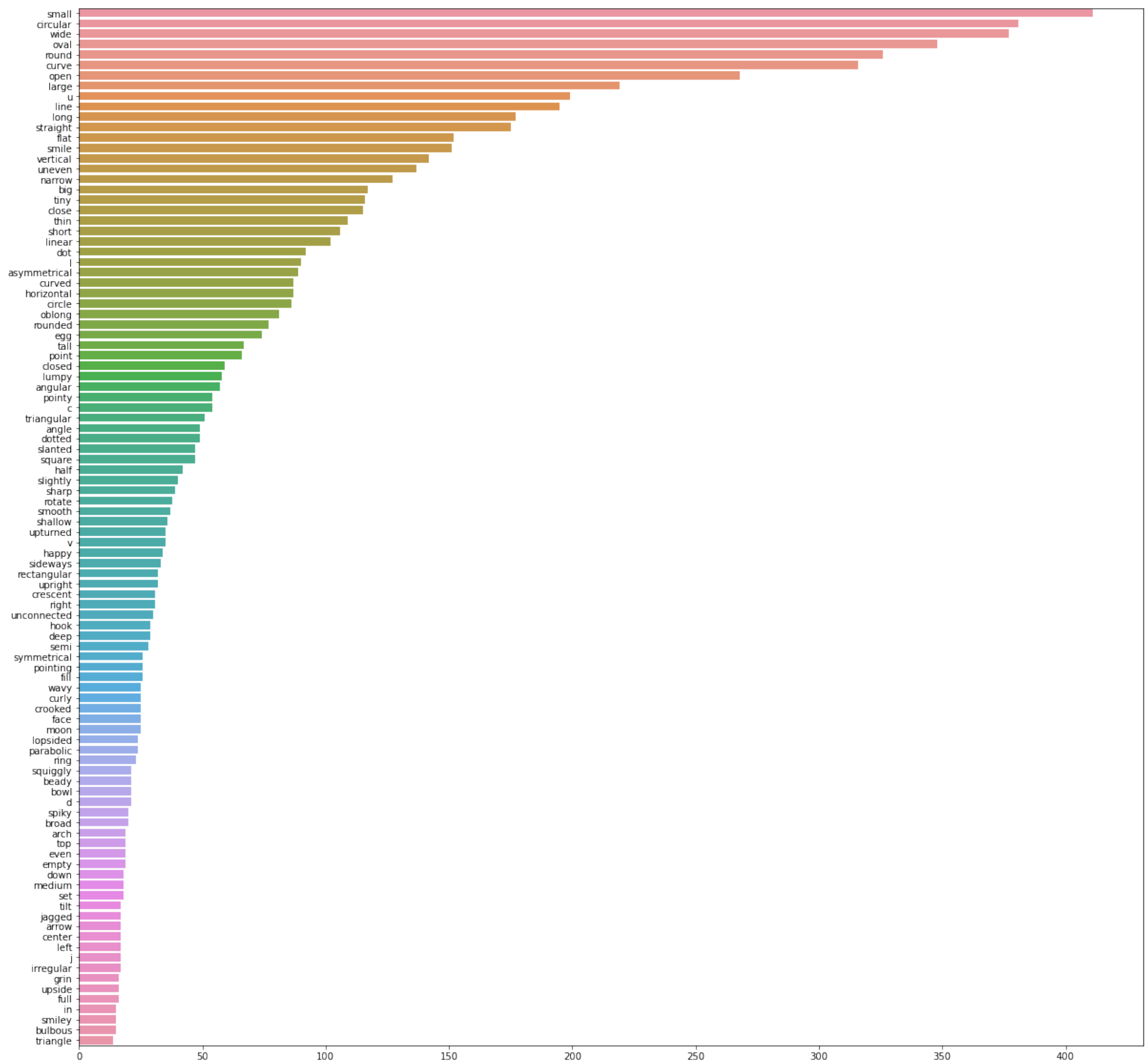


Figure 3.13: Top 100 most frequent words in the dataset corpus.

Chapter 4

Modeling

4.1 Task Definition

(a) t_1: *wide triangular body* (b) t_2: *small rectangular body*

Figure 4.1: Two angel sketches, s_1 on the left and s_2 on the right, and their part annotations, t_1 and t_2. The task is for CLIP to determine which sketch matches a given t_j.

Given two sketches (s_1, s_2) and their part annotations (t_1, t_2), such as the pair shown in Figure 4.1, the task is to determine which sketch t_j should be paired with. We use this task to evaluate the joint vision-language embedding space of CLIP, which is often used as part of the objective function for models that generate image from text; the objective function often involves maximizing the cosine similarity between the generated image and the provided text (or some variants engineered to work for the particular latent space of the generator used) (Frans et al., 2021; Patashnik et al., 2021; Gal et al., 2021; Ramesh et al., 2022). Through this task, we can study how well CLIP can recognize different visual concepts in sketches, and the experiments can help us determine if CLIP can be used in similar ways to guide part-based sketch generation from language. Moreover, since we give CLIP the same pairs that were given to the annotators, we can learn if CLIP can understand how people are using language to describe the visual features of the sketches.

4.2 Method

CLIP stands for Contrastive Language-Image Pre-training, and it seeks to learn image representations that transfer to a wide range of downstream tasks such as image classification on datasets with a wide variety of source domains. To do so, it uses the task of pairing images with their corresponding captions for pre-training. CLIP has two main parts: a text encoder and an image encoder, and both can be transformers. To model our dataset, we fine-tune the pre-trained `ViT-B/32` CLIP model by leveraging the `Python clip` package. In this section, we will first introduce the details of the CLIP contrastive objective. We will then introduce the transformer-based text and image encoders.

4.2.1 Contrastive Objective

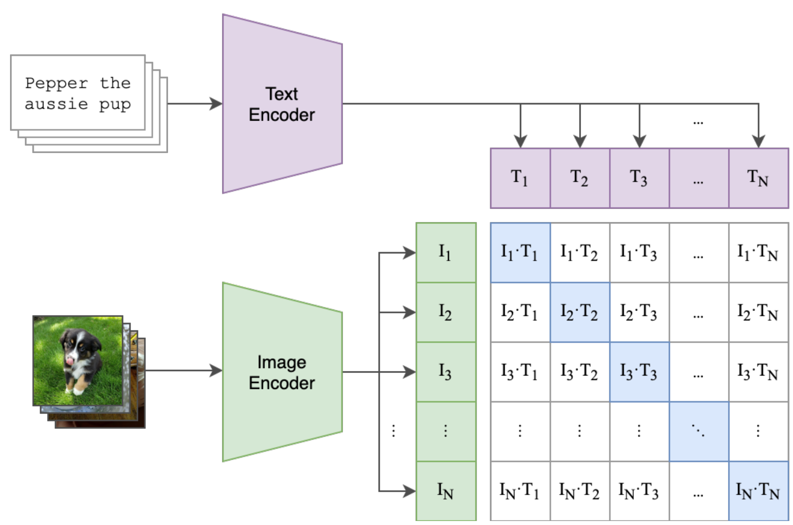


Figure 4.2: CLIP uses contrastive instead of generative objective for pre-training to learn joint vision-language embeddings. This image is taken from Figure 1 in the original CLIP paper (Radford et al., 2021). Each grid contains the dot-product of normalized image,text features.

Pre-training on an enormous amount of image-caption data with a contrastive objective, CLIP is able to learn robust vision-language joint embedding that allows it to perform on par with state-of-the-art (SOTA) models learnt with supervised objectives (Radford et al., 2021). During pre-training, for a batch of N (text,image) pairs, CLIP obtains N image features, $I_1, \ldots, I_N$, and N text features, $T_1, \ldots, T_N$, from the encoders. It then calculates dot-product between each pair of normalized vectors, where there are $N \times N$ possible pairing in total, as shown in the grid in Figure 4.2. This grid is a logit matrix X of dimension $N \times N$ and $X_{ij} = I_i \cdot T_j$. If we normalize the i-th row ($i \in [N]$) through softmax ($\frac{\exp\{X_{i,i}\}}{\sum_{j=1}^{N} \exp\{X_{i,j}\}}$), we obtain a distribution over the captions representing the likelihood of pairing the j-th caption with the i-th image. Similarly, normalizing the j-th column through softmax gives a distribution over all the images of how likely an image pairs with the j-th caption.

As explained in Radford et al. (2021), we can treat each batch as containing N visual concepts expressed through language. By normalizing the rows and columns into distributions, we perform classification over the N captions for the images and classification over the N images for the captions. Moreover, we know that the ground-truth is pairing the i-th image with the i-th caption. Therefore, for classification over captions, we have the ground-truth vector Y_I; for classification over images, we have the ground-truth vector Y_T, and we know:

$$Y_I = Y_T = \begin{bmatrix} 1 & 2 & \cdots & N \end{bmatrix}^T$$

Similar to standard multi-class classification, we can use cross-entropy loss as our objective function. The loss $L_I(X, Y)$ of selecting the correct caption for each image:

$$L_I(X, Y) = \frac{1}{N} \sum_{i=1}^{N} -\log \frac{\exp\{X_{i,i}\}}{\sum_{j=1}^{N} \exp\{X_{i,j}\}} \tag{4.1}$$

The loss $L_T(X, Y)$ of selecting the correct image for each caption:

$$L_T(X, Y) = \frac{1}{N} \sum_{j=1}^{N} -\log \frac{\exp\{X_{j,j}\}}{\sum_{i=1}^{N} \exp\{X_{i,j}\}} \tag{4.2}$$

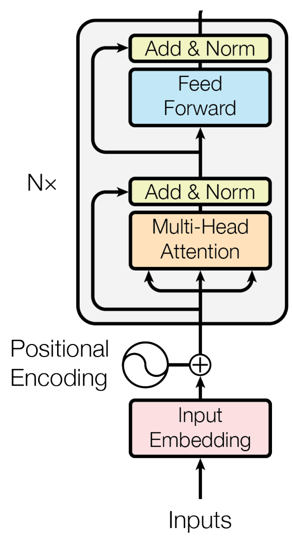
The final loss is defined as:

$$L = \frac{1}{2}(L_I(X, Y) + L_T(X, Y)) \tag{4.3}$$

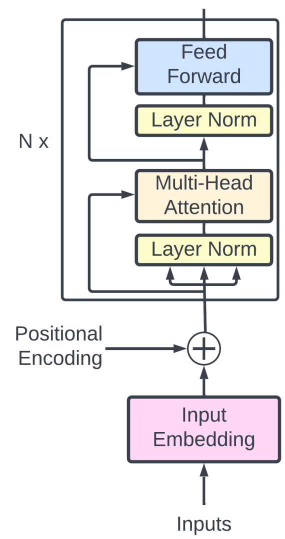
In order to minimize the cross-entropy loss, the model has to increase the logits on the diagonal of X, which means that it has to learn an embedding space where the feature vectors of the i-th image and the i-th caption are similar and the feature vectors of the other pairs are far apart. The term *contrastive* in the title of CLIP comes from this objective to pull together the real pair in the joint embedding space.

4.2.2 Text Encoder

Overview of Transformer Architecture



(a) Encoder architecture used in original Transformer paper; the figure is taken from Figure 1 in Vaswani et al. (2017).

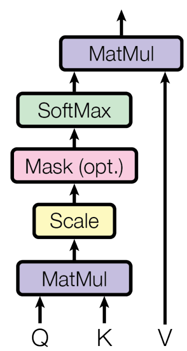


(b) Encoder architecture of CLIP (Radford et al., 2021).

Figure 4.3: Text encoder architecture: the text encoder is a transformer with stacked layers of self-attention and feed-forward network sublayers. The main implementation difference between the original Transformer (on the left) and the transformer implemented by CLIP is the placement of layer normalization.

The text encoder of CLIP is based on the transformer architecture introduced in Vaswani et al. (2017). Transformer relies only on self-attention mechanism to compute a representation for the input sequence. In this way, it alleviates the computation inefficiency and difficulty capturing long-range dependencies witnessed in recurrent layers.

Figure 4.3a and 4.4 are the same figures used in Vaswani et al. (2017) to illustrate the transformer architecture. In Figure 4.3a, Vaswani et al. (2017) gives an overview of the encoder architecture. Firstly, tokenized texts go through an embedding layer; the input embeddings are summed with learned position embeddings that inject information on order of the sequence. The input is computed using Byte Pair Encoding (BPE) with a $49,152$ vocabulary. As explained in the GPT-2 paper, BPE, a sub-word tokenization scheme, strikes a good balance between word-level and character-level word embeddings, since one works well with common words and the other with rare sequences (Radford et al., 2019).



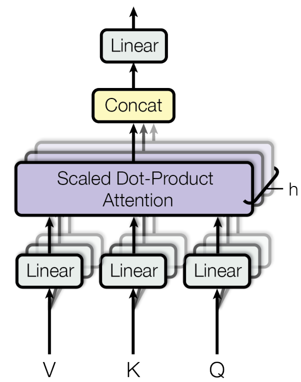


Figure 4.4: An illustration of the scaled dot-product attention on the left; on the right, an illustration of the multi-head self-attention mechanism (MSA) used in every layer of the transformer. Both figures are from the original Transformer paper (Vaswani et al., 2017).

Next, the input is passed through stacked layers multi-head self-attention (MSA) mechanism followed by point-wise feed-forward networks (FFN). In the version of CLIP that we use, the text encoder is a 12-layer transformer, and the model dimenison is 512, $d_{model} = 512$, meaning that output of the initial embedding layers, MSA, FFN all have dimension 512.

Compare to the formulation $LayerNorm(x + Sublayer(x))$ used in Vaswani et al. (2017) (Figure 4.3a), the CLIP text encoder uses $x + Sublayer(LayerNorm(x))$ (Figure 4.3b), where $Sublayer$ refers to either MSA or FFN. Each layer still contains residual connection and layer normalization, but the order is switched between Vaswani et al. (2017) and Radford et al. (2021).

Multi-Head Self-Attention

As illustrated in Figure 4.4, given query, key, value matricies Q, K, V (in our case, all three matrices equal to the input text embeddings), transformer uses different linear projections to create multi-head attention, and Vaswani et al. (2017) explains the benefit of multi-head attention as allowing the model to attend simultaneously to multiple representation subspaces of the input.

$$
\begin{aligned}
MultiHead(Q, K, V) &= Concat(head_1, \ldots, head_h)W^O \\
head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \\
Attention(Q, K, V) &= softmax(\frac{QK^T}{\sqrt{d_k}})V
\end{aligned}
\tag{4.4}
$$

ViT-B/32 uses a version with $h = 8$ attention heads, so $W_i^Q, W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, where $d_k = d_v = \dfrac{d_{model}}{h} = \frac{512}{8} = 64$. At the end of the multi-head attention mechanism, the weighted combination of values from each head is concatenated together and passed through a linear layer, represented here as $W^O \in \mathbb{R}^{(d_v \times h) \times d_v}$. CLIP uses the "Scaled Dot-Product Attention" in Vaswani et al. (2017), illustrated in details in Figure 4.4. The dot products between query and key determine the weights that are used to sum the values; in this way, we have a contextualized representation; compared to convolutions that use static kernels, attention weights are dynamic.

4.2.3 Vision Transformer

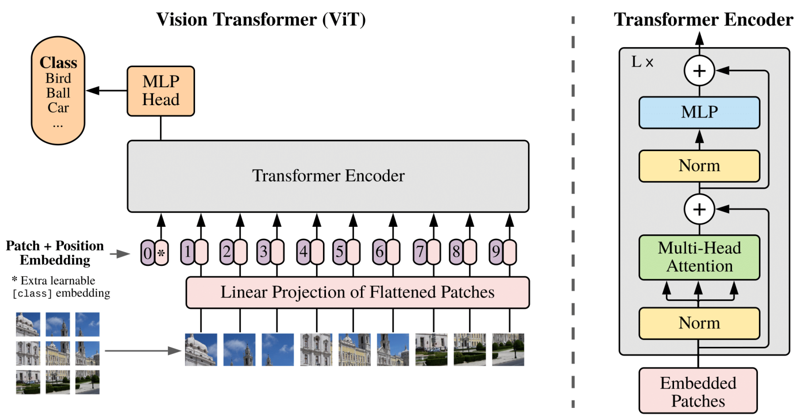


Figure 4.5: Vision Transformer (ViT) architecture (Dosovitskiy et al., 2020).

The image encoder of CLIP uses Vision Transformer (ViT) introduced in Dosovitskiy et al. (2020). The architecture of ViT is based on the original transformer introduced in Vaswani et al. (2017). In order to reuse the transformer model, ViT needs to first turn an image of size $H \times W \times C$, (H, W, C stands for image height, width, channel size, respectively), into a sequence of "tokens", similar to the text input. To do so, Dosovitskiy et al. (2020) reshapes the image to size $N \times (P^2 \cdot C)$, where N is the number of patches and P the patch size; the reshaped image can be seen as a sequence of N image tokens, each having a dimension of $P^2 \cdot C$. Each image token is then passed through a linear layer to be mapped to dimension D, similar to the model dimension d_{model} earlier.

In the version of CLIP that we used, all input images have dimension 224×224, the patch size $P = 32$, and the output of the initial embedding layers, MSA, FFN all have dimension 768 ($D = 768$). As explained in Dosovitskiy et al. (2020), before passing into the transformer, we also need to prepend a `[class]` token at the front the sequence, whose embedding at the last layer of the transformer will be used as the representation for the entire image. In this way, each image is represented as a sequence of $7 \times 7 + 1 = 50$ tokens, illustrated as purple boxes in Figure 4.5. The pink boxes that are right next to the patch embeddings represent position embeddings, with a similar function to encoder sequence order information as in the text transformer. For the CLIP image encoder, layer normalization is applied to the input before passing into the transformer and to the output at the last layer.

4.2.4 Fine-Tuning CLIP

We fine-tune CLIP with our dataset containing (sketch, part description) pairs. Our sketches come from the QuickDraw dataset (Ha & Eck, 2017), and the semantic part annotations come from the SPG dataset (Li et al., 2018).

Sketch Pre-Processing

The QuickDraw sketches are stored in vector format: each sketch is composed of a sequence of n strokes $S_i, i \in [n]$, and each stroke S_i is a sequence of vectors $(\delta x, \delta y, p, l)$. δx and δy are changes in the x, y coordinates with respect to the previous point; for the first point, its coordinate is with respect to the point $(25, 25)$. All points are assumed to be drawn on a 256×256 canvas. $p = 1$ if the point is the last point in the current stroke, and $p = 0$ otherwise. The SPG dataset provides annotation for semantic segmentation of the sketches, and l is an integer representing which semantic part the current point belongs to. For angels, l is

Text Pre-Processing

We used the `spacy` package to preprocess the text. `spacy` provides trained natural language processing pipeline and includes models for, for example, token-to-vector and part-of-speech tagging. We use the `en_core_web_sm` pipeline and its lemmatizer to reduce words to their basic forms.

Moreover, we lower-case all words and remove punctuation, a list of which is provided by `Python string` package, `string.punctuation`. We also remove words like *shaped*, *sized*, *and*, *like*, since they act like stop words and do not provide additional visual descriptions of the sketches. Text descriptions are also tokenized by CLIP's tokenizer before passing into CLIP text encoder.

Given n pairs of two sketches and two part annotations, the same pairs that were provided by the annotators, we calculate an accuracy-like metric:

$$acc = \frac{\sum_{k=1}^{n} \sum_{j=1}^{2} \mathbb{1}(f(j) = j)}{2n}$$

Given (s_1, s_2), we use CLIP image encoder (zero-shot/fine-tuned) f_v to extract visual features for the two sketches, $f_v(s_1) \in R^{512}$, and $f_v(s_2) \in R^{512}$. We then use the zero-shot/fine-tuned CLIP text encoder to extract the text features for the part descriptions, namely we fill in the template $t =$ `[ADJ] [PART NAME]`, where `[ADJ]` is filled with the adjective phrases annotations, and `[PART NAME]` is the name of the part in the sketches. For angels, `[PART NAME]` is one of *halo*, *eyes*, *nose*, *mouth*, *body*, *outline of face*, *wings*; for face, `[PART NAME]` is one of *eyes*, *nose*, *mouth*, *hair*, *outline of face*. After filling in the above template, we obtain the part annotations for the two sketches t_1, t_2. We obtain embeddings for the part annotations by encoding them through CLIP text encoder f_t: $f_t(t_1) \in R^{512}$, and $f_t(t_2) \in R^{512}$. We then calculate cosine similarity between all four pairs of $(f_v(s_i), f_t(t_j))$, $i, j \in [2]$, where consine similarity between two vectors u, v is defined as $S_c(u, v) = \frac{u \cdot v}{\|u\|\|v\|}$.

$$\begin{aligned}
&I_1, I_2 \\
&T_1, T_2 \\
S_c(I_i, T_j) &= \frac{I_i \cdot T_j}{\|I_i\|\|T_j\|} \\
S_c(u, v) &= \frac{u \cdot v}{\|u\|\|v\|} \\
f(j) &= \max_i S_c(f_v(s_i), f_t(t_j)) \qquad i \in [2]
\end{aligned} \tag{4.5}$$

Therefore, given that our entire pipeline is f, $f(j) \in [2]$ output which of the two sketches t_j will be paired with, and

$$f(j) = \max_i S_c(f_v(s_i), f_t(t_j)) \qquad i \in [2]$$

.

Chapter 5

Results & Analysis

In this chapter, we present experiment results and analysis to understand the language used to describe sketch parts. We found the following:

1. CLIP, despite its large pre-trained corpus, cannot easily generalize to unseen category on the task of pairing sketches with their descriptions. This indicates that the relationship between natural images and language cannot be easily translated to sketches and their descriptions. (Section 5.1)

2. Average cosine distance has increased between a pair of descriptors that are used by annotators to differentiate two sketches, suggesting that word embeddings extracted from the fine-tuned CLIP can better reflect the contrasting nature of these pairs, compared to BERT or pre-trained CLIP. (Section 5.2)

5.1 Classification Experiment

As explained in Section 4.1, annotators provide descriptions (t_1, t_2) to differentiate the two sketches (s_1, s_2) presented to them, and from this process, we have the ground-truth pairing of (s_1, t_1) and (s_2, t_2). We use CLIP to decide which sketch should be paired with description t_j, and the results are reported in Table 5.1.

Model	Face		Angel	
	Test	Dev	Test	Dev
zero-shot	53.8	54.6	56.4	57.2
finetuned on face	70.1	68.5	58.1	57.5
finetuned on angel	58.1	61.0	67.2	70.5
finetuned on face + angel	74.3	71.7	70.4	71.0

Table 5.1: Accuracy (%) of CLIP on choosing the correct sketch for a given text description from a pair of sketches. During annotation, the annotator is given the pair of sketches and provided descriptions of certain parts in the sketches for both sketches.

Compare to the zero-shot performance, fine-tuning on a single category would increase performance significantly on the category: for face, accuracy increased 16.3% on the test set after fine-tuning on face; performance increased 10.8% on the angel test set after fine-tuning on angel. The increase demonstrates that CLIP has adapted its embedding space and has learnt to associate sketches with their part descriptions. However, CLIP fine-tuned on a single category cannot generalize easily to unseen category: only 1.7% increase on angel when fine-tuned on face, and 4.3% increase on face when fine-tuned on angel. The two categories share vocabulary, and similar shapes appear in the two categories, but CLIP does not generalize as easily as expected. Concepts like *big* and *small* appear in both categories, but it is not guaranteed that CLIP can recognize them in sketches from unseen category.

Transferring from angel to face results in slightly better performance compared to transferring from face to angel (+4.3% vs. +1.7%). One likely explanation is that the angel category contains more sketches (787 vs. 572) and a larger vocabulary (1107 vs. 833), so CLIP performed better because it has seen more sketches and familarized itself with the domain, and it has learnt from a wider variety of ways that sketches can be described. A second explanation is that angels contain all face parts (*eyes,nose,face outline,mouth*) except *hair*, but face sketches do not contain *body,wings*, and *halo*, which are the most definitive features of angels. Therefore, CLIP, after learning how face parts are described through a few angel sketches that contain them, can recognize face sketches better. However, when it encounters angel sketches with unseen parts and new usage of words seen in the face category, it fail to transfer the learnt concepts to angels.

Fine-tuning on both categories results in slightly better performance than fine-tuning on a single category. CLIP has the capacity to learn how different words are used in both categories, but it cannot transfer the learnt concept across categories. Moreover, as explained in Section 4.2.1, CLIP uses words in the WordNet synsets to search for images to construct the datasets used for pre-training, and 88% of all words in our datasets exist in WordNet, meaning that CLIP has seen them and how the visual concepts are demonstrated in images, but it cannot adapt the concepts to a different visual domain, sketches, without fine-tuning from sketches in both categories.

5.2 Word Similarity Experiment

Figure 5.1: An example pair of sketches with different eyes that was presented to annotators. We do not pre-define a list of attributes and ask the annotators to determine whether the sketches are different in these attributes; instead, we highlight the parts in colors and implicitly prompt them to pick up on the differences. As expected, most annotations naturally include the size and shape differences. From the descriptions, we can extract pairs of *antonyms* used to indicate opposite visual concepts. In this case, the annotation was *large circular eyes* for the sketch on the left and *tiny solid eyes* for the one on the right.

In order to learn more about how have relationships between word embeddings changed through fine-tuning, we use cosine similarity as a metric to measure how close two words are. During annotation, we present two sketches with contrasting parts (for example, a face with big eyes and another with small eyes, like Figure 5.1) to the annotators and ask them for descriptions of the parts. We do not explicitly tell them that the eyes are different in sizes, yet it is highly likely that they would provide descriptors that can indicate this visual differences. We extract all tuples of words from a pair of contrasting descriptions. In the case of Figure 5.1, one annotator provide the descriptions *large circular eyes* for the sketch on the left and *tiny solid eyes* for the other; from these two descriptions, we can extract 4 pairs of contrasting descriptors: (*large, tiny*), (*large, solid*), (*circular, tiny*), and (*circular, solid*). There are a total of $9,823$ pairs, and the top 20 most frequent pairs are shown in Table 5.2. For example, the pair *small* and *large* are use 228 times by annotators to contrast certain parts in the two sketches across face and angel sketches. We consider two words in each pair as "antonyms", because they are used to represent two *opposite* visual concepts in the two sketches. It is in cases like this that our common understanding of synonym and antonym falls short: (*large, tiny*) might align with most people's idea of two words opposite in meaning, and (*large, solid*) do not appear as antithesis of each other, but the pair is used for the purpose of differentiating the sketches. Therefore, from a pragmatic perspective, these words are antonyms in our dataset.

word 1	word 2	#occurrences
small	large	228
circular	oval	142
wide	narrow	124
small	round	112
wide	small	108
oval	small	100
circular	small	94
oval	round	92
close	open	83
big	small	71
short	long	68
curve	wide	68
thin	wide	61
open	small	60
oval	wide	60
large	round	55
round	uneven	53
oval	open	51
open	circular	47
curve	small	46

Table 5.2: 20 pairs of descriptors most frequently used by annotators to differentiate parts in face and angel sketches. There are a total of 9823 pairs, and 7194, or 73.2%, pairs are used only once in the dataset.

	avg cosine sim
zero-shot	0.833
finetuned on face	0.752
finetuned on face + angel	0.691

Table 5.3: Average cosine similarity.

After fine-tuning CLIP to learn to contrast sketches with these descriptions, we expect that the similarity between contrasting words to decrease. Indeed, as shown in Table 5.3, we see a trend that as CLIP learns from more pairs of (sketch, part description) in our dataset, the word embeddings of contrasting words become dissimilar.

Across the entire dataset, there are $\binom{1450}{2}$, about 1 million, pairs of words, whether they are used in contrasting descriptions or not, and 99.9% of the pairs decreased in cosine similarity, from pre-trained CLIP to CLIP fine-tuned on face and angel sketches. One possible explanation is that we collected our data by presenting two sketches with contrasting features, but it could also be a general trend of CLIP fine-tuning on additional datasets, and we need future studies to give an definitive answer.

We only see increase in cosine similarity in 824 pairs of words. When we look at the top 20 pairs of words with the most increase in cosine similarity, we see that 18 pairs include the word *slash*, which appears only once in our entire dataset. The increase is less than 0.06, compared to a decrease of 0.65 for the pair that has the largest decrease in similarity. If we only look at pairs of words that are among the 250 most frequent words, pairs whose cosine similarity has increased the most are *arrow, smiley*, *circle, arrow*, *trilateral, triangular*. However, the increase is still too marginal, ≈ 0.02, for us to conclude anything meaningful about the text embeddings of the fine-tuned CLIP model. Therefore, we cannot provide reasons for the marginal increase, and it is likely to be a chance event due to rare occurrences.

Chapter 6

Future Work

Figure 6.1: .

There are many creative use of language in our dataset, and in follow-up work, we want to investigate what categories of creative usage are there and quantify how many cases are in each category. Most of our current characterization of the dataset are based on looking through a few examples in the datasets (there are a total of 11K!). Since we did not put limits on what language annotators could use during data collection, such as specifying a fixed list of attributes for the annotators to choose from, the dataset contains a variety of ways of describing the same concept. For example, for a nose that looks similar to the one in Figure 6.1, annotators describe it as *hook-shaped*, *c-shaped*, *inverted j*, *curved*, etc. We believe that this case is not the only one, and we want to quantify the diversity. In this way, we can evaluate how a multi-modal embedding space, such as the CLIP joint embedding, captures these relationships. Is the feature vector of the sketch collinear with the word features of every one of these words?

So far, we have observed that face and angel sketches share many visual concepts. These could be related to length (*long,short*), size (*big,small*), geometry (*circular, triangular*), direction (*horizontal, vertical*), and many more. We want to evaluate more thoroughly which concepts are shared and which ones are unique to each category. In this way, we can understand better the chalenges around generalizing CLIP to unseen categories, indicated by results in Section 5.1.

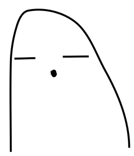
Appendix A

Designing the Requirement Section for the Prompt-Guided Sketch Text Dataset

An excerpt from an old version of the instructions, in which we tried to explain a single *step* of the annotation:

> In each task, we show **1 prompt** from which we would like to get
>
> 1. A drawing containing 1 **entity** that you think illustrates the prompt.
> 2. Text annotation for every **"component"** that makes up the entity. The word "component" is intentionally vague, and it depends on how you compose your drawing. For example, in the above example, the prompt is "smiley face", and during the process of creating a "smiley face" entity, we used 4 components: a face, a left eye, a right eye, and a mouth. For each component, you can annotate with "face", "left eye", "right eye", "mouth", respectively; you can also annotate with more details describing the shapes of each component: "face that looks like an arc opening downwards", "a left eye that is an arc", "a right eye that looks exactly like the left eye", "an arc-like mouth". Try to use creative and descriptive languages. You can draw a component using multiple strokes.

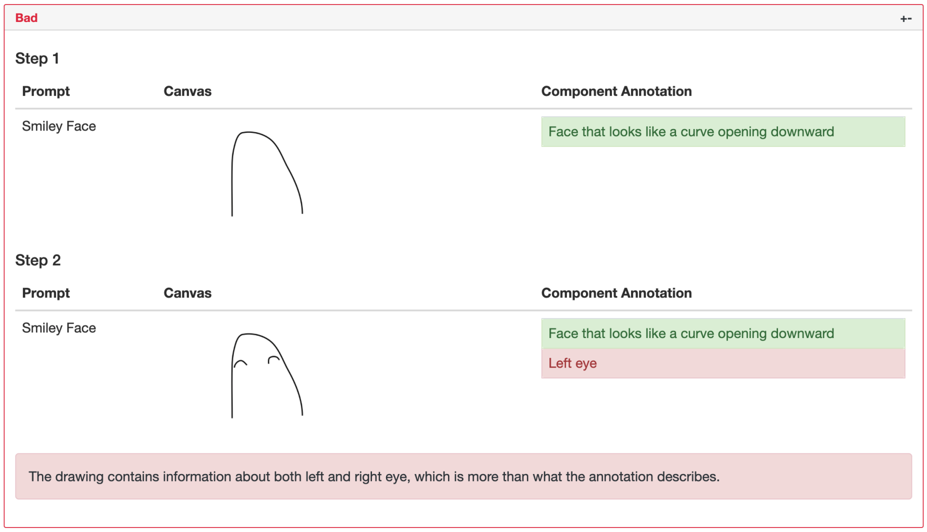
We also need to come up with examples explaining each requirement. We select a few major sub-versions of the requirements resulted from circulating the interface within our lab.



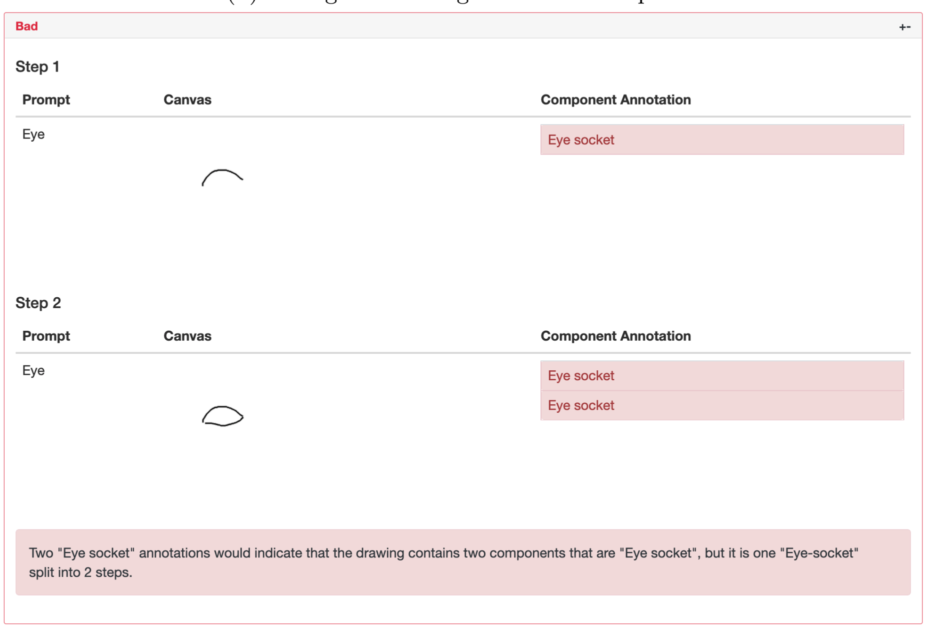
(a) An example included in the first version of the requirements, explained in more details in item 3.

1. Do not draw entity that does not respond to the prompt. For example, given the prompt *Smiley Face*, the drawing should not contain irrelevant objects like a house.
2. Do not draw more than one entity that responds to the prompt. For example, One should not draw two *Smiley Face* entities, although each *Smiley Face* entity is good. However, you can draw multiple tree objects to illustarte the prompt *Forest*.
3. Do not draw entity that is ambiguous in terms of illustrating the prompt. For example, the drawing (Figure A.1a) looks more like a *Sad Face* than *Smiley Face*.
4. Do not draw one component that contains more information/content than what the annotation for that component describes. (A counterexample is illustrated in Figure A.1b.)
5. Do not split the drawing of a component into multiple steps, unless you can annotate each step separately. (A counterexample is illustrated in Figure A.1c.)
6. Do not annotate a component more than once.

1. Draw *one* item at a time and provide its corresponding annotation. For example, the text annotation says "left eye", but two items are drawn: a left eye and a right eye.
2. The annotation should describe its corresponding item in the drawing *entirely*.
3. The annotation should name the item.
4. Desired properties of good drawings:
   - Contain as many items as possible, but be sure that they all contribute to illustrating the prompt. For example, draw more than just two eyes for a face.
   - Use shapes creatively. For example, draw a triangle for the left eye, and annotate accordingly with "triangular left eye that shows suspicion".



(b) Unaligned drawing and text description.



(c) An example of misalignment: text description *overflow* into multiple steps.

Figure A.1: Screenshots of counterexamples used in first version of the requirements in Version 1.

5. Desired properties of good annotations:

   - Use descriptive languages. For example, “a left eye that looks an arc facing downward”.
   - Include the intention/purpose of drawing an item. Explain in the annotation reasons for drawing the item. For example, “thumbs-up next to the face that really shows how happy

the face is".

1. Each drawing should contain at least 2 steps.
2. Annotation of each step should include at least the name of the drawn object(s).
3. If draw multiple copies of the same object, draw each object in a separate step and give different annotations by using, for example, cardinal or ordinal numbers. (An example shown in Figure A.2)
4. Differentiate between plural and singular forms.
5. The name of the whole should not be used for its parts. (An example shown in Figure A.2)
6. The word "right" always refers to this side: $\Longrightarrow$

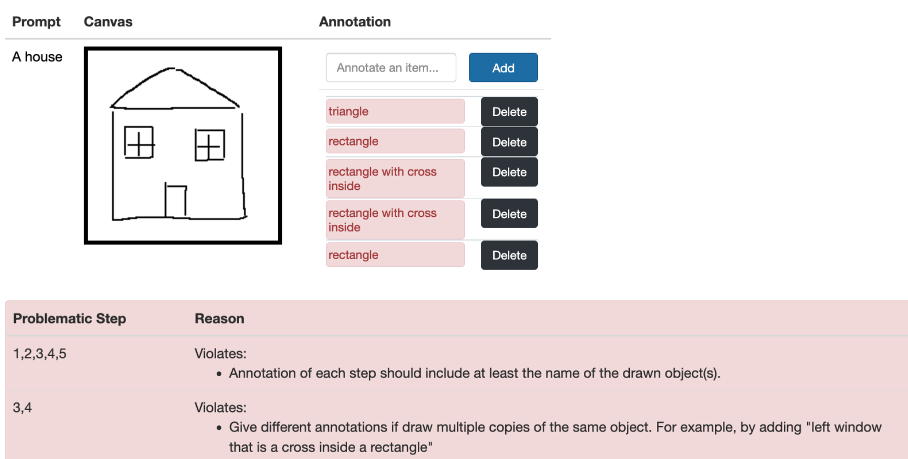


Figure A.2: Screenshots of counterexamples used in third version of the requirements.

Bibliography

Allado-McDowell, K., & Okojie, I. (2020). *Pharmako-ai.* Ignota.

Bau, D., Strobelt, H., Peebles, W., Wulff, J., Zhou, B., Zhu, J., & Torralba, A. (2019). Semantic photo manipulation with a generative image prior. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, *38*(4).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). *Language models are few-shot learners.* arXiv. Retrieved from `https://arxiv.org/abs/2005.14165` doi: 10.48550/ARXIV.2005.14165

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby, N. (2020). *An image is worth 16x16 words: Transformers for image recognition at scale.* arXiv. Retrieved from `https://arxiv.org/abs/2010.11929` doi: 10.48550/ARXIV.2010.11929

Eitz, M., Hays, J., & Alexa, M. (2012). How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, *31*(4), 44:1–44:10.

Frans, K., Soros, L. B., & Witkowski, O. (2021). *Clipdraw: Exploring text-to-drawing synthesis through language-image encoders.* arXiv. Retrieved from `https://arxiv.org/abs/2106.14843` doi: 10.48550/ARXIV.2106.14843

Gal, R., Patashnik, O., Maron, H., Chechik, G., & Cohen-Or, D. (2021). *Stylegan-nada: Clip-guided domain adaptation of image generators.* arXiv. Retrieved from `https://arxiv.org/abs/2108.00946` doi: 10.48550/ARXIV.2108.00946

Ge, S., Goswami, V., Zitnick, C. L., & Parikh, D. (2020). *Creative sketch generation.* arXiv. Retrieved from `https://arxiv.org/abs/2011.10039` doi: 10.48550/ARXIV.2011.10039

Ha, D., & Eck, D. (2017). *A neural representation of sketch drawings.*

Li, K., Pang, K., Song, J., Song, Y.-Z., Xiang, T., Hospedales, T. M., & Zhang, H. (2018). *Universal perceptual grouping.* arXiv. Retrieved from `https://arxiv.org/abs/1808.02312` doi: 10.48550/ARXIV.1808.02312

Nichol, A., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., ... Chen, M. (2021). *Glide: Towards photorealistic image generation and editing with text-guided diffusion models.* arXiv. Retrieved from `https://arxiv.org/abs/2112.10741` doi: 10.48550/ARXIV.2112.10741

Patashnik, O., Wu, Z., Shechtman, E., Cohen-Or, D., & Lischinski, D. (2021). *Styleclip: Text-driven manipulation of stylegan imagery.* arXiv. Retrieved from `https://arxiv.org/abs/2103.17249` doi: 10.48550/ARXIV.2103.17249

Qi, Y., & Tan, Z.-H. (2019). Sketchsegnet+: An end-to-end learning of rnn for multi-class sketch semantic segmentation. *IEEE Access*, *7*, 102717-102726. doi: 10.1109/ACCESS.2019.2929804

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... Sutskever, I. (2021). *Learning transferable visual models from natural language supervision.* arXiv. Retrieved from `https://arxiv.org/abs/2103.00020` doi: 10.48550/ARXIV.2103.00020

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners..

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). *Hierarchical text-conditional image generation with clip latents.* arXiv. Retrieved from `https://arxiv.org/abs/2204.06125` doi: 10.48550/ARXIV.2204.06125

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... Sutskever, I. (2021). *Zero-shot text-to-image generation.* arXiv. Retrieved from `https://arxiv.org/abs/2102.12092` doi: 10.48550/ARXIV.2102.12092

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). *Generative adversarial text to image synthesis.* arXiv. Retrieved from `https://arxiv.org/abs/1605.05396` doi: 10.48550/ARXIV.1605.05396

Sangkloy, P., Burnell, N., Ham, C., & Hays, J. (2016, jul). The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Trans. Graph.*, *35*(4). Retrieved from `https://doi.org/10.1145/2897824.2925954` doi: 10.1145/2897824.2925954

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). *Attention is all you need.* arXiv. Retrieved from `https://arxiv.org/abs/1706.03762` doi: 10.48550/ARXIV.1706.03762

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). *The Caltech-UCSD Birds-200-2011 Dataset* (Tech. Rep. No. CNS-TR-2011-001). California Institute of Technology.

Xu, G., Kordjamshidi, P., & Chai, J. Y. (2021). *Zero-shot compositional concept learning.* arXiv. Retrieved from `https://arxiv.org/abs/2107.05176` doi: 10.48550/ARXIV.2107.05176

Xu, P., Hospedales, T. M., Yin, Q., Song, Y.-Z., Xiang, T., & Wang, L. (2020). *Deep learning for free-hand sketch: A survey.* arXiv. Retrieved from `https://arxiv.org/abs/2001.02600` doi: 10.48550/ARXIV.2001.02600

Xu, T., Zhang, P., Huang, Q., Zhang, H., Gan, Z., Huang, X., & He, X. (2017). *Attngan: Fine-grained text to image generation with attentional generative adversarial networks.* arXiv. Retrieved from `https://arxiv.org/abs/1711.10485` doi: 10.48550/ARXIV.1711.10485

Yu, Q., Liu, F., Song, Y.-Z., Xiang, T., Hospedales, T. M., & Loy, C. C. (2016). Sketch me that shoe. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 799-807). doi: 10.1109/CVPR.2016.93

Yuan, S., Dai, A., Yan, Z., Guo, Z., Liu, R., & Chen, M. (2021). Sketchbird: Learning to generate bird sketches from text. In *2021 ieee/cvf international conference on computer vision workshops (iccvw)* (p. 2443-2452). doi: 10.1109/ICCVW54120.2021.00277