

# Gestural Inputs as Control Interaction for Generative Human-AI Co-Creation

John Joon Young Chung<sup>1</sup>, Minsuk Chang<sup>2</sup> and Eytan Adar<sup>1</sup>

<sup>1</sup>University of Michigan, Ann Arbor, MI

<sup>2</sup>Naver AI Lab, Seongnam, Republic of Korea

## Abstract

While AI-powered generative systems offer new avenues for art-making, directing these algorithms remains a central challenge. Current methods for steering have focused on conventional interaction techniques (widgets, examples, etc.). This position paper argues that the intersection of user needs in creative contexts and algorithmic capabilities requires re-thinking our interactions with generative AI. We propose that rough gestural inputs, such as hand gestures or sketching, can enhance the experience of human-AI co-creation—even for text. First, the undetermined and ambiguous nature of gestural inputs corresponds to the purpose and the capabilities of generative systems. Second, rough gestural can be intuitive and expressive, facilitating iterative co-creation. We discuss design dimensions for inputs of artifact-creating systems, then characterize existing and proposed input interactions with those dimensions. We highlight how gestural inputs can expand the control interaction for generative systems by analyzing existing tools and describing speculative input designs. Our hope is that gestural inputs become actively studied and adopted to support user intentions and maximize the perceived efficacy of generative algorithms.

## Keywords

generation, controllability, gestural input

## 1. Introduction

Technologies such as Generative adversarial network (GAN) [1], and pretrained language models (PLM) [2] have the potential to enable human-AI co-creation. These algorithms are attractive in creative contexts as the AI can generate novel creations—something the human hadn't considered. However, this process can backfire when novelty and surprise misalign with the user's intention and preference. Most commonly, produced text can suddenly turn from what the author wants. Users often have to re-run the algorithm and iterate until they get the desired results. Without control, users can only hope that the next generation will be better than the last. Thus, controllability becomes key to effective iteration. The best controls go beyond steering the behavior of the algorithm. They also manage the user's expectations of what the algorithm will produce. There are many conventional ways to provide interactive control but without addressing these goals effectively. This paper proposes that rough gestural 'sketches' coupled with abstract representations of content (i.e., information visualizations) can facilitate control interaction for generative algorithms.

Our proposal is strongly motivated by limitations in

existing control interactions for generative algorithms. Current control interactions range from inputting a simple number (e.g., *have a violin play with the maximum amount of vibrato, by setting the parameter value of 1.0* [3]) to using natural language prompts (e.g., *produce an image of a dragon sitting on a castle* [4]) to providing examples (e.g., *make this photograph look like this example from Picasso* [5]). We argue that these approaches are limited in different ways, particularly in co-creative tasks. For example, numerical inputs imply an 'exact' level of control. This over-promises and sets a very high expectation for the user: the system will produce exactly what was specified. Unfortunately, this does not often match algorithmic capabilities. Natural language prompts and example-driven interfaces are problematic for other reasons. Creative work often requires iteration and experimentation with alternatives. Prompts and examples do not readily support this iteration. Users may not understand why the algorithm did what it did, how the results can be corrected, or may simply be challenged to find or create new examples or prompts. The cost of iterative practice may make generative algorithms unappealing in practice.

In answer to many challenges for generative tools, we propose that rough gestural inputs, such as sketching, can be a sweet spot for human-AI co-creation. First, gestural input conveys imprecise and ambiguous intentions [6, 7], which corresponds to the nondeterministic nature of generative algorithms. Second, because impreciseness is allowed (e.g., simple brush strokes [8]), the interaction of specification would be easier. With easier interactions,


Joint Proceedings of the ACM IUI Workshops 2022, March 2022, Helsinki, Finland

✉ jjyc@umich.edu (J. J. Y. Chung); minsuk.chang@navercorp.com (M. Chang); eadar@umich.edu (E. Adar)

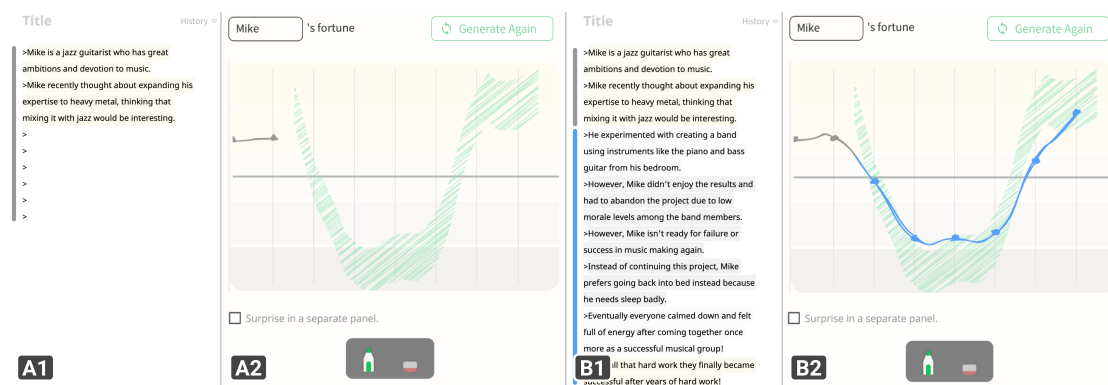
🌐 <https://johnr0.github.io/> (J. J. Y. Chung);

<https://minsukchang.com/> (M. Chang); <http://cond.org/> (E. Adar)

© 2021 Copyright © 2022 for this paper by its authors. Use permitted under

 Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)



**Figure 1:** TaleBrush facilitate human-AI story co-creation by allowing users to intuitively control and sensemake story generation with a line sketching interaction. Specifically, users can control the protagonist’s fortune with line sketching. In TaleBrush, the writer can write a portion of a story (A1) and sketch the change of the protagonist’s fortune as a control input (A2, green shaded area). In the sketched line, the x and y positions stand for the chronological story position and the protagonist’s fortune, respectively. For the protagonist’s fortune, the higher the position, the better the fortune. In the sketch, the width shows the possible variance in the fortune of the generated sentences. With the given line sketch, TaleBrush generates story sentences (B1, indicated with blue). These sentences are then visualized upon the original sketch (B2, the blue line and dots).

the end-user may not need to think carefully about the examples or prompts they generate, thus allowing for more rapid iteration.

The idea of gestural or sketching inputs in the context of generation has some history. For example, low fidelity sketches created by the end-users can guide the generation of photorealistic images [9]. Here the sketch is the *input* and is in the same modality as the output (e.g., take the *visual* dragon I scribbled and make a *visual* photorealistic version). The input indicates “what generation should be done.” *Control* is implemented through more standard interactive approaches (e.g., by adjusting this slider, I am indicating how to bias color selection for the dragon). This is not to imply a clear separation between input and control, as they are often inexorably connected as mechanisms to have a system produce the desired output. However, our specific suggestion is that the input interactions—and sketching and gesture, in particular—can be expanded to also control “how the generation should be done.”

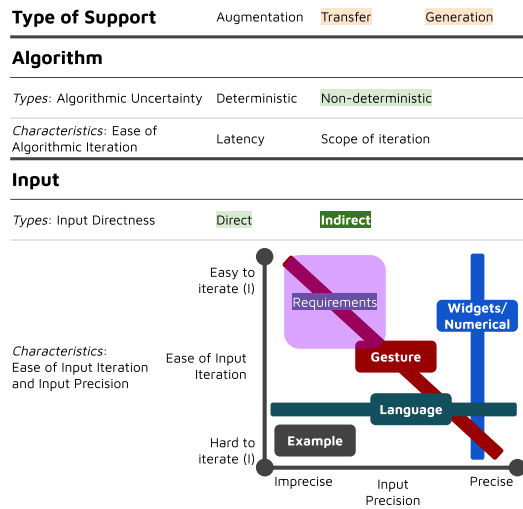
One example of a ‘sketch-as-control’ interaction is our controllable story generation system, TaleBrush (Figure 1). Here, TaleBrush leverages *abstract visual representation* of the character’s fortune to control the story generation. The canvas is a 2D plane that allows for the specification of the protagonist’s fortune (y-axis) and the story’s progression (x-axis). In this interface, the control interaction is as simple as a single stroke of a line. This approach has several benefits. First, it is easier to interact with than alternatives (e.g., having multiple sliders for different story parts). Most importantly, the ambiguous

and imprecise nature of sketching corresponds to the user’s ambiguous intentions and the algorithm’s uncertainty. This example also demonstrates how one input modality and representation (i.e., visual) can be used to guide a different output modality (i.e., textual).

In this paper, we expand on this idea. We first introduce the design dimensions for inputs of artifact-creation systems. These include the types of support one wants with an AI tool, considerations of algorithmic uncertainty, the precision of input, and ease of iteration on algorithms and inputs. Using design dimensions, we characterize different existing input types for generative human-AI co-creation. We specifically discuss how our example system, TaleBrush [10], adopts sketched inputs to facilitate iterative human-AI co-creation in story writing. Considering sketching and gestural inputs for control will enable new ways to support human-AI co-creation.

## 2. Design Dimensions of Artifact-Creation Support

We first consider possible dimensions for designing control interactions for creation support. We scope “creation support” to systems that help creatives *directly* implement artifacts. We exclude those tools that serve a more indirect role, such as critiquing the created artifact. This boundary is something we have previously considered in surveying the range of tools in the creative space [11]. We propose a focus on three aspects: 1) *type of support*, 2) *algorithm*, and 3) *input* (summarized in Figure 2).



**Figure 2:** Summary of the design dimensions of artifact-creation support in relation to designing generative co-creation tools. Orange items (transfer and generation) are two supports provided with generative co-creation tools. Items in green (non-deterministic, direct, and indirect) are design elements for generative co-creation tools (our focus is on indirect inputs). In indirect input design, the requirements for generative co-creation tools include: 1) easy and fast iteration and 2) algorithmic uncertainty matching the user’s expectation. These align with gestural inputs, in considering the ease of interaction and their ability to express the user’s ambiguous intentions. Note that the two-dimensional diagram in characteristics of input is drawn based on the qualitative analysis of different input approaches.

### 2.1. Type of Support

CSTs are an extremely diverse and broad category, even when restricted to direct influence [11]. Within this category we see tools that can *augment*, *transfer*, or *generate*. Though the last two categories are most relevant to our proposal, augmentation is also worth considering.

CSTs that provide *augmentation* support often enhance a task the creative is already doing through computational means. Many direct manipulation tools fall into this category. The least ‘intelligent’ of these replicate existing tools in a digital format. For example, a digital painting canvas has various types of digital brushes. Other augmentation tools provide some limited automation. For example, a bucket tool will flood-fill a closed area in a sketch. Most augmentation tools are highly deterministic. They are “predictable” and more naturally correspond to the user’s mental model of what the system will do. When using augmentation-focused tools, the user is firmly in control over both the idea and style of the final artifact.

The second and third types of support, *transfer* and

*generation*, use variants of generative algorithms. In contrast to the augmentation category, the end-user is ceding some creative control to the tool. Though, of course, the human maintains ultimate control over what makes it into the final artifact.

*Transfer* tools turn one artifact into another. A common feature is that they receive some ‘original’ artifact (e.g., a picture, a piece of text, a sketch, etc.) as input. The tool will then act on this input to generate a *variant*—often some alteration of the original input. A wide range of tools fall into this category, and they are often modality-specific. For example, in the visual design/art space, we see systems that transfer one visual art piece’s style to another image [5]. Other tools in the space will transform rough sketches into photorealistic images [9]. As with image-based style transfer, we find similar approaches for text where the software can transform the written input to the style of a particular author [12]. Though target styles are commonly required, not all transfer tools need them. In music, for example, there are tools that transform some input piece of music by adding effects like delay or compression [13].

Finally, we observe tools focused on *generation*. With these, the algorithm generates content from incomplete inputs or those of a different modality. For example, an input might be some previous part of the music, story, or some portion of drawings. The algorithm’s purpose is not to change this initial input, but rather to add to them. In music and text, these algorithms continue the from the user-provided ‘start’ or ‘infill’ when given some start and end states [14, 15, 16, 17]. In visual arts, we most often find this type of algorithm in systems that can fill empty spaces in an image [18, 19]. Note that many tools sit somewhere between transfer and generation and may depend on how the underlying task is defined. For example, we might have a tool that automatically colors a part of an image. From the perspective of the whole image, this may be *transfer* (especially if the input is some color palette or color model). However, because we are also *generating* new colors, we might treat the colorization task as generative.

Different types of tools will require different types of controls. However, there are similarities in user needs and expectations (e.g., surprise and novelty but also a willingness to cede some creative control to the software). This is in contrast to non-creative applications (e.g., predictive form filling) where ambiguity and surprise and undesirable. As we argue below, gestural and sketched inputs hold promise here.

## 2.2. Algorithm

### 2.2.1. Type: Algorithmic Uncertainty

A tool’s algorithmic pipeline can differ depending on how certain we are of the pipeline’s output. *Deterministic algorithms* are one extreme in that users can predict the result when using these algorithms. Direct manipulation implementations are, naturally, one example. When a box is dragged with a mouse cursor, the end-user knows where it will end up. Automated algorithms with clear rules are also deterministic. For example, with flood-fill (e.g., a bucket tool), the user knows the system will fill closed areas. If something goes wrong, the user can quickly isolate the problem.

On the other extreme are *non-deterministic* algorithms which represent many machine learning (ML) algorithms. Though powerful, the inferences made by these algorithms lead to increased uncertainty and failures. For example, in comic colorization, ‘flattening’ is the process of automatically creating colored polygons under different parts of the linear art (e.g., one for the face, one for the shirt, etc.). The algorithm for automated flattening makes inferences about shapes even when they are not ‘closed’ in an expected way. For example, creases in a drawing for a shirt may lead a poorly designed algorithm to make too many polygons or not connect them appropriately. Ideally, the system will produce one polygon that encapsulates the entire shirt. However, current flattening software is imperfect and can make the wrong inference. The algorithm’s uncertainty in what makes up the object can lead to unexpected bleeding [20]—a failure case.

However, in the creative setting, and specifically for generative algorithms, uncertainty can be a feature (rather than a failure). Or, more precisely, the line between a novel, desirable result, and an error are not necessarily clear cut. There is rarely a single gold standard for what should be generated, and the user might subjectively decide whether the output fits their goal.

### 2.2.2. Characteristics: Ease of Algorithmic Iteration

Iterative design is important in creating artifacts. This is mainly due to the explorative nature of the task. How easy it is to iterate depends on the algorithm’s properties. First among these is latency—the time taken by the algorithm for each cycle. The lower the latency, the easier the iteration. For example, when moving a box with a mouse cursor, the iteration is real-time as the box’s position instantly updates with the user’s movement. On the other hand, many generative algorithms take significant time to generate artifacts, significantly slowing down iteration.

A second algorithmic aspect that impacts iteration is scope. Here, we define scope as relating to what parts of

the artifact are iterated on. For example, with the direct manipulation of the box, only the position is changing, but not the color or size. Style transfer algorithms [21] are at the other extreme. With every run of these algorithms, the entire image (or many parts of it) will change.

Iteration naturally connects back to algorithmic uncertainty. If the user better understands the algorithm’s behavior (e.g., what the transfer algorithm changes and how), iteration may become easier. With high uncertainty, the user may need to iterate many times to get the effect or artifact they want.

## 2.3. Input

### 2.3.1. Type: Input Directness

An input method targets the artifact directly or indirectly [11]. With direct input, the end-user indicates the artifact or subject ‘target.’ Because of this directness, inputs are usually in the same medium as the target artifact. In some situations, a portion of the artifact can also be used as a direct input. For example, we can select a portion of the image or the story. At the other extreme are those inputs that do not directly impact the artifact but may give broad instructions on how the tool should implement something. The simplest example might be a slider control for some parameters. The user isn’t touching the artifact directly (i.e., the story or image) but the change in the slider guides the tool. The modality of indirect input can be far from the medium (e.g., visual arts as artifacts and numbers as inputs). As with our introductory example, abstract visual encodings can also be used for indirect inputs. In that example, the end-user drew the character’s fortune to produce text.

### 2.3.2. Characteristics: Input Precision

While there are numerous input approaches for artifact-creation systems, they vary on the spectrum of precision. These varying levels are helpful in different contexts. The most traditional type of widgets receives one specific value. Examples include a number in the slider or a category in a dropdown box. With this precise control, users will expect the output to react precisely.

Not all inputs need to be precise. *Natural language prompt* is one example and can handle a wider range of input precision [22, 23]. Roughly specified language would be imprecise, but at the same time, allow a high degree of freedom in how it can be interpreted. For example, asking for a “rough texture” can mean many things—anything from Jackson Pollock’s chaotic style to Van Gogh’s impressionism. However, language *can* support finer control. For example, if we say “move the selected square 3 pixels left,” this does not leave much room for misinterpretation.

At the imprecise end, we often find *Examples* as inputs [24, 21]. While they are often used as direct material for transfer (e.g., source of style in visual style transfer), it is up to the algorithm to determine, if it can, which aspects of the input should be followed closely and which are only suggestions. For example, when transferring the style of Van Gogh’s *The Starry Night*, it may not be clear whether the user wants the colors or textures to be transferred. Adding more examples might make the target clearer. However, it may be hard for the user to determine which attributes overlap between the examples and which are ambiguous. The interaction with uncertain algorithms makes this problem even more complex as it is not obvious if the issue is with the input or the inherent ambiguity of the system.

As with language prompts, *Gestural inputs*, such as sketches or hand gestures, can also have a wide range of precision. For example, gestural inputs for direct manipulation require outputs to follow the given input exactly. When resizing a box in graphics editors, users expect the box to follow the cursor they are moving. However, sketches can be used for low-precision input. For example, sketches can express flexible and lightweight ideas with their roughness, ambiguity, and uncertainty [7, 6]. Similarly, hand gestures have been used to provide imprecise but intuitive and flexible inputs, such as serving as rough scaffolds in 3D modeling [25]. As we see in these examples, gestural inputs can be designed to provide high intuitiveness and flexibility and be traded off against precision.

We note that input precision is often related to input difficulty. As we know from psychophysical properties such as Fitts’s Law [26], certain input precision comes at the cost of time or difficulty. Lower precision interactions, such as gestures, can often lower interaction difficulty.

### 2.3.3. Characteristics: Ease of Input Iteration

Just as we consider the iterative cost at the algorithmic level, it is worth considering it at the input level as well. Though these two might be tied, a tool might have relatively small back-end iterative costs but widely diverging front-end costs. For example, the algorithm itself might run quickly but generating good example inputs may take a long time. Thus, different input approaches vary in how well they support iteration.

Traditional input widgets, such as numerical values on sliders, are relatively easy. With a single slider, the control options given to the user are tightly restricted, and a change in value does not require much effort. However, even with simple slider widgets, the user needs to decide if a control should be changed and then make the actual change to the correct value. As the number of input controls grows, so does iteration cost.

Other types of inputs, such as natural language prompts

or examples, can increase iterative costs. This is mainly due to the vast space of options for these modalities. For natural language prompts, the user needs to come up with better wordings or more specific details on the prompts. This can be tricky if the user is to express differences in degree (e.g., how would one use language to express the level of roughness of the texture in a painting?). Similarly, iterating with examples is difficult because the user needs to search for more or better examples. If such an example can’t be easily found or created, the user will struggle to iterate.

Gestural or sketch inputs can help with iteration. While these input modalities come at the cost of precision, gestural input is flexible, intuitive with lowered cognitive demands. These properties can reduce iteration time. For example, the user can erase and redraw a portion of the sketches to quickly change the specifications.

## 3. Designing Generative Co-Creation Tools

The design dimensions above represent a large design space. However, we can begin to consider points in the space that are either required, or are more suitable, for co-creation tools.

### 3.1. Requirements for Generative Co-creation Tools

As we argued above, generative algorithms are usually used to support transfer or generation. Additionally, these systems have increasingly leaned towards machine-learning-based approaches. Thus, we are largely working in the non-deterministic algorithmic space and this implies a couple of key requirements for tools.

First, **iteration should be easy and fast**. In creative tasks, iteration and exploration are necessary as they expose the artist to more options and, eventually, a concretization of ‘direction’ [27, 28]. Thus, users of creative tools often *want* to be able to iterate, which is well-aligned with the reality that to use non-deterministic tools, they *need* to iterate. Unfortunately, sometimes the cost of iteration becomes high. Thus, tools should either act to speed up the number of iterations and, if that is not possible, to reduce them. In both situations, reducing the iteration cost is critical.

Second, **algorithmic uncertainty should match the user’s expectations**. With standard algorithms, we would only need to worry about the user’s expectations in how their input and deterministic output relate. For example, dragging an icon into the trash would lead to it being deleted. However, with non-deterministic algorithms, instead of a specific output, they would need to

model a range of possible outputs. Without this understanding, end-users are likely to be dissatisfied with the results. They will also find it difficult to model how their input choices will lead to a better, or more certain, output. Our advantage in creative tools is that some degree of uncertainty is actually a desired property. Our goal is not necessarily to make the tool appear deterministic as creativity often requires ‘surprise.’ Thus, users both want and expect some level of (controlled) uncertainty. A user may be willing to make a rough specification. At the same time, they would understand and expect that the tool will have some degrees of freedom within that space. Note that none of this is to say that we need to force the algorithms to match the user’s expectations. In some cases, users might not have well-defined expectations. In other cases, we may change their expectations.

### 3.2. Algorithmic Design

There are various ways to approach the iteration and uncertainty problems on the algorithmic side. For example, by adding extensive controllability features, we can provide the user with fine-grained controls for steering the behavior of the generative algorithms. However, this requires building algorithms that can actually accept all these controls.

On the positive side, detailed control may reduce the number of iterations, at least from the algorithmic perspective. That is, fine-grained controls would reduce the ambiguity of the input and enable the generative system to produce a more targeted response. Detailed controls also work to ‘teach’ the end-user how to model and direct the underlying algorithm. Their expectations of system capabilities would come to be more in alignment with reality with fewer iterations. Of course, reducing the latency of the algorithm would also facilitate the ease of iteration. Clever designs, such as using smaller models before executing more costly larger ones may help here.

However, shifting the responsibility of satisfying our requirements to the algorithmic side exclusively is not realistic. Regardless of the algorithm, many bottlenecks for iteration are from the interaction side. Increasing the number of controls may be cognitively costly for the end-user. This is not to say that fewer controls or simpler inputs, such as examples or prompts, reduce cognitive cost. The cognitive cost of figuring out how to change or create an example can be equally bad. When coupled with the specific demands of creative applications—that we want some iteration and some surprise—achieving a ‘sweet-spot’ through algorithmic means alone seems implausible. Put another way, simply changing the algorithm can’t solve our problem if the interface costs are high or the user’s requirements for a creative tool are unmet.

### 3.3. Input Design

To achieve our requirements, we argue that interaction is a critical factor. We focus on possible input approaches. These will naturally range based on the type of input directness.

#### 3.3.1. Direct Inputs: A Small Space of Design

Direct inputs are usually made in the same medium as the artifact. For example, we might use low-fidelity sketches on the drawing surface when the target artifact is visual. These sketches will then be transferred to high-fidelity images on (essentially) the same surface/encoding [9]. In other cases, the algorithm may simply append elements to the sketch [18]. This type of input serves as the ‘material’ for the generation—where the transfer is applied or what the generative algorithms build upon.

While direct inputs may depend on the application domain, their specific type may be largely constrained to a small design space. This is largely because the representation depends on the target artifact’s medium (e.g., a drawing canvas). Additionally, the interactions are constrained by the underlying algorithm. For example, we may train an algorithm to produce a photo-realistic image given a low-resolution sketch. Such datasets are more readily available and easier to produce. The user-facing input modality and form are thus constrained to something that looks like the training data. Finally, direct inputs create a set of expectations for the end-user that need to be maintained in the interactive controls. Because of these constraints, which may limit our design space options, we move to consider indirect inputs.

#### 3.3.2. Indirect Inputs: Approaches and Their Limitations

Indirect inputs serve as instructions for both *what* and *how* to generate. Unlike direct inputs, they are not dependent on the artifact’s medium. One can work in abstract spaces or through abstract representations. Thus, there is often more freedom to design with indirect inputs. The consequence of freeing ourselves from the constraints of the domain also enables us to consider additional algorithm types. This flexibility further affords a better ability to match the end user’s high-level intentions rather than forcing them to work within their algorithm and interface constraints. However, this is not to say that all indirect inputs are good ones. A novel indirect interaction might be further from the target artifact’s modality and thus might be harder to master when the mapping is complex. A poorly designed indirect interaction can also increase cognitive costs and reduce the ability to iterate. All together, indirect inputs open up a vast space of possibilities but introduce various pitfalls.

To better understand which aspects may help or hinder, we focus on three types of inputs: traditional input widgets, natural language prompts, and gestural inputs. Traditional input widgets, such as sliders for numerical inputs, represent the simplest option. If we are able to use these in the interface, it often means that we can directly map the user’s expectation to the actual behavior of the system. In reality, this depends on how the end-user understands the construct represented with the widget. If the label on the slider is ambiguous (e.g., this will control the ‘brightness’ of the text) or novel (e.g., this will control the ‘certitudeness’ of the text), the user may struggle with the control. Clearly, increased experience with the tool will improve as the user calibrates to the system. More critical, however, are situations where a user only has a rough idea of what they want to generate. Here, a standard input widget may be insufficient. As critically, the algorithms themselves may not deliver on the precision of the input. Thus, the interface is over-promising. The ease of iteration with input widgets largely depends on the complexity of the interface. A single slider is simple, but many controls and interactions will naturally become more challenging.

Recent generative co-creation systems have enabled indirect natural language prompts as input. For example, natural language prompts can steer vision-language models to generate visual images [22, 23]. As these approaches can be used with imprecision or ambiguity, they are useful for giving high-level specifications on generations. However, they would be difficult to iterate with due to the vast space of inputs.

Surprisingly, few human-AI co-creation tools have used gestural or sketch interactions for indirect control. We argue that this is a missed opportunity as there are several benefits to this approach. In the next section, we expand on this possibility and why it may be appropriate.

## 4. Gestural Indirect Inputs for Generative Co-Creation

We propose that gestural or sketch-based interfaces for indirect specification satisfy our requirements for co-generation tools. At the very least, this approach may complement other input controls. First, simple gestural interactions (e.g., producing a rough sketch) are easy to iterate with. This characteristic can complement more effortful controls such as prompts or examples. Moreover, the multi-dimensional characteristics of sketches and gestures can reduce the effort to interact with multiple attributes simultaneously. For example, 2D sketching coupled with pressure and speed recognition can be used to encode multiple parameters simultaneously. This flexibility also means that we can work in abstract visual

encodings. Finally, as we have argued before, gestural inputs convey the sense of being rough and flexible. This strongly aligns with the non-determinism of the algorithm and the ambiguity of the user’s intent. Moreover, it can convey the ‘unfinished’ nature of the generative process.

As a demonstration of the feasibility of this approach we describe our system TaleBrush [10] (Figure 1). TaleBrush is a human-AI story co-creation tool that generates story sentences according to the specifications of the protagonist’s fortune. For example, if we were describing Cinderella’s fortune, we might say that: her fortune started low (with her stepmother and sisters), improved greatly as she went to the ball, collapsed as she was forced to flee, and then improved again when the prince found her.

TaleBrush allows the user first to input a portion of the story (direct input) in the text box (Figure 1A1). Then, they can sketch out the protagonist’s fortune in a 2-dimensional line sketch as in Figure 1A2. This is roughly a standard time series with the  $x$  and  $y$  axes standing for sequence position and fortune levels, respectively. Using this sketched line (which is actually represented as a sketch rendering), TaleBrush will generate a story (Figure 1B1). Because the underlying algorithm is ambiguous and may not precisely match the desired fortune sketch, the best matching generation is also displayed in the visualization (Figure 1B2). Technically, this sketch-based control is powered by steering a big pretrained language model with a smaller language model that receives sketch position as control code. More details can be found in the Chung et al. [10]

With TaleBrush, the benefits of gestural inputs hold. First, iteration on the generation is easy. The user only needs to redraw parts of the sketch. Additionally, a single drawn line expresses two dimensions simultaneously: (1) where in the story and (2) at what fortune level should the sentence be. Notably, the first (position) is a direct input, whereas the fortune level represents an indirect one. In reality, we also use the speed at which the sketched line is drawn to indicate how much ambiguity the user will tolerate in the generated result. This is visually represented in the thickness of the line. A thinner line indicates the user wants a better match. Internally, this is implemented by regenerating the sentences multiple times and finding the one that best matches that desired fortune level. This visualized boundary further emphasizes the ambiguity and non-determinism of the algorithm. Note that a ‘sketch’ does not necessarily mean a ‘sketchy appearance.’ However, we have opted to use this rendering aesthetic to further lower the user’s expectations that the algorithm should be precise [7, 6].

## 4.1. Design Approaches for Gestural Indirect Inputs

Implementing TaleBrush has given us some insight about what may work well (and poorly) for indirect gestural inputs.

### 4.1.1. Ease of Iteration on Input

**Combine direct and indirect input if possible** For some generation tools, indirect inputs may be sufficient. For example, if the tool generates any character biographies based on the good-evil nature of the character, then it might not require direct inputs. However, as with TaleBrush, certain tasks require control with direct input. The user needs to be able to indicate, “where the generation should be done” (e.g., where in the story a certain fortune level should exist or what does the start of the story look like?). In some cases, as we did with TaleBrush, the indirect and direct controls can be combined into a single gestural sketch. That is, with a single brushstroke, the sequential position ( $x$  position—the ‘where’) and the level of the protagonist’s fortune ( $y$  position—the ‘how’) are both specified.

**Complement hard-to-iterate control inputs (language, examples)** Spatial positions by themselves do not necessarily convey meaning. They are meaningful when combined with semantic structures that can be put on a continuous scale. For example, TaleBrush takes a restricted set of numerical semantics: whether the character’s fortune is good or ill. However, this design can be extended to receive qualitative inputs as the endpoints of the axes. For example, the user can give natural language prompts or examples on each end and explore the confined space with gestural inputs. This complements the limitations and features of different input approaches. Language prompts and examples lack the ease of iteration, which is the strength of gestural inputs. On the other hand, gestural inputs lack semantics, which language prompts and examples can convey.

### 4.1.2. Matching Input Precision with Algorithmic Uncertainty

**Match the algorithmic precision with the input precision** To have better expectations of how the algorithm will behave, the user should ideally be aware of the precision of the algorithm. Gestural inputs can be designed to convey this information. For example, in TaleBrush, this level of precision is reflected in the width of the sketched line. This was designed to match the median error from the test dataset we used during development. Thus, the interaction and representation can be used in ways that reduce ambiguity and help to match (and manage) expectations.

**Controlling the precision** When using gestural inputs, the user’s intention regarding the precision might vary. For example, in TaleBrush, the user might have wanted the algorithm to follow the generation more tightly when they draw the line with more care. The system can be designed to leverage other input dimensions to control the precision to reflect these intentions better. In TaleBrush, the sketching speed was used to decide how tightly generation should be done. That is, if the user drew slowly, we assumed this indicated that they wanted a better fit (represented as a thinner error envelope). In this way, gestural interactions and representations can be used to align input precision with system capabilities.

## 5. Conclusion

In this position paper, we have explored where generative algorithms sit in the overall design space of co-creative tools. We have further isolated those properties that are desirable and potentially required for supporting human-AI co-creation. Our focus was on how inputs (both the ‘what’ and the ‘how’) can interact with underlying algorithms. Our focus on enabling iteration and managing expectations allowed us to consider the pros and cons of different input types. Ultimately, we argued that gestural and sketch-based interactions would work well for the control of generative algorithms. We showcased the benefits of this approach with TaleBrush. We believe that there are significant possibilities opened up by using abstract visual representations when coupled with novel interaction types.

## Acknowledgments

We thank our reviewers for providing helpful feedback on this work.

## References

- [1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, volume 27, Curran Associates, Inc., 2014. URL: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>.
- [2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh,



- D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin (Eds.), *Advances in Neural Information Processing Systems*, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>.
- [3] Y. Wu, E. Manilow, Y. Deng, R. Swavely, K. Kastner, T. Cooijmans, A. Courville, C.-Z. A. Huang, J. Engel, Midi-ddsp: Detailed control of musical performance via hierarchical modeling, 2021. arXiv:2112.09312.
- [4] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, I. Sutskever, Zero-shot text-to-image generation, in: M. Meila, T. Zhang (Eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 8821–8831. URL: <https://proceedings.mlr.press/v139/ramesh21a.html>.
- [5] L. A. Gatys, A. S. Ecker, M. Bethge, Image style transfer using convolutional neural networks, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, USA, 2016, pp. 2414–2423. doi:10.1109/CVPR.2016.265.
- [6] M. D. Gross, E. Y. Do, Ambiguous intentions: A paper-like interface for creative design, in: *ACM Symposium on User Interface Software and Technology*, ACM, 1996, pp. 183–192.
- [7] J. Landay, B. Myers, Sketching interfaces: toward more human interface design, *Computer* 34 (2001) 56–64. doi:10.1109/2.910894.
- [8] M. Eitz, J. Hays, M. Alexa, How do humans sketch objects?, *ACM Trans. Graph. (Proc. SIGGRAPH)* 31 (2012) 44:1–44:10.
- [9] S.-Y. Chen, W. Su, L. Gao, S. Xia, H. Fu, DeepFaceDrawing: Deep generation of face images from sketches, *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2020)* 39 (2020) 72:1–72:16.
- [10] J. J. Y. Chung, W. Kim, K. M. Yoo, H. Lee, E. Adar, M. Chang, TaleBrush: Sketching Stories with Generative Pretrained Language Models, *Association for Computing Machinery*, New York, NY, USA, 2022.
- [11] J. J. Y. Chung, S. He, E. Adar, The intersection of users, roles, interactions, and technologies in creativity support tools, in: *Conference on Designing Interactive Systems*, ACM, 2021, pp. 1817–1833.
- [12] B. Syed, G. Verma, B. V. Srinivasan, A. Nataraajan, V. Varma, Adapting language models for non-parallel author-stylized rewriting, 2020. arXiv:1909.09962.
- [13] C. J. Steinmetz, J. D. Reiss, Steerable discovery of neural audio effects, 2021. arXiv:2112.02926.
- [14] R. Louie, A. Coenen, C. Z. Huang, M. Terry, C. J. Cai, Novice-ai music co-creation via ai-steering tools for deep generative models, in: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1–13. URL: <https://doi.org/10.1145/3313831.3376739>. doi:10.1145/3313831.3376739.
- [15] C.-J. Chang, C.-Y. Lee, Y.-H. Yang, Variable-length music score infilling via xlnet and musically specialized positional encoding, 2021. arXiv:2108.05064.
- [16] P. Ammanabrolu, W. Cheung, W. Broniec, M. O. Riedl, Automated storytelling via causal, common-sense plot ordering, *CoRR abs/2009.00829* (2020). URL: <https://arxiv.org/abs/2009.00829>.
- [17] A. Fan, M. Lewis, Y. Dauphin, Strategies for structuring story generation, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Florence, Italy, 2019, pp. 2650–2660. URL: <https://aclanthology.org/P19-1254>. doi:10.18653/v1/P19-1254.
- [18] J. E. Fan, M. Dinculescu, D. Ha, Collabdraw: An environment for collaborative sketching with an artificial agent, in: *Proceedings of the 2019 on Creativity and Cognition, C&C '19*, Association for Computing Machinery, New York, NY, USA, 2019, p. 556–561. URL: <https://doi.org/10.1145/3325480.3326578>. doi:10.1145/3325480.3326578.
- [19] Y. Lin, J. Guo, Y. Chen, C. Yao, F. Ying, It is your turn: Collaborative ideation with a co-creative robot through sketch, in: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1–14. URL: <https://doi.org/10.1145/3313831.3376258>. doi:10.1145/3313831.3376258.
- [20] C. Yan, J. J. Y. Chung, K. Yoon, Y. Gingold, E. Adar, S. R. Hong, FlatMagic: Improving Flat Colorization through AI-driven Design for DigitalComic Professionals, *Association for Computing Machinery*, New York, NY, USA, 2022.
- [21] L. Sheng, Z. Lin, J. Shao, X. Wang, Avatar-net: Multi-scale zero-shot style transfer by feature decoration, in: *Computer Vision and Pattern Recognition (CVPR)*, 2018 IEEE Conference on, 2018, pp. 1–9.
- [22] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, M. Chen, Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2021. arXiv:2112.10741.
- [23] F. Huang, J. F. Canny, Sketchforme: Composing

- sketched scenes from text descriptions for interactive applications, in: Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 209–220. URL: <https://doi.org/10.1145/3332165.3347878>. doi:10.1145/3332165.3347878.
- [24] E. Frid, C. Gomes, Z. Jin, Music creation by example, in: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1–13. URL: <https://doi.org/10.1145/3313831.3376514>. doi:10.1145/3313831.3376514.
- [25] Y. Kim, S.-G. An, J. H. Lee, S.-H. Bae, Agile 3D Sketching with Air Scaffolding, Association for Computing Machinery, New York, NY, USA, 2018, p. 1–12. URL: <https://doi.org/10.1145/3173574.3173812>.
- [26] I. S. MacKenzie, Fitts' law as a research and design tool in human-computer interaction, *Hum.-Comput. Interact.* 7 (1992) 91–139. URL: [https://doi.org/10.1207/s15327051hci0701\\_3](https://doi.org/10.1207/s15327051hci0701_3). doi:10.1207/s15327051hci0701\_3.
- [27] T. M. Amabile, The social psychology of creativity: A componential conceptualization., *Journal of personality and social psychology* 45 (1983) 357.
- [28] T. M. Amabile, Componential theory of creativity (2012).