Keyframer: Empowering Animation Design using Large Language Models

TIFFANY TSENG, Apple, USA

RUIJIA CHENG, Apple, USA
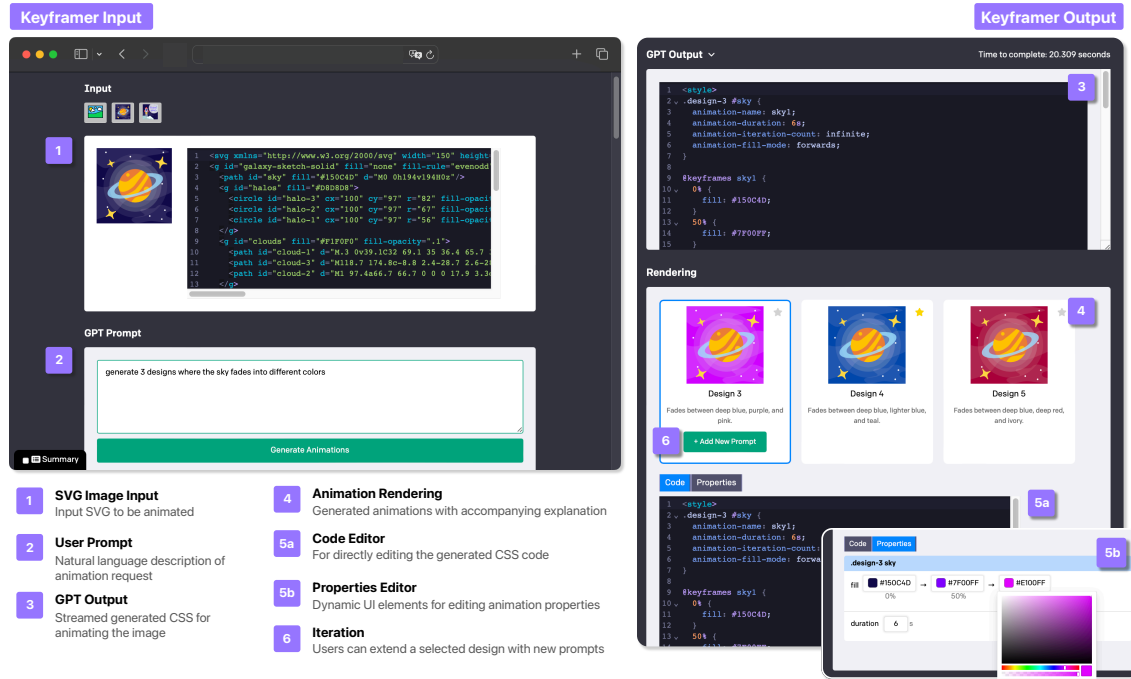
JEFFREY NICHOLS, Apple, USA

Fig. 1. Keyframer is an LLM-powered animation prototyping tool that can generate animations from static images (SVGs). Users can iterate on their design by adding prompts and editing LLM-generated CSS animation code or properties. Additionally, users can request design variants to support their ideation and exploration.

Large language models (LLMs) have the potential to impact a wide range of creative domains, but the application of LLMs to animation is underexplored and presents novel challenges such as how users might effectively describe motion in natural language. In this paper, we present Keyframer, a design tool for animating static images (SVGs) with natural language. Informed by interviews with professional animation designers and engineers, Keyframer supports exploration and refinement of animations through the combination of prompting and direct editing of generated output. The system also enables users to request design variants, supporting

Authors' addresses: Tiffany Tseng, tiffanytseng@apple.com, Apple, USA; Ruijia Cheng, rcheng23@apple.com, Apple, USA; Jeffrey Nichols, jwnichols@apple.com, Apple, USA.

comparison and ideation. Through a user study with 13 participants, we contribute a characterization of user prompting strategies, including a taxonomy of semantic prompt types for describing motion and a 'decomposed' prompting style where users continually adapt their goals in response to generated output. We share how direct editing *along with* prompting enables iteration beyond one-shot prompting interfaces common in generative tools today. Through this work, we propose how LLMs might empower a range of audiences to engage with animation creation.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**.

Additional Key Words and Phrases: generative ai, animations, design software, large language models

1 INTRODUCTION

There is great potential for large languages models (LLMs) to influence and support creative work across the design process, from concept development to refinement to execution. Over the past year alone, researchers have developed LLM-powered design tools in a range of domains such as visual design [12, 13, 44], creative writing [26, 46], and 3D modeling [27, 37]. The use of descriptive natural language prompts has the potential to reduce the technical expertise needed to create design artifacts and provide novices with educational opportunities to improve their design skills [32]. Yet, while prompting strategies for text-to-image generators have been well studied in recent work [12, 35, 36, 44], the transferability of these strategies to new domains is uncertain, and research is needed to understand domain-specific needs.

In this paper, we apply LLMs to the less explored domain of animation design. Using LLMs in this domain may be especially fruitful as creating animations requires many different types of technical skills, from applying motion design principles for compelling visual communication, to executing and implementing animations in code in production (such as for advertising, games, and user interfaces); as a result, animation work often involves varied stakeholders like motion designers, technical artists, and software engineers. While one-shot prompting interfaces are common in commercial text-to-image systems like Dall·E[1] and Midjourney[2], we argue that animations require a more complex set of user considerations, such as timing and coordination, that are difficult to fully specify in a single prompt—thus, alternative approaches that enable users to iteratively construct and refine generated designs may be needed especially for animations.

We combined emerging design principles for language-based prompting of design artifacts with code-generation capabilities of LLMs to build a new AI-powered animation tool called Keyframer. With Keyframer, users can create animated illustrations from static 2D images via natural language prompting. Using GPT-4 [3], Keyframer generates CSS animation code to animate an input Scalable Vector Graphic (SVG). To support user refinement of generated designs, Keyframer has multiple editor types for users to directly edit generated animations. Additionally, users can iterate on their designs through sequential prompting and request variants from the LLM to ideate on new design directions. Through these features, Keyframer accommodates users exploring and adapting their design goals as they iteratively construct an animation through combined prompting and editing actions.

---

[1]https://openai.com/dall-e-3
[2]https://www.midjourney.com/
[3]https://openai.com/research/gpt-4

To inform the development of Keyframer, we first conducted formative interviews with 9 professional designers and engineers who work on animations, such as character design and UI design of mobile applications. Through these interviews, we aimed to answer the research question,

- **RQ1**: What painpoints exist for motion designers, and what ideas do they have for how AI might assist with these processes?

We used insights from these interviews to shape the design of Keyframer and evaluated the efficacy of Keyframer for supporting animation design through an exploratory user study with 13 users with a range of animation and programming experience. During this study, participants used Keyframer to animate two provided illustrations over the course of a 90 minute session, with agency to decide how to prompt the system and refine their designs with Keyframer. Our evaluation of Keyframer examined the following two research questions:

- **RQ2**: What design strategies do users take to prompt LLMs for animations using natural language?
- **RQ3**: How does Keyframer support iteration in animation design?

Through our analysis of user prompting strategies, we contribute a taxonomy of semantic prompting styles users employed to describe motion with natural language, showing how Keyframer empowered both novices and experts to create animations while focusing on high level design goals. We describe how users predominantly built up their designs through a sequential prompting style we call 'decomposed' prompting. With decomposed prompting, users iteratively develop their ideas through incremental prompts that animate individual elements within a scene. This workflow can allow users to adapt their goals continually in response to generated output. By providing multiple pathways for iterating on designs (through a prompting interface and editors for modify generated animation properties), Keyframer allows users to maintain creative control throughout their design process. Further, we share how unexpected LLM output can be beneficial both in exploration and refinement stages of the design process. Our work makes the following contributions:

(1) Keyframer, an LLM-powered application for generating animations from static images.
(2) A taxonomy of prompting strategies users employed to describe animations with natural language.
(3) Insights into supporting design iteration through combined prompting and editing interfaces.

2 RELATED WORK

Our work draws from related efforts considering how generative AI might support and create new opportunities for design, along with prior work underscoring the value of iteration in design processes.

2.1 Generative AI in Design

The introduction of LLMs like ChatGPT[4] have facilitated an unprecedented rise in commercial and research efforts to explore their application to design fields. Muller et al. argue that generative AI introduces new challenges to the field of HCI "due to the serendipitous and uncertain nature of the design space," with numerous open questions about how to design user experiences that can effectively facilitate creative work [40]. The use of natural language input presents opportunities to lower the barrier to entry for engaging in creative practice, while the interface paradigms of co-creating with AI are being proposed in domains such as graphic design [12, 25, 36, 44], software development [6, 49], creative computing [3, 32], UI design [43], writing [26, 46], 3D CAD [27, 37], and music [38].

---

[4]https://openai.com/chatgpt

A common challenge in natural-language-based generative AI tools is developing effective prompting strategies for steering generated output. Today, prompting tends to require much trial and error, with a lack of meaningful controls for end users [36]. Current efforts to define prompting strategies largely center on text-to-image generators that tend to employ one-shot approaches, where the user's only lever to edit the image is via text prompting (though researchers are also exploring multi-modal input with images and painting interfaces for editing selected regions [4, 17, 24, 36, 45]). Several prompt taxonomies have been proposed [42, 44], with generative art communities using modifiers specifying artistic style (e.g., 'Cubism') and quality (e.g., 'award winning'), along with keywords to spur surprising output (what Oppenlaender refers to as 'magic terms' [42]). Similarly, Chiou et al distinguish between 'operational' keywords that specify concrete reference terms and 'conceptual' keywords using abstract modifiers that are more likely to lead to unexpected results [12]. Emerging research on LLM-generated scalable vector graphics (SVGs) suggests a promising path towards recombining high-quality, designer-created assets for consistent visual style [39].

Early work on applying LLMs to create animated content has appeared in the form of creative coding support for p5js[5][3] and video generation with tools like RunwayML[6]. However, research exploring generative AI for creating animations from *existing* image assets has been underexplored; this process is most closely aligned with current professional practices, where animations are created from assets created by visual designers. Research is needed to determine whether animations may require alternative prompting strategies compared to text-to-image generation, or even to determine how effective LLMs may be at generating animations altogether.

To mitigate the trial-and-error workflow common to prompting generative AI tools, several prototype tools have sought to visualize related keywords to help users better explore their options [7, 25, 37, 46]. Commercial image generators like Dall-E [7] and Midjourney[8] often present users with polished visual output with only a handful of keywords; yet, this may lead users to narrow in on one design direction before fully exploring alternatives [46]. Presenting users with multiple options has been found to support users overcoming creative blocks in their process [3, 13], spur new ideas [46, 47], and verify the quality of generated output [6]. In the context of code generation, other related work has similarly distinguished between two use cases for LLMs: supporting 'exploration,' when a user is not sure of what they want to create yet, and 'acceleration,' where a task is well defined and the user wants the LLM to help them get to a solution faster [3, 6, 32]. In this work, we are interested in how prompting for animations might compare to the existing literature on prompting in areas like image generation and prose. Further, we consider how the integration of design variants might help support ideation processes and ultimately promote design iteration.

2.2   Iteration in Design Prototyping

Iteration is a fundamental and crucial aspect of the design process. Typically consisting of repeatedly exploring new ideas and refining designs, iteration can foster alternative perspectives [18], reflection [2, 28], and stakeholder feedback [18, 51], aiding designers in identifying challenges and uncovering new directions [1, 10, 11, 23, 33]. Iteration is essential to complex designs where exploring different alternatives can reveal potential issues and opportunities. Animation design exemplifies a domain that greatly benefits from iteration due to its multiple stages and stakeholders. For example, film animations involves the integration of story, script, storyboarding, media design, and visual narratives [48], while animations for games can involve character rigging, lighting, and special effects [30]. Creating animations often requires collaboration among diverse stakeholders and iteration to refine ideas and ensure alignment across all

---

elements of the project. Our work with Keyframer focuses on computer-based animation, specifically the creation of web-based animations, which often involves graphic designers and front-end engineers working together to create production-ready animations such as animated transitions, loading animations, and data visualizations [22, 29].

Iteration involves both *exploration* and *refinement*, as illustrated by the classic "Double Diamond" model in design [14]. Exploration typically entails generating ideas and multiple variants of a design solution, where designers experiment with different styles and design languages. Instead of committing to a single design concept at the early stage of the design process, creating design variants can help designers avoid the pitfall of fixation [31], discover potentially valuable directions [15, 16], and generate a more diverse array of ideas [9]. An effective practice of creating design variants is *parallel prototyping*, which involves creating multiple design variants in parallel before making further improvements on any of the variants [11]. By generating and comparing varied prototypes in parallel, designers were able to create higher quality and more diverse design solutions and feel more confident in the design process than working on single designs [19, 20, 41]. Refinement is where designers go deep on improving a single design to match their design goals [5]. It has been proven in multiple studies that rapid refinement on existing designs can enhance design outcomes [21]. In this paper, we explore how the emerging technology of large language models can assist exploration and refinement in animation design.

3 FORMATIVE STUDY

We conducted a formative interview study to answer our research question

- **RQ1**: What painpoints exist for motion designers, and what ideas do they have for how AI might assist with these processes?

We recruited animation designers, developers, and prototypers via Slack channels dedicated to animation and front-end engineering at a large technology company. We invited participants who have 'experience designing and/or developing animations as part of your work' to fill out a short survey with background questions such as their job title and what tools they use to create their animations. Selected participants received a $12 meal voucher for their time.

During our 45-minute, semi-structured, video conference-based interviews, we asked participants about the types of animations they have worked on in the past and then asked each participant to walk us through a specific example where they created an animation using screen sharing. We then asked about their perspective of the role of generative AI in design tools. Finally, we probed the participants with a brief demo of an early prototype of Keyframer that takes in natural language prompts to generate animation for a simple SVG. We used this demo as a probe to ask for feedback about potential benefits or limitations of LLM-based animation tools.

To analyze our interviews, two researchers edited and reviewed the transcriptions automatically generated by the video conferencing tool and applied thematic analysis [8] to inductively identify a set of painpoints and opportunities across different stages of the animation creation process. We also categorized different opportunities for generative AI support shared by participants when they were describing their experience with existing generative AI design tools and reacting to the concept of LLM-based animation tools that we showed in the demo. We conducted the interviews in sets of three, analyzing and discussing emergent themes in weekly meetings. We continued the interviews until we reached saturation, with no new emergent themes, which happened after 9 total interviews.

3.1 Formative Interview Results

From a total of 27 survey respondents, we selected and interviewed 9 participants (2 female, 7 male) representing a range of job titles such as designer, technical R&D artist, and front-end developer. Their animation work includes user interface animations, advertising, instructional documentation, character animation, and interactive data visualization with between 3–22 years of professional experience (details about our participants can be found in Table A.1). [9] While a full discussion of challenges identified throughout the entire animation lifecycle is out of scope for this paper, we highlight a few opportunities especially pertinent to the role generative AI might play in animation design tooling.

Challenge: Translating design to engineering implementation. Transitioning from design to engineering implementation (commonly referred to as "handoff") is time-consuming and can ultimately take away from other stages of the animation design process. Translation happens both because design is a collaborative process between designers and engineers, and because of differences in performance of animations across tools (e.g., production designs may appear differently than design mockups due to latency or rendering discrepancies): "In animation software, I can do anything, but real product might have limitations" (FP3). Handoff often manifests in the creation of design specs or the development of custom tooling for translating animation properties in software like After Effects into production code (Swift, JavaScript, etc). Translation work, especially when it can involve several rounds of feedback and adjustment between prototyping and production, takes time away from design iteration: "If I do one animation and try it and then it takes me 3 hours till I get to see it on the product, I don't get many cycles [of iteration] and then the animation quality drops significantly" (FP3). For these reasons, there was a general preference to get as close as possible to the final format the animation will ultimately appear in, which for the creators we interviewed, was either implemented in code on a webpage or in a mobile application. Participants like FP1 imagined how generative tools that generate both animation and the underlining code could mitigate this challenge: "It would help because the code is there, so it will also help an engineer understand [the design]... it's a communication tool for both [engineering and design]."

Opportunity: Using AI to help generate an initial starting point. Animation creators saw potential in using AI to generate an initial starting point that they could then refine. Prototype engineer FP6 described his enthusiasm for tools like Midjourney: "It's a great starting point for a lot of things, and it helps unblock that creative process a lot of the times of, like, hey, where do I start?" Front-end developers anticipated the usefulness of AI for writing code: "Eventually, it [AI design tools] is going to be very helpful — not to write all the code, but to help in certain parts of the code" (FP2).

Even with potential benefits of using AI, designers emphasized the importance of ultimately customizing the AI-generated output, rather than relying on the AI to produce final designs, as designer FP6 described: "If an [AI] could help alleviate and create some more complex animations, or at least start it for me so I don't have to write every single line of code, that would save me so much time... I'm not writing the actual base code, I'm just truly putting that human touch to it."

Opportunity: Supporting design iteration. Another area animation creators saw potential for generative AI was in iterating on animations, particularly by helping them explore multiple options. Some participants like FP7 imagined LLMs serving more to refine designs rather than proposing completely new designs, suggesting using AI to apply design standards like accessible complementary colors or a dark mode from a light mode design. FP5 shared how it was rare for animators to create a completely new timing curve, so applying animation properties from references examples would be valuable. Designer FP1 shared how AI can be especially helpful in generating design variants quickly from an initial design: "I feel like this can be super helpful, especially for those things that need to be tweaked."

---

[9]We use FP to refer to formative interview participants and EP to refer to evaluative study participants in Section 6

Limitations: Lack of Creative Control. A general concern about using generative AI was the risk of losing creative control, both in terms of reduced editing functionality along with a loss in creative satisfaction. For example, FP4 shared, "[When] AI is being used to create the entire work, it doesn't scratch the itch that I think creative people have when they have that desire to create something. [With Midjourney] I feel like I'm pushing a button and something pops out." Participant FP9 also shared that, "It's the granularity of customization that I really struggle with. When you're drawing, you control every stroke, and the nuance of it matters, and you control so little in AI by definition." Designers also feared having decreased understanding of how to edit generated designs. Developer FP8 described this gap by stating that, "If you involve yourself deeply in all the details of the work, you have more freedom because you know exactly where things could break... [AI could] obfuscate some of these layers. And that could make it harder, if something goes wrong, to know where to look because that code was essentially not written by you."

3.2 Design Goals for AI-Supported Animation Tooling

Overall, our interviews revealed that animation creators were optimistic about using generative AI tools for quickly prototyping animation concepts. Following the challenges and opportunities we identified from the formative interviews, we developed the following design goals (DG) for an AI-supported animation tool:

DG1: Support exploration for animations. Animation creators imagined leveraging the power of AI to generate initial designs and to rapidly explore alternatives. We made these needs part of our system by leveraging LLMs to generate initial starting points users can iterate on. We also took advantage of the flexibility of LLMs to allow users to generate multiple versions of a design at once to aid comparison and selection.

DG2: Enable granular controls for editing animations. Our formative study revealed that animators perceive a lack of creative control in current text-based image generation AI systems. We also found that animation creators iteratively refine their designs when adapting to evolving design goals. As many animations are implemented in code, we anticipated opportunities to enable fine-grained refinement through direct or GUI-based edits to LLM-generated animation code.

DG3: Empower non-experts to work with animation code. We aimed to reduce the translation burden of turning designs mockups into implemented animations by leveraging the code generation power of LLMs, which we thought could be especially helpful for designers less familiar with code. Additionally, using natural language prompts to generate animation might open up opportunities for non-experts to get started with animation, which led us to expand our audience for testing Keyframer to both professionals and beginners.

4 KEYFRAMER SYSTEM

Keyframer is an LLM-powered application we developed for creating animations from static images. Leveraging the code-generation capabilities of LLMs, along with the semantic structure of Static Vector Graphics (SVGs), Keyframer generates CSS for animating SVGs based on user-supplied natural language prompts. Keyframer was designed to support both 1) exploration by enabling users to request, compare, and explore design variants; and 2) refinement, as users are able to iterate on their designs either through reprompting or by editing the LLM-generated animations directly through several designed editor modes. Keyframer is implemented using Open AI's Chat Completion API [10], utilizing GPT-4 as its base model. In this section, we describe the design of the Keyframer interface and how its features align with our Design Goals (DGs) described in Section 3.2.

---

[10]https://platform.openai.com/docs/guides/text-generation/chat-completions-api
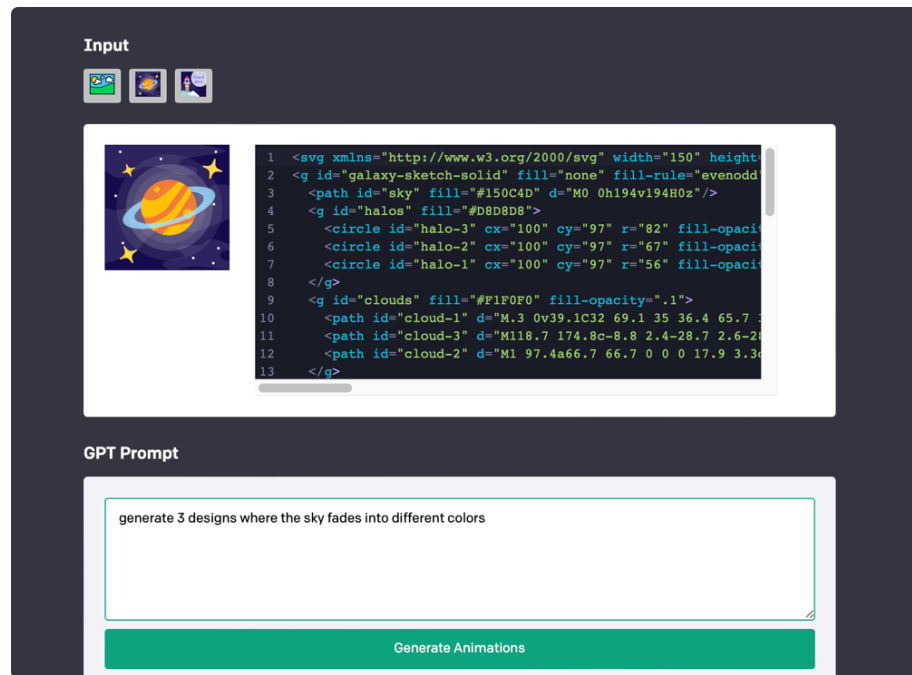
Fig. 2. Image input field for adding SVG code and previewing image; GPT Prompt section for entering a natural language prompt.

Input. Our system offers an input area where users can paste the code for an SVG image they want to animate. SVG is a standard and popular image format used commonly in illustration due to its scalability and compatibility on many platforms; its use for animations is also established in user interface design and advertising [22, 29]. In Keyframer, a rendering of the SVG is displayed alongside the code editor so users can preview the visual design of their image as shown in Figure 2. Because SVGs are XML-based images, the LLM is able to use descriptions of the scene embedded in the code, such as object identifiers. For example, in Figure 2, the SVG code for the Saturn illustration contains identifiers such as sky, halos, and clouds. Object identifiers are commonly defined by designers by naming layers when creating graphics in applications like Illustrator or Sketch.

GPT Prompt. Following DG3, our system allows users to create animations by entering a natural language prompt into a text input field. As an open-ended text field, users can make a request for a single design ("Make the planet spin") or for multiple design variants ("Create 3 designs where the stars twinkle", DG1). The user clicks the *Generate Animations* button to begin their request.

Before passing a user request to GPT, we supplement their prompt with the full raw SVG XML and specify the format of the LLM response. For example, we request that the response consists solely of CSS, as generating both SVG and CSS led to longer response times and higher risk of exceeding token limitations. Further, we ask for each CSS snippet to be accompanied by a descriptive explanation that can help users inspect what the LLM generated design is intended to look like. The full details of our prompt can be found in Appendix A.5.

GPT Output. Once a prompt request has started, we stream the response from GPT, which consists of one or more CSS snippets as shown in Figure 3. The streaming response provides feedback to the user about whether the request is

Fig. 3. GPT Output section where the streamed response from the LLM is displayed in advance of rendering each CSS snippet.

still in progress, while also allowing users to preview the LLM's response. Following DG3, we support users unfamiliar with CSS to interpret animation code by prompting the LLM to generates descriptive names for keyframes. Generated names provide a hint for the design that will be rendered, such as 'flame-flicker' or 'flash-sparkles'. Each time we detect a complete snippet, we render the animation inline as described in the next section.

Rendering. The Rendering section consists of, 1) a visual rendering of each animation along with 1-sentence explanation generated by the LLM, and 2) a series of editors for modifying the design.

Once a code snippet is detected (e.g., the delimiter between code snippets is found in the GPT-output), we inject the CSS snippet along with the original SVG code onto the page to display the design. As we render multiple designs with the same underlying SVG on a single page, we have the LLM provide a unique class for each CSS snippet (e.g., `.design-1`, `.design-2`, etc.). Visual renderings appear side by side to invite comparison and aid users in selecting the design that best matches their vision.

To provide granular controls for editing the generated animations (DG2), we offer two kinds of editors in Keyframer. When a user clicks on a given design, the editors below display the CSS snippet in two formats: a **Code Editor** where the CSS can be edited directly, and a **Properties Editor**, a dynamically created UI-layer for editing CSS properties. Following DG3, the Properties Editor is designed to be approachable to users less familiar or comfortable with editing CSS syntax and is modeled after UI elements one might see in graphics editors like Illustrator.

The **Code Editor** is implemented with CodeMirror and has syntax highlighting and autocompletion for CSS. We re-inject the revised CSS onto the page on every keystroke so that users can preview their changes immediately. The **Properties Editor** provides property-specific UI for editing the code (DG3); for example, for editing colors, we provide a color picker and for editing timing function curves, we provide a dropdown of defaults (like *linear* and *ease-in*), along with a bezier editor for entering custom timing curves. Animation properties and keyframes are grouped together by identifier to improve readability. Figure 5 shows how an example code snippet appears in both the code editor and the properties editor. Edits in either mode are immediately reflected in the companion mode to support direct manipulation [34].

Iteration. To support user exploration in the animation creation process (DG1), we offer a feature to allow users to iteratively build on a generated animation using prompts. Underneath every generated design is a button *+ Add New Prompt*; clicking this button opens up a new form at the bottom of the page for a user to extend their design with a new

Fig. 4. Rendering section for viewing generated designs side by side and editing response code in the Code or Properties editors.

Code Editor

Properties Editor

Fig. 5. The Code and Properties Editors enable users to edit generated animation code directly. The editors are bi-directional so that any edits in one editor are reflected in the corresponding editor. The Properties editor adds a UI layer for editing animation properties without having to understand CSS syntax.

prompt. In this way, users can continually add additional prompts in sequence to iterate on their animation. When a user adds a new iteration, our prompt requests the LLM to extend existing CSS code as outlined in A.5.

Saved Designs Sidebar & Summary. We enable users to star designs to favorite them and add them to a sidebar, as displayed on the right-hand side of Figure 6. Clicking on a given design in the sidebar will scroll to where the design

Fig. 6. A summary view that hides all text editors and displays the user prompt and generated designs at each iteration. The Saved Designs sidebar (shown on the right) appears in either the Summary or regular view and shows all favorited designs. Clicking on a favorited design scrolls to the iteration in which the design was generated.
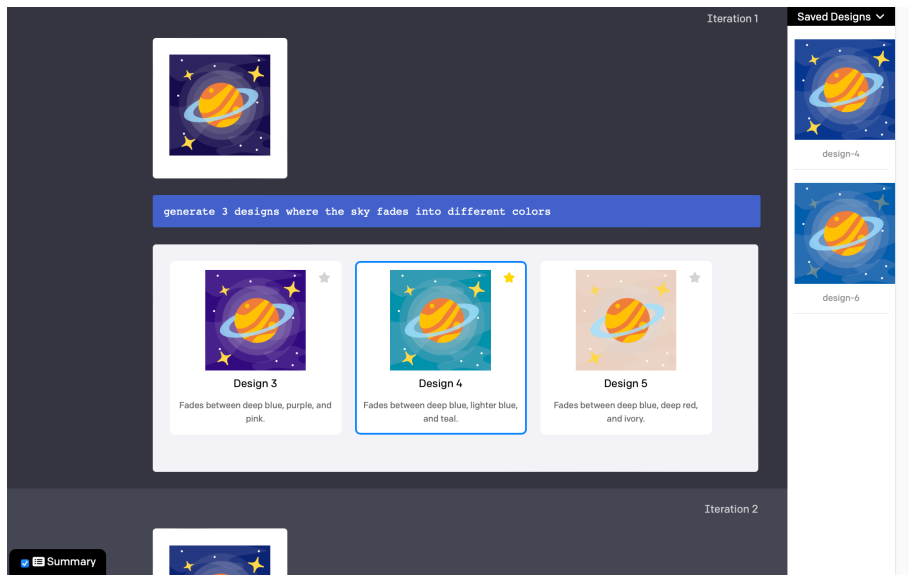
was generated so the user can revisit its corresponding and code. We also have a Summary mode which hides all text editors and displays the animations along with their prompts, enabling users to quickly revisit their previous prompts and designs.

5 USER STUDY METHODOLOGY

We conducted a user study to answer the following research questions about Keyframer:

- **RQ2** What design strategies do users take to prompt LLMs for animations using natural language?
- **RQ3** How does Keyframer support iteration in animation design?

5.1 Study Procedures

To understand the user experience of designing animations with Keyframer, we facilitated 90-minute sessions over video conference in which individuals built a set of animations with the tool.

At the beginning of the study, we presented a hypothetical scenario in which the participant is asked to help a friend, who is a children's book illustrator, design animations for her personal website using Keyframer. We chose this framing as it assumes the illustrations have already been created, enabling us to focus the user study solely on animation. We briefly introduced Keyframer to the participants, demonstrating how to generate animations from SVG images using natural language prompts, how to generate multiple designs at once, and how to use the Code and the Properties Editors. We also introduced participants to a third-party system called Boxy[11] that could be used during the study to explore the provided SVG images and read their element identifiers to use in prompting (see A.3 for further details).

---

[11] https://boxy-svg.com

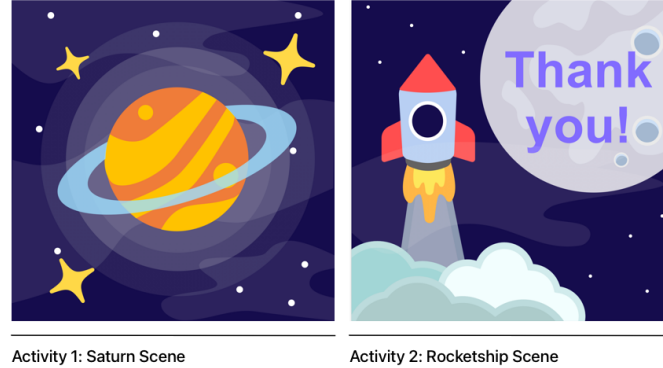Activity 1: Saturn Scene   Activity 2: Rocketship Scene

Fig. 7. The two illustrations used for our user study activities. The Saturn illustration contains 20 unique elements, including a Saturn, its ring, and yellow 'sparkles' (the three stars surrounding the planet). The Rocketship scene contains 19 elements, including a rocketship, clouds, and typography.

We then started the main part of the study by asking participants to create animations using two provided images, displayed in Figure 7. Activity 1 involved generating animations for a header image of the website, and Activity 2 consisted of animating an image thanking newsletter subscribers. Both SVG images were prepared by the authors of this paper in advance (full details on the creation of these SVGs can be found in A.2). For each activity, participants were first asked to explore the SVG using Boxy and brainstorm how they would like to animate the elements in the image. Then the participants used Keyframer freely for 15 minutes to generate animations with the goal of creating designs they would be happy to share with their friend. We asked participants to think aloud as they crafted their prompts and responded to LLM output. At the end of both activities, we interviewed participants about their experience creating animations using Keyframer. All participants completed Activity 1 followed by Activity 2.

After both activities, we asked participants a series of Likert scale questions on their satisfaction with the animations they created in 15 minutes, the helpfulness of the system for generating ideas for animations, and the helpfulness of the system for refining animations. Participants were also asked open-ended questions about their prompting strategies, feature requests, and potential use cases of using Keyframer for future work. The full details of our study design, research instruments, and study format can be found in A.4.

5.2 Participant Recruitment

To recruit participants, we advertised our user study on an location-based, general-purpose Slack channel with 2,500 members internal to a large technology company. We invited anyone interested in creating animations and described the study as centering around a 'Generative AI design tool for prototyping animations,' specifying that no coding experience was required. In addition, we reached out via email to participants in our formative study for snowball sampling among their co-workers.

Those interested in participating first filled out a screener survey, answering several questions about their job title, gender identity (for recruitment balancing purposes), any prior experience they might have with animation or coding, and whether they had experience using any AI tools. From the total pool of respondents, we selected a subset representing a range of programming and animation experience, as several professional animators interviewed in our formative study pointed out that Keyframer might be especially helpful for novices.

We mapped out a 2x2 matrix representing animation and programming experience such that we had representation from four groups: High Code and High Animation (HCHA), Low Code and Low Animation (LCLA), High Code Low Animation (HCLA), and Low Code High Animation (LCHA). A participant was considered low code if they had little or no prior programming experience; in contrast, a high code participant writes software professionally. Low animation users had little or no experience with animation design, while those categorized as high animation included users who had taken classes in animation design in college, do animation work professionally, or have taught animation coursework. We selected participants to have a balanced number of participants from each group.

5.3 Data Collection & Analysis

We recorded the video conference sessions (audio and video), giving us 19.5 hours of video for analysis. Additionally, we instrumented the application with logging features to create timestamped activity logs, tracking the prompts users entered, the full response from the LLM to each request, and which editors they used to edit the LLM output, if any. The anonymized logs were stored locally in the participant's browser and exported and shared with the researchers by the participants at the conclusion of each session. Transcripts were auto-generated by the video conferencing software and reviewed and edited by the research team.

Following the procedures of thematic analysis [8], two researchers met weekly over the course of two months to analyze the transcripts and logs from our sessions. We inductively coded the transcripts for participants' experience using Keyframer for animation design. In addition to analyzing the interview transcripts, we quantitatively analyzed all the prompts users generated in both activities, all the animations and code generated, and whether or not users edited the output from the LLM (and if applicable, what editor mode they used for their edits), comparing across all four groups.

Additionally, we again followed the procedures of thematic analysis to analyze prompting styles and semantic prompt types. We first independently coded a subset of 25 prompts (or 12% of all unique prompts) over multiple rounds to create a set of prompting styles and semantic prompt types. After reviewing and discussing any discrepancies in our application of these codes, we then split the list of prompts and independently coded half, discussing until we reached agreement on all prompts.

6 USER STUDY RESULTS

In this section, we first share a summary of our study participants and their overall impressions of using Keyframer. We then provide details about the performance of the LLM in generating animations, along with summary statistics about the designs and code generated. Next, we share results relating to prompting strategies (RQ2) and how Keyframer supported iteration in animation design (RQ3).

From 54 total survey respondents, we selected 13 participants (6 female, 7 male) to try out Keyframer in our user studies. Table 1 summarizes our participants, with three participants in each skill level category (with the exception of the HCHA group that had four participants, including one who was also in our formative study). Participants had a range of job titles including brand content manager, research engineer, and UX/UI designer, with a mix of skill levels in both animation and programming. Four participants (EP1, EP2, EP3, EP17) had no prior programming experience. All reported having tried out existing AI tools (11 out of 13 have used ChatGPT, and 7 have used Dall·E and/or Midjourney), largely to test their capabilities and for fun.

Overall, participants were satisfied by their experience with Keyframer. When asked to rate their satisfaction with the animations they generated in only 15 minutes, participants on average rated 3.9 (SD = 1.0), which is between

Table 1. Evaluative User Study Participants

Participant	Job Title	Group
EP1	Research Scientist	Low Code, Low Animation
EP2	Product Manager	Low Code, Low Animation
EP3	Brand Content Manager	Low Code, Low Animation
EP4	Software Engineering Manager	High Code, Low Animation
EP5	Machine Learning Engineer	High Code, Low Animation
EP6	Research Engineer	High Code, Low Animation
EP7	Research Engineer	High Code, High Animation
EP8	AR/VR Engineer	High Code, High Animation
EP9	UX/UI Designer	High Code, High Animation
EP10	Creative Technologist	High Code, High Animation
EP11	Instructional Designer	Low Code, High Animation
EP12	Product Design Engineer	Low Code, High Animation
EP13	Motion Designer	Low Code, High Animation

Fig. 8. Six frames taken from animations generated by EP4 (HCLA) and EP9 (HCHA). The Activity 1 Saturn animation from EP4 has the sparkles fade in and out independently, the clouds fade in, the halos fade in one after another, and the specks alternate colors from yellow to orange to pink. The Activity 2 Rocketship animation from EP9 has the rocketship move up and down, the clouds grow and shrink in size, and items in the background (the moon and specks) shift down to give the appearance of the rocketship lifting off.

'satisfied' (4) and 'neutral' (3). Participants generated 223 designs (94 for Activity 1, 129 for Activity 2). On average, each participant generated 17.2 designs (SD = 5.5). An example of two participants' final animations are displayed in Figure 8.

Participants were happily surprised by the efficacy of the system to turn natural language instruction into animations. Keyframer supported participants with limited expertise in motion design to generate animations from scratch efficiently. For instance, EP6 stated, "This is just so magical because I have no hope of doing such animations manually...I would find [it] very difficult to even know where to start with getting it to do this without this tool."

Participants appreciated that the system sped up their animation prototyping process: "I think this was much faster than a lot of things I've done... I think doing something like this before would have just taken hours to do" (EP2). Those without experience in programming animations found Keyframer "democratize[d] this whole creative process" and "highlights creativity more so than the mechanics of [the code]," letting users focus on higher level goals instead of having to know exactly how an animation could be implemented in CSS. Seeing LLM-generated CSS in Keyframer also offered novices "an interesting little window to see what's happening behind the scenes" (EP1) to connect code to animations.

Even professional motion designer EP13 saw the potential of Keyframer to extend his capabilities: "Part of me is kind of worried about these tools replacing jobs, because the potential is so high. But I think learning about them and using them as an animator — it's just another tool in our toolbox. It's only going to improve our skills. It's really exciting stuff."

System Performance. On average, 90.4 (SD = 36.7) lines of code were generated by the LLM in the final designs for Activity 1, and 87.2 (SD = 63.5) for Activity 2. The generated CSS from GPT-4 included CSS properties such as opacity, fill color, visibility, scale, and timing-function (a full list of generated CSS properties is listed in A.6). Keyframer was able to generate code for animations in a timely manner, taking on average 17.4 seconds (SD = 9.7s, max = 62.0s, min = 5.9s) to generate a code instance from a prompt. In addition, the code generated by the LLM were generally high-quality. Only 6.7% of GPT-4 generated CSS (n=15) resulted in code with syntactical errors (the details of which are provided in A.7). In those cases, participants would regenerate code using the same prompt or edit the prompt.

The most common challenges users encountered when requesting animations were having the LLM correctly interpret group versus individual level behavior, and getting the LLM to correctly execute time-sensitive sequencing. For example, in Activity 1, a common prompt request was to 'animate the sparkles to twinkle.' Users almost always meant for each of the three sparkles to animate independently, but because there was a group-level specification in the SVG (`<g id="sparkles">`), the LLM applied the animation to the three sparkles as a group. Users would then reprompt to specify independent animation, e.g., 'animate *each* sparkle to twinkle.' Another common challenge was getting two elements to move together, especially in the rocketship illustration. While this could be accomplished by grouping items in the underlying SVG code, doing so solely in CSS is more difficult as it requires matching time delays and duration, which the LLM did not always successfully execute.

6.1 RQ2: What design strategies do users take to prompt LLMs for animations using natural language?

Collectively, our participants generated 205 unique prompts.[12] On average, each participant generated 15.8 (SD = 4.0) unique prompts during their user study session. Participant EP1 generated the most unique prompts (26), and participant EP12 generated the least (11). On average, the prompts consist of 16.7 (SD = 15.2) words, with the longest prompt consisting of 142 words and the shortest prompt consisting of only 2 words ("remove exhaust").

Among all the unique prompts, 51.2% are new prompts (i.e., prompts entered after clicking the "Add New Prompt" button), while 48.8% are edits to existing prompts ('reprompts'). The majority of prompts (52.7%) requested refinement to the generated animation, while 42.9% requested new animation behavior for an item in the scene and 5.9% were a combination of the two. On average, each prompt asked for 2.0 (SD = 1.6) elements in the SVGs to be animated. The prompt that mentions the most number of elements contains 10 elements, while 3 prompts do not mention any elements and only provide high level instructions such as "Generate a few animations for this scene."

---

[12]Participants in total entered 221 prompts to the system. In some cases, the LLM would produce code with bugs and the participants were prompted to regenerate code using the same prompt. The count of unique prompts excludes those repetitive prompts.

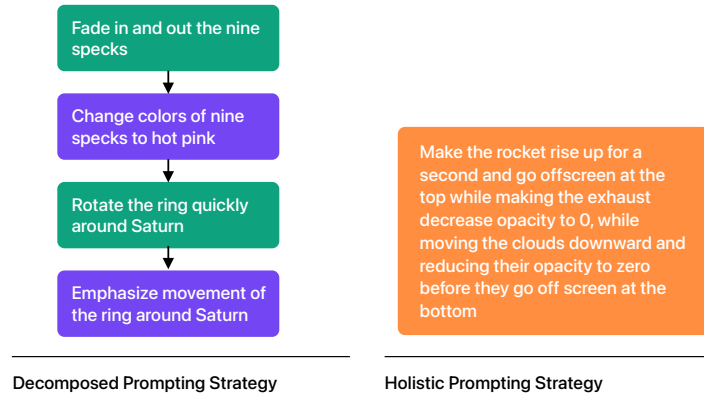**Decomposed Prompting Strategy** | **Holistic Prompting Strategy**

Fig. 9. Decomposed prompts involve sequential prompting to animate and refine individual elements within an image, alternating between specifying animation for a new item in the scene and refining the generated animation. In contrast, holistic prompting specifies the entire behavior in a single prompt.

We identified two different dimensions of prompting strategies: 1) decomposed versus holistic prompting, and 2) high specificity vs semantic prompting. Next, we describe the differences between these dimensions by illustrating with examples from our user study.

Decomposed vs Holistic Prompting. We classified prompts as *decomposed* if they involved animating elements one by one through sequential prompts. In contrast, *holistic prompts* describe how multiple elements within a scene should be animated simultaneously within a single prompt.

All users took on a decomposed prompting strategy for at least one activity, alternating between A) prompts specifying new animation behavior for an item in the scene, and B) prompts requesting refinements to the generated animation. Figure 9 displays on the left how EP3 first requests new animation for the 'specks' (shown in green), followed by a refinement prompt to change the specks colors (in purple), then repeats for the Saturn ring. Through this process, users focused on animating a single feature and getting it to a state they were satisfied with before moving on to animating another element: "It's how I'm used to doing animations—one piece at a time... I have to do this one animation, I have to tweak it, I have to do the next animation, and then I tweak it" (EP10).

Holistic prompting was observed with 4 of the 13 users, with each user using this strategy during the second activity in particular [13]. With holistic prompting, users specified the behavior of multiple items simultaneously, often defining expected coordination or timing between elements in a single prompt (as shown on the right in Figure 9 where EP11 specified how the rocket, exhaust, and cloud should move in one prompt). Software engineer EP4 described this strategy by sharing, "It's like sculpting. It was probably easier to try and get the rough shape of it all at the very beginning, and then fine tune the details versus trying to tell it cut by cut." Participants EP1 and EP11 used holistic prompting because they believed it was necessary to have items to move simultaneously, rather than one after another. Participant EP10 also saw this strategy as a time saving step, describing how he would use the initial generated design to do further refinement in the properties editor.

---

[13]We believe holistic prompting may have been more popular in the second activity because the illustration lended itself to more sequenced-based animation compared to Activity 1, with users specifying how items in the scene should respond to the rocket lifting off.

However, one challenge with holistic prompting was that in reprompting for multiple items to be animated at once, users run the risk of overriding desired behavior, as the temperature of the LLM requests was set to enable slight variations with the same prompt. This led users like EP5 to ask if it were possible to lock in certain parts of the code to prevent from being overwritten on regeneration.

In summary, the predominant use of decomposed prompting suggests that Keyframer enabled users to iteratively refine their designs through sequential prompting, rather than having to consider their entire design upfront. However, users that employed holistic prompting found a one-shot approach could effectively reduce time to generate by enabling more of the scene to be animated at once.

High Specificity vs Semantic Prompting. We observed two different styles of describing animations: 1) high specificity and 2) semantic prompts. High specificity prompts, which represented 34.6% of all unique prompts, are characterized by using animation keywords (like opacity, rotate, and scale) as well as expected values (e.g., 'between 0-1.5 seconds'). In contrast, the majority of prompts (84.4%) were semantic, which were less defined and more descriptive (e.g., 'make the clouds wiggle').[14] We present a taxonomy of semantic prompts types in Table 2, with prompts describing movement, timing, position, and coordination of objects in the scene, as well as incorporating analogies, indefinite values, and ideation requests, where users let the LLM come up with ideas (e.g., 'make it look cool'). Figure 10 shows the breakdowns of high specificity and semantic prompts among the four user groups; we observed the greatest percentage of semantic prompts from the LCHA group, while the highest use of high specificity prompts comes from those with programming experience (HC).

Table 2. Taxonomy of Semantic Prompt Types

Prompt Type	Definition	Frequency	Examples
Movement	Describing translation	68.4%	twinkle, wobble, balloon upwards
Timing	Describing speed	27.5%	slowly, very fast, with an elastic feel
Position	Describing relative placement	30.1%	from below, vertically on the same plane
Coordination	Relationships between items	19.7%	together, keep attached
Other Visual	Opacity, color change, fonts	8.8%	increasingly transparent, alternate color
Analogies	Similes or metaphors	2.6%	like stars in the sky, like an active fire
Indefinite Values	Indeterminate values	14.0%	at different rates, random value
Ideas	Requesting new ideas from the LLM	11.9%	do something crazy, show alternatives

Participants were surprised by how well the LLM could interpret semantic prompts. Participant EP7, who used words like 'morph,' 'shimmer,' and 'grow' in his prompts, stated, "I was generally impressed by how ambiguous I could be with my descriptions of the animation. It usually picked the correct transform to use, or a reasonable transform to use." This idea was echoed by EP8, who used the prompt *Give me 3 designs where the clouds wiggle*: "The fact that it got make the clouds wiggle in an interesting way, in a way that I was happy with, is kind of wild." This was surprising especially to users who questioned whether they needed to use specific animation keywords: "I would be curious if I can use human language, like pulse or twinkle or if I have to use, like, turn and rotation—more technical words" (EP9).

Others shared how being less specific opened up creative possibilities, potentially increasing the likelihood the LLM would return unexpected results. Participant EP13 described how he intentionally avoided using standard animation terminology in his prompts, believing that doing so might "give it [the LLM] more room to try something creative."

---

[14]We qualitatively coded the "high specificity" and "semantic" features among all 205 unique prompts. These features were not mutually exclusive, as some prompts could contain high specificity for one element and semantic descriptors for another; thus the percentages sum up over 100%.

High Specificity vs Semantic Prompts by User Group

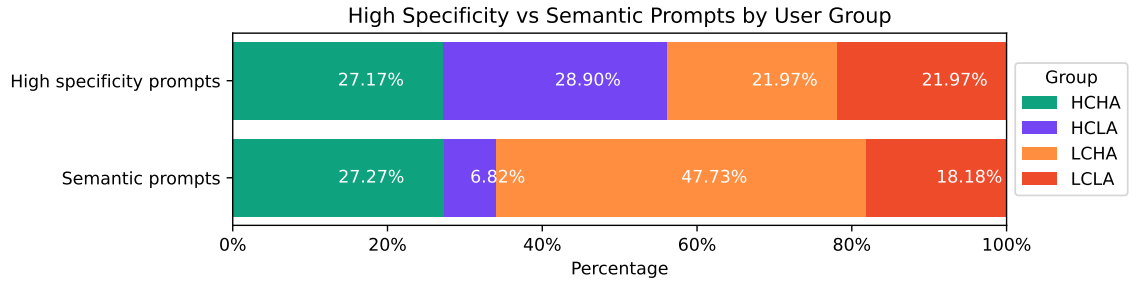|			
High specificity prompts	27.17%	28.90%	21.97% 21.97%
Semantic prompts	27.27%	6.82%	47.73% 18.18%

Group: HCHA, HCLA, LCHA, LCLA

Fig. 10. Percentages of high specificity and semantic prompts by participant groups.

In another example, when EP5 did not have a specific design in mind, he would prompt the system with a general instruction (such as "Make it look cool") to curate new ideas: "I am not that sure what to do sometimes... instead of explaining the steps of making it look as cool as possible at an atomic level...instead of being all imperative about it, you can just say what the goal is...[the LLM] can just fill in the rest." Participant EP4 summarized: "If you could be more vague [in your prompts], then you have more options for surprise and delight."

For others, semantic prompts were a time saving step, reducing the need to spend time defining all parameters upfront. For example, participant EP8, who transitioned from high specificity to more semantic prompts during her session as shown in Figure 11, said, "It's a lot easier to just assume it [the LLM] will do a pretty good duration, or that I could mess with it easily, so I don't need to type it all the way through."

Overall, we observed how semantic prompts enabled users to focus on higher level goals, both because the LLM had reasonable interpretations of their requests and because it returned sensible defaults that users could directly refine through Keyframer's editors.

For each of the specks animate them pulsing between 85% and 115% of their current size. Each pulse should take between .5 and 1.5 seconds.

• • •

Move the rocket, exhaust, and clouds from the bottom to the top.

→

Give me 3 designs where the clouds wiggle.

High Specificity

Low Specificity

Fig. 11. High specificity prompts (such as the one on the left) use animation keywords and values such as opacity and duration. Low specificity prompts (such as the two on the right) are more descriptive and more open to interpretation. These three prompts were all used by participant EP8, who started off with high specificity prompts but then transitioned to using semantic prompts.

Debugging Prompts. While participants were frequently surprised that Keyframer understood their prompts, they expressed how articulating motion in natural language was challenging in and of itself. There were several moments where the LLM output did not match a user's expectations, yet they acknowledged that the LLM had a reasonable interpretation: "It's definitely that sort of language barrier thing, where it did exactly what I wanted, but it turns out that wasn't what I wanted. It did exactly what I told it to do, but I should have told it to do something else" (EP4).

When the output did not match their expectations, it could be unclear if the source of the problem was the prompt or the LLM's capabilities: "I'm not sure if it was me or the program" (EP3). In these cases, the one sentence explanation generated alongside each code snippet, along with semantically descriptive generated keyframe names and animation properties, could aid users with verifying whether their prompts were interpreted correctly: "I like that it's writing it out. I can see what it's doing or how it's taking my commands" (EP9). Seeing the streaming output could also give users ideas for how to prompt in the future: "As it's generating code, you can kind of look into the black box of what is actually happening. If you see a field [name], that can be useful... if I want to change that property, that's the word I can use" (EP5). Additionally, the ability to generate multiple designs unexpectedly helped participant EP4 debug his prompt 'Generate 3 images of the rocket flying over the moon.' When he found that none of the generated designs matched his vision, he shared: "I clearly need to find a way to get it [the LLM] to understand what the moon is. With the 3 different outputs, it definitely ignored me, so I need to hone in on getting it to recognize that." Asking the LLM to generate three designs thus enabled him to confirm across multiple examples the need to refine his prompt.

Several users wanted the LLM to confirm its understanding before generating designs. For example, EP5 suggested a 'more interactive process': "It [the LLM] could give me feedback like, 'Do you mean this? Or 'Do you want to do this instead?"' Others proposed having the system make suggestions *during* the prompting process: "If I type in 'moon,' highlight the moon [in the image preview], but also in the text itself, make the word moon be colored so that I understand that it's a keyword" (EP4). Providing this type of visual feedback may help users gain more confidence that the system is capable of understanding keywords and requests.

6.2 RQ3: How does Keyframer support iteration in animation design?

In the following sections, we focus on how Keyframer shapes users' iteration process—including *exploration* and *refinement*—on their animation designs.

6.2.1 Exploration.

Participants regard Keyframer as useful in helping them brainstorm different ideas for animations, rating the system with a 3.7 (SD = 0.6) on average, which is between 'helpful' (4) and 'neutral' (3).

Surprise LLM output facilitates new ideas. At times, Keyframer surprised participants with a design that they had never thought of. Participants tended to value such surprises as they could spur new ideas. For example, EP8 once prompted the LLM to generate a design in Activity 2 where the clouds wiggle, whereas the LLM generated animations to the rocket as well. Although it was different from what EP8 has asked for, she ended up liking the animation and built subsequent designs on top of it. The generated output itself can seed future prompting; participant EP1, who is new to animations, shared, "I'm reacting to what I see as its output and then adjusting on my end...in some ways, it's informing and giving me ideas for what to explore next." Similarly, EP3 summarized her experience with the unexpected designs Keyframer generated: "Even if it doesn't come out the way you want it to, it's showing you another path that you can take... I think it's helpful to just come up with some other creative solutions." For those more experienced in designing animations, Keyframer could offer them new directions for technical implementation of animation effects, potentially broadening their animation skills. For instance, when EP7 prompted the LLM to create animation on the flame of the rocket as the rocket launches, the LLM generated code that stretched the flame vertically, which was not what he had in mind: "I thought I'd have to morph the flame to get something like this effect... I was like, 'Oh, this isn't bad either.' It's basically like another person suggesting something for me to try."

LLMs for generating design variations. The other approach was to explicitly ask the LLM to generate multiple designs, such as "generate 3 designs where the cloud wiggle." A total of 9 out of the 13 participants prompted the LLM to generate multiple designs. Participants thought the option to have the LLM generate multiple designs at once was helpful for brainstorming and comparing ideas: "It was nice to kind of see the different possibilities and then feel like I could choose among them which one was more closely aligned with" (EP1).

The option to generate multiple designs could be effective in different stages of the animation design process. In the very beginning of the design process where participants were exploring what to animate, they would leverage it to map different directions. As EP4 explained, "I know roughly what I want, but then if I could get it [the LLM] to initially show me like a larger set of possibilities, that would probably help me change the vision of what I want, because it's showing possibilities and then maybe more surprises." In later stages of the process, when participants were making decisions on certain details or parameters of the animation, they also valued experimenting with variations. For example, when EP2 was deciding on the speed of the animation of certain elements, he asked the LLM to generate variants to compare and contrast the effect of different duration values. In another example, EP5 thought generating design variants was a good way to overcome "artist block" when they encountered "friction" in the design process, and this could happen at the beginning of the process when they did not know where to start and towards the end when they were unsure what to do next.

Tensions with generating variants. Although some participants found the option to prompt the LLM to generate multiple designs helpful, they could be hesitant to use the feature in practice. When participants already had a clear idea of what they wanted, they would not generate multiple designs as it might steer them away from what they envisioned. Participants also reflected on their existing workflow when approaching creative solutions, where they typically focus on improving one element at a time instead of going broad. Participants from an engineering background, in particular, found a tension between the option to explore multiple design variations and the option to keep building on a single design: "I think it's probably part of being an engineer and trained to do things iteratively and see what happens, particularly with a non familiar system...it's a troubleshooting thing. If I ask it to do 10 things, I don't even know if it's going to do one correctly first" (EP4). Participant EP8 further explained the different needs they have when engineering versus designing animations: "When I'm thinking in the more design side, you don't know what you want until you see it sometimes. So it really helps to have a couple variations on what it looks like... If it's more pure engineering, I probably would just like the one design." Participants also wished that the variants could better match their designs goals, be more distinct, and take less time to generate. For example, EP5 shared, "I was hoping that it [the LLM] would maybe be willing to [take] more liberties with some of these suggestions," which aligned with feedback from EP2 and EP7 that sometimes only 1 or 2 of their variants were viable. We did not see any users request more than 3 designs and believe this may be because generating multiple designs leads to longer response times.

6.2.2 Refinement.

Participants in general consider Keyframer a helpful tool in assisting them with refining animations, rating 3.8 (SD = 1.0) on average, between "helpful" (4) and "neutral" (3). We observed two ways that the participants refine their animation designs using Keyframer: through editing code and editing prompts.

Code editing interfaces offer granular creative control. Our system supports participants with all levels of programming experience with making edits to LLM-generated code, even for those without programming background (i.e., those in the LCHA and LCLA groups). A majority of participants (69.2%) edited the LLM-generated code with either of Keyframer's editors. Seven participants edited the code directly using the Code Editor feature, while 7 edited

Table 3. Characteristics of code edits by participant group

Participant group	% of users who used the Code editor	% of users who used the Property editor	% of users who did not edit code at all	Mean % of code instances with user edits
LCLA	33.3%	66.7%	33.3%	16.7%
HCLA	33.3%	33.3%	66.7%	13.8%
HCHA	75%	75%	0%	35.6%
LCHA	66.7%	33.3%	33.3%	17.4%

the code using the Properties Editor feature. Five participants used both editors, and 4 participants did not make any edits to the code generated by LLM. On average, each participant made edits to 25.1% (SD = 0.2) of the instances where they received code generated by the LLM. Table 3 shows the editing actions by participant group. Notably, 4 out of 6 Low Code users edited output code, and of the 5 High Code users who edited LLM output, the majority (80.0%) used the Properties editor, suggesting that the Properties editor can be beneficial to those with all levels of programming experience. In fact, most participants that used the editors used *both* the Code and Properties editor throughout their use of Keyframer (55.6%), suggesting that each can be beneficial regardless of prior experience with animation or code.

Participants preferred to make direct edits on the code generated by the LLM when they had clearly defined goals: "It was nice that I was able to then go into the code and edit things that I didn't want to ask the model to do because I already knew exactly what I wanted" (EP4). For participants that were less familiar with CSS like EP2, the Properties Editor gave them an easier option to make these edits: "I thought it did a really good job at parsing the code and then giving me kind of a WYSIWYG type editor for the code. I think that's really helpful for people that aren't comfortable jumping into the code."

Combination of reprompting and code editing fuels design iteration. For those who chose to not edit the output code such as EP1 and EP4, editing prompts was an alternative way to modify the animation designs. Those who did not edit code output, like EP4, still acknowledged the potential benefit of making those edits, but chose not to due to the novelty effect of using a prompt-based interface and the time limit of the activity: "The prompt is the novelty, whereas hand editing [the code], there's no magic in that...I can absolutely do that by hand, but it's not as fun."

During the sessions, we observed the common practice of both re-prompting and editing output to refine animation designs. Participants shared the strategy of first using natural language to prompt for a design close to what they had imagined, and then leveraging the editing interfaces to adjust details: "I feel like the natural language part gets me 95% of the way there, and then the last 5% I would prefer to do edits here [the Code Editor]" (EP9).

Some valued using the LLM to generate boilerplate code that they could build off. CSS experts like EP7 appreciated that LLM-generated code saved them from having to look up syntax documentation: "Remembering all these names [of CSS properties] and writing them out always is annoying...it's nice that it [the LLM] shows all the transform origins and iteration count so I didn't have to think about that." Building off boilerplate code gives creators more time to focus on the design itself: "You can save so much time using a tool like this versus typing everything out. It allows you to do more stuff because you have more time to iterate" (EP13).

In fact, participants agreed that it was the combined powers of prompt input and code editing that made Keyframer helpful for refinement. For example, EP13 shared how he found himself largely refining his animations through manual code edits, rather than via prompting because the output editors reduced the amount of time to refine his animations.

7 DISCUSSION

Supporting design through iterative prompting. In our user study, the majority of participants took on a "decomposed" prompting strategy (explained in Section 6.1), with which they iteratively build up prompts, alternating between requests for animating new items and refinement of existing elements. This prompting strategy is a direct result of the *+ Add New Prompt* feature in Keyframer as part of its affordances for supporting design iteration (explained in Section 4).

This iterative way of prompting LLMs can effectively support collaboration between animation creators and the LLM. As users may not initially have a holistic picture on what they want their animation to look like, this prompting strategy allows them to iteratively develop ideas in collaboration with the LLM, exploring what is possible through small, incremental steps. In addition, this prompting strategy aids the understandability and control of the LLM as incremental changes can help designers connect their prompts to changes in generated code. These findings echo recent studies on "prompt chaining" with LLMs [50], which refers to the mechanism by which the output of one step becomes the input for the next and can improve system transparency and controllability.

However, different from [50], we found that with Keyframer, users were adaptively and fluidly refining their goals through responding to the output from the model. This prompting strategy also contrasts with the one-shot prompting process commonly found in current LLM-based text-to-image systems, which encourage users to describe their goals in a detailed, single prompt and refine the same prompt to achieve a desired output [35]. We argue that the decomposed prompting mechanism supported by Keyframer is more *human-centered*, allowing users to engage in an iterative design process [33] by gradually forming design goals as they converse with the LLM across multiple prompts. We hope that future LLM-powered design tools can consider this prompting strategy to increase user control and interpretability.

LLMs can empower a broad range of users to create animations in new ways. Several results positively support the use of LLMs to empower both beginners and professional animators. First, we found that the majority of prompts users created (84.4%) were semantic prompts descriptive of parameters like visual effects, timing, and coordination without using keywords commonly used in animation software. This result suggests that users do not need to have prior animation experience to effectively create animations using natural language.

Both "low code" (LC) and "high code" (HC) users engaged with editing LLM-generated code (80.0% and 71.4% respectively), indicating the value in enabling direct refinement of generated designs even for users without prior CSS experience. This is likely because many CSS properties can be interpretable to non-experts (such as duration and opacity). Showing a direct connection between rendered animations and the code behind it can support learning opportunities for creative computing. But even expert programmers appreciated the ability of the LLM to generate boilerplate code, saving them time to look up CSS syntax externally.

There were multiple occasions where the LLM generated designs that stimulated a user's creativity, whether through unexpected design output or through users explicitly prompting for new ideas; these results indicate that LLMs can encourage creativity in the domain of animations.

Open UX questions for integrating design variants. While users who requested multiple designs often found this feature valuable for overcoming creative blocks or comparing alternatives, users did not necessarily choose to use this feature on their own. All but one of the 10 participants who generated multiple designs did so only in the second activity after the facilitator asked whether they intentionally did not use the feature in the first activity. While some participants forgot about this feature, potentially due to the lack of UI indicators for generating multiple designs, many users described how their typical workflows do not normally accommodate these design explorations, particularly when they have clear goals of what they want to create. While some systems like Midjourney automatically generate

multiple designs, our participants expressed wanting to keep this feature opt-in, sharing how they would prefer as fast of a response as possible.

This time tradeoff, in which users have to actively decide whether asking for multiple designs is worth the longer response time, appears to impact use of this feature, along with parallel requests from participants for the LLM to produce richer, more unexpected design variants. We imagine multiple approaches to further investigate the use of variants, including fine tuning models to improve generated animation quality, considering alternative user interface for requesting variants (perhaps beyond or in conjunction with prompting), and providing feedback loops for users to understand what levers are available for better steering the LLM towards creating more distinct options.

Opportunities for improving interpretability. Participants were able to gain insight into the LLM's interpretation of their prompts through the combination of inline animation previews, descriptive generated code (such as keyframe names like 'flameFlicker,' and domain-relevant CSS properties like 'opacity' and 'fill'), and the generated explanations accompanying each design. Beyond these affordances, users offered several ideas for improving interpretability, many of which involve visual highlighting of relevant elements in both the SVG and generated animations. We heard requests for improving interpretability at three stages: during prompting, before design generation, and after the design has been generated. During prompting, participants suggested syntax highlighting when an identifier from the SVG was recognized in the prompt input field, helping them confirm the LLM's interpretation of keywords in their prompt. Before generating a design, some participants wanted the LLM to proactively confirm its understanding by asking a series of questions such as if the animation should happen repeatedly or just once. Once a user request has been made, others saw the multiple variant feature as a means to confirm the range of ways an LLM might interpret their prompt.

8 LIMITATIONS AND FUTURE WORK

A common user request shared by 7 of the 13 participants was to be able to edit the underlying illustration, which is not easily achieved with CSS alone. Not being able to edit the SVG itself prevents actions like adding new items to a scene, modifying the shape of objects, or re-grouping objects, which might limit a user's creativity, as described by EP11: "There's not a lot of space for brainstorming because a lot of decisions were made, like the shape of the rocket or the font." However, while an earlier version of Keyframer enabled generating CSS *and* SVG edits, it led to much longer response times and likelihood of exceeding token limitations because of the amount of code in the SVG itself. So while it might be ideal for users to be able to leverage the LLM to make changes to the illustration metadata directly, doing so can impact the performance of the system.

We designed Keyframer to resemble the ChatGPT interface in which prompting happens sequentially and linearly. However, this linear representation was not necessarily representative of how some of our users approach animations. For example, EP12 stated that he would typically tune the movement of two items independently and simultaneously throughout the process of creating an animation. A non-linear representation may help users expecting to tweak and merge different animated elements together, as explored in related work [3, 17].

Finally, many users wanted prompt-based interfaces alongside direct manipulation controls they are accustomed to in applications like After Effects. Users such as EP13 described how transform operations can be faster through direct manipulation (moving items to different places for individual keyframes), while more complex, non-translational animations might be easier through prompting. We imagine future AI-powered animation tools combining the affordances of direct manipulation with prompting for streamlining the definition and editing of animated properties.

9 CONCLUSION

Through Keyframer, we introduced how LLMs can shape future animation design tools by supporting iterative prototyping across exploration and refinement stages of the design process. Keyframer takes advantage of the code generation capabilities of LLMs and affordances for natural language input to empower users to generate animations using highly semantic prompts. Through our exploratory user study, we contribute a taxonomy of semantic prompting styles observed from users describing motion in natural language, ranging from those describing timing and coordination, along with more open-ended prompts soliciting ideas from the LLM (e.g., "make it look cool"). Further, we illustrated how we can enable animation creators to maintain creative control by providing pathways for iteration, with users alternating between prompting and editing generated animation code to refine their designs, and users building up designs through sequential prompting (what we describe as 'decomposed' prompting). Yet, we also found that Keyframer users found value in unexpected LLM output that helped spur their creativity. One way we observed this serendipity was with users testing out Keyframer's feature to request design variants. Through this work, we hope to inspire future animation design tools that combine the powerful generative capabilities of LLMs to expedite design prototyping with dynamic editors that enable creators to maintain creative control in refining and iterating on their designs.

REFERENCES

[1] Robin S Adams. 2002. Understanding design iteration: Representations from an empirical study. In *Common Ground - DRS International Conference 2022*. 5–7.

[2] Robin S Adams, Jennifer Turns, and Cynthia J Atman. 2003. Educating effective engineering designers: The role of reflective practice. *Design studies* 24, 3 (2003), 275–294.

[3] Tyler Angert, Miroslav Suzara, Jenny Han, Christopher Pondoc, and Hariharan Subramonyam. 2023. Spellburst: A Node-based Interface for Exploratory Creative Coding with Natural Language Prompts. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–22.

[4] Artbreeder. 2022. Collager. https://www.artbreeder.com/create/collage.

[5] Linden J Ball, Jonathan St. BT Evans, Ian Dennis, and Thomas C Ormerod. 1997. Problem-solving strategies and expertise in engineering design. *Thinking & Reasoning* 3, 4 (1997), 247–270.

[6] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[7] Stephen Brade, Bryan Wang, Mauricio Sousa, Sageev Oore, and Tovi Grossman. 2023. Promptify: Text-to-image generation through interactive prompt exploration with large language models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[8] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[9] Bill Buxton. 2010. *Sketching user experiences: getting the design right and the right design*. Morgan kaufmann.

[10] William Buxton and Richard Sniderman. 1980. Iteration in the design of the human-computer interface. In *Proc. of the 13th annual meeting, Human Factors Association of Canada*. 72–81.

[11] Bradley Camburn, Vimal Viswanathan, Julie Linsey, David Anderson, Daniel Jensen, Richard Crawford, Kevin Otto, and Kristin Wood. 2017. Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science* 3 (2017), e13.

[12] Li-Yuan Chiou, Peng-Kai Hung, Rung-Huei Liang, and Chun-Teng Wang. 2023. Designing with AI: An Exploration of Co-Ideation with Image Generators. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*. 1941–1954.

[13] DaEun Choi, Sumin Hong, Jeongeon Park, John Joon Young Chung, and Juho Kim. 2023. CreativeConnect: Supporting Reference Recombination for Graphic Design Ideation with Generative AI. *arXiv preprint arXiv:2312.11949* (2023).

[14] Design Council. 2004. Framework for Innovation. https://www.designcouncil.org.uk/our-resources/framework-for-innovation/

[15] Nigel Cross. 2004. Expertise in design: an overview. *Design studies* 25, 5 (2004), 427–441.

[16] Nigel Cross. 2006. *Designerly ways of knowing*. Springer.

[17] Hai Dang, Frederik Brudy, George Fitzmaurice, and Fraser Anderson. 2023. WorldSmith: Iterative and Expressive Prompting for World Building with a Generative AI. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–17.

[18] Steven Dow, Julie Fortuna, Dan Schwartz, Beth Altringer, Daniel Schwartz, and Scott Klemmer. 2011. Prototyping dynamics: sharing multiple designs improves exploration, group rapport, and results. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2807–2816.

[19] Steven P Dow, Alana Glassco, Jonathan Kass, Melissa Schwarz, and Scott R Klemmer. 2009. The effect of parallel prototyping on design performance, learning, and self-efficacy. *Stanford Technical Report* 10 (2009).

[20] Steven P Dow, Alana Glassco, Jonathan Kass, Melissa Schwarz, Daniel L Schwartz, and Scott R Klemmer. 2010. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Transactions on Computer-Human Interaction (TOCHI)* 17, 4 (2010), 1–24.

[21] Steven P Dow, Kate Heddleston, and Scott R Klemmer. 2009. The efficacy of prototyping under time constraints. In *Proceedings of the seventh ACM conference on Creativity and cognition*. 165–174.

[22] Sarah Drasner. 2017. *SVG animations: from common UX implementations to complex responsive animation*. " O'Reilly Media, Inc.".

[23] Brock U Dunlap, Christopher L Hamon, Bradley A Camburn, Richard H Crawford, Dan D Jensen, Matthew G Green, Kevin Otto, and Kristin L Wood. 2014. Heuristics-based prototyping strategy formation: development and testing of a new prototyping planning tool. In *ASME International Mechanical Engineering Congress and Exposition*, Vol. 46606. American Society of Mechanical Engineers, V011T14A019.

[24] Omar Elharrouss, Noor Almaadeed, Somaya Al-Maadeed, and Younes Akbari. 2020. Image inpainting: A review. *Neural Processing Letters* 51 (2020), 2007–2028.

[25] Yingchaojie Feng, Xingbo Wang, Kam Kwai Wong, Sijia Wang, Yuhong Lu, Minfeng Zhu, Baicheng Wang, and Wei Chen. 2023. PromptMagician: Interactive Prompt Engineering for Text-to-Image Creation. *IEEE Transactions on Visualization and Computer Graphics* (2023).

[26] Katy Ilonka Gero, Tao Long, and Lydia B Chilton. 2023. Social dynamics of AI support in creative writing. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.

[27] Frederic Gmeiner, Humphrey Yang, Lining Yao, Kenneth Holstein, and Nikolas Martelaro. 2023. Exploring Challenges and Opportunities to Support Designers in Learning to Co-create with AI-based Manufacturing Design Tools. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–20.

[28] Björn Hartmann, Scott R Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*. 299–308.

[29] Val Head. 2016. *Designing interface animation: improving the user experience through animation*. Rosenfeld Media.

[30] Steven Heller and David Womack. 2011. *Becoming a digital designer: A guide to careers in web, video, broadcast, game and animation design*. John Wiley & Sons.

[31] David G Jansson and Steven M Smith. 1991. Design fixation. *Design studies* 12, 1 (1991), 3–11.

[32] Martin Jonsson and Jakob Tholander. 2022. Cracking the code: Co-coding with AI in creative programming education. In *Proceedings of the 14th Conference on Creativity and Cognition*. 5–14.

[33] Tom Kelly. 2002. The art of innovation. *Profile Business* (2002).

[34] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. mage: Fluid moves between code and graphical work in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 140–151.

[35] Chinmay Kulkarni, Stefania Druga, Minsuk Chang, Alex Fiannaca, Carrie Cai, and Michael Terry. 2023. A Word is Worth a Thousand Pictures: Prompts as AI Design Material. *arXiv preprint arXiv:2303.12647* (2023).

[36] Vivian Liu and Lydia B Chilton. 2022. Design guidelines for prompt engineering text-to-image generative models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–23.

[37] Vivian Liu, Jo Vermeulen, George Fitzmaurice, and Justin Matejka. 2023. 3DALL-E: Integrating text-to-image AI in 3D design workflows. In *Proceedings of the 2023 ACM designing interactive systems conference*. 1955–1977.

[38] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J Cai. 2020. 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*. 1–13.

[39] Geethu Miriam Jacob, Sagar Sahu, Chisato Ito, Masashi Kawamura, and Bjorn Stenger. 2024. Text2Illustration: Sampling and Composing Creatives With Text. In *Proceedings of the 7th Joint International Conference on Data Science & Management of Data (11th ACM IKDD CODS and 29th COMAD)*. 538–541.

[40] Michael Muller, Lydia B Chilton, Anna Kantosalo, Charles Patrick Martin, and Greg Walsh. 2022. GenAICHI: generative AI and HCI. In *CHI conference on human factors in computing systems extended abstracts*. 1–7.

[41] W Lawrence Neeley Jr, Kirsten Lim, April Zhu, and Maria C Yang. 2013. Building fast to think faster: exploiting rapid prototyping to accelerate ideation during early stage design. In *international design engineering technical conferences and computers and information in engineering conference*, Vol. 55928. American Society of Mechanical Engineers, V005T06A022.

[42] Jonas Oppenlaender. 2023. A taxonomy of prompt modifiers for text-to-image generation. *Behaviour & Information Technology* (2023), 1–14.

[43] Savvas Petridis, Michael Terry, and Carrie J Cai. 2023. PromptInfuser: How Tightly Coupling AI and UI Design Impacts Designers' Workflows. *arXiv preprint arXiv:2310.15435* (2023).

[44] Téo Sanchez. 2023. Examining the Text-to-Image Community of Practice: Why and How do People Prompt Generative AIs?. In *Proceedings of the 15th Conference on Creativity and Cognition*. 43–61.

[45] Jaskirat Singh, Liang Zheng, Cameron Smith, and Jose Echevarria. 2022. Paint2pix: interactive painting based progressive image synthesis and editing. In *European Conference on Computer Vision*. Springer, 678–695.

[46] Sangho Suh, Meng Chen, Bryan Min, Toby Jia-Jun Li, and Haijun Xia. 2023. Structured Generation and Exploration of Design Space with Large Language Models for Human-AI Co-Creation. *arXiv preprint arXiv:2310.12953* (2023).

[47] Jakob Tholander and Martin Jonsson. 2023. Design ideation with ai-sketching, thinking and talking with Generative Machine Learning Models. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*. 1930–1940.

[48] Mou Tsai-Yun, Jeng Tay-Sheng, and Chen Chien-Hsu. 2013. From storyboard to story: Animation content development. *Educational Research and Reviews* 8, 13 (2013), 1032–1047.

[49] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[50] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*. 1–22.

[51] Yu-Chun Grace Yen, Steven P Dow, Elizabeth Gerber, and Brian P Bailey. 2017. Listen to others, listen to yourself: Combining feedback review and reflection to improve iterative design. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*. 158–170.

A APPENDIX

A.1 Formative Interviews Participants

Our formative interview participants consisted of professionals who create animators as part of their work at a large technology company.

Table 4. Formative User Study Participants

Participant	Job Title	Animation Domain	Years of Experience	Animation Tools
FP1	Designer	UI Animation	8	After Effects, Cinema 4D
FP2	Creative Technologist	Advertising	22	Javascript, After Effects
FP3	Technical R&D Artist	Character Design	10	Maya, Blender
FP4	Front-end Developer	Advertising	7	Javascript, After Effects
FP5	Software Engineer	Instructional Content	6	After Effects
FP6	UI Prototype Engineer	UI Animation	15	Swift, After Effects
FP7	Research Engineer	Data Viz	8	JavaScript
FP8	Research Engineer	Data Viz	3	Swift
FP9	Web Technologist	Data Viz & UI Animations	20	Javascript, Unity

A.2 Illustration Creation

We created two illustrations for users to animate, shown in Figure 7. We intentionally designed the two illustrations to help us observe a range of user prompting styles. For the Saturn illustration, multiple items can be animated simultaneously without timing dependencies. But for the Rocketship illustration, users might want items to move elements together in sequence (such as the rocketship and clouds moving together, followed by the text popping up). Given that we anticipated that prompting for timing between elements may be more difficult, we had users first animate the Saturn scene followed by the Rocketship illustration.

A.2.1 SVG Pre-Processing. For our user study, we did not expect users to need to have any knowledge of SVG syntax to successfully complete our tasks — we wanted users to pay attention only to the identifiers for items in the scene (which could be used for prompting the LLM). All other SVG properties (like fills or path origins) could be ignored. Additionally, to reduce the LLM response time and likelihood of encountering token limitations (as the SVG itself is passed in each request to GPT-4), we took efforts to make the SVGs as small as possible by reducing the lines of code in the XML. To accomplish both of these goals, we pre-processed the SVGs through the following steps:

Step 1: Create Illustrations in Sketch. We created the original illustrations in the design software Sketch. In Sketch, we gave layers descriptive names (e.g., 'saturn' or 'halos') and grouped them as appropriate (e.g., a 'halos' group included 'halo-', 'halo-2', and 'halo-3'). We then exported each illustration as an SVG.

Step 2: Remove Inline Transforms. We used Inkscape (https://inkscape.org/), an open-source vector graphics tool, along with the Inkscape extension Apply Transforms (https://github.com/Klowner/inkscape-applytransforms) to remove transform definitions within the SVG XML. This step was taken to minimize potential issues with applying CSS transforms like rotations that could conflict with transform properties already defined in the SVG.

Step 3: Minimize SVG. We used the tool SVGOMG (https://jakearchibald.github.io/svgomg/) to simplify the SVG by removing unnecessary metadata like docstyle or XML instructions not needed for our study.

Step 4: Reformatted SVG Identifiers. We manually edited the SVG code to move all identifiers to the beginning of an element definition (e.g., `<path id="sky".../>`) so that when users are quickly inspecting the SVG code, they can easily find the id of each element without having to scroll horizontally in the code editor.

After pre-processing, the SVG for Activity 1 consisted of 42 lines of code, and the SVG for Activity 2 consisted of 40 lines of code, translating to 2,503 and 3,667 tokens respectively for GPT-4[15].

A.3 Using Boxy for Reviewing SVG Elements

We used a web-based third party tool called Boxy[16] for participants to interactively inspect the SVG so they could familiarize themselves with the identifiers of items in the illustration. These identifiers could then be used in Keyframer to animate the scene via natural language prompts. We loaded each illustration into Boxy, which can display the SVG illustration alongside its code as shown in Figure 12. With Boxy, users can click on items within the image to reveal its identifier. You can also click on a line of code to highlight the corresponding item in the illustration.

A.4 User Study Procedures

Each facilitated 90-minute session involved the following protocol:

Intros (15 min) Brief introductions and several questions about the participant's background (including what they do for work and what prior experience they may have with animation and programming)

Boxy Intro (3 min) The facilitator shared their screen and gave a brief demo of Boxy with a sample illustration distinct from the two used in the user study task (Figure 12). They demonstrated both how to click on items in the image to see the corresponding code, and how to click on lines of code to highlight the item in the illustration.

Keyframer Demo (5 min) The facilitator walked through the Keyframer interface and described how to use it through an example. The example used an SVG whose scene consists of a sun, several clouds, a sky, and a hill. The facilitator first entered the prompt 'animate the sun to move from the top left to the bottom right.' Once the LLM fulfilled this request, they showed how you can either edit the output animation through the code editor or the property editor. Next, the facilitator demonstrated how you can continue animating the scene by creating a new iteration. They then entered the prompt 'generate 3 designs where the sky changes colors like a sunset' to show how you can request multiple examples from the LLM. Once three designs variants appeared, the facilitator showed how custom UI elements appear for different CSS properties (in this case, showing how you can edit color properties generated by the LLM using a color picker). The facilitator then asked the participant if they have any questions before proceeding.

Activity 1 Boxy Exploration (5 min). The facilitator introduced the design scenario and sent the participant a link to Boxy to inspect the Saturn illustration (Figure 7). At this point, the participant shared their screen for the remainder

---

[15]https://platform.openai.com/tokenizer
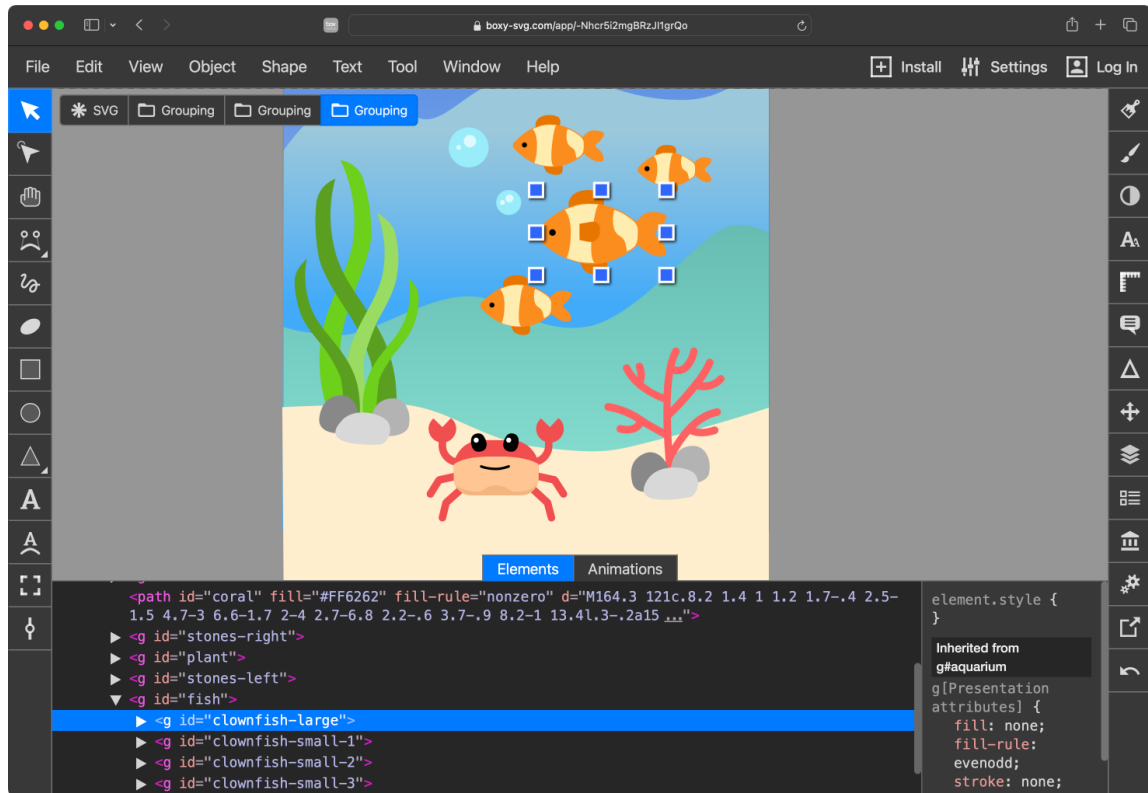[16]https://boxy-svg.com/

Fig. 12. Boxy, a 3rd-party application we used for participants to explore each SVG illustration before using Keyframer. Clicking on an item in the scene reveals the corresponding line of code and its identifier (in this example 'clownfish-large' for the selected fish). Likewise, clicking on a line of code will highlight the corresponding item in the illustration.

of the session. The participant was asked to take 5 minutes to explore the SVG in Boxy to get the name of elements in the scene and come up with a few ideas for what they might want to animate, thinking aloud as they worked.

Activity 1 Keyframer Exploration (15 min). The facilitator sent the participant a link to the Keyframer app they could open in their web browser. They had 15 minutes to come up with a few animation designs they are happy with and could share with their friend. During this time, the participant entered as many prompts as they wanted to create their design.

Activity 1 Interview (5-10 min) The facilitator asked the participant several questions about their experience including: What went well? What did not go as expected? Did anything surprise you about using the LLM?

Activity 2 Boxy Exploration, Keyframer Exploration, and Interview (5 min, 15 min, 5-10 min) We repeat the same process for the second illustration (Rocketship scene) where the participant was asked to create several animations they are happy with, followed by the same questions about their experience.

Closing Interview (10 min) We closed the session asking several questions about their overall experience. First, we asked users to rate on a scale from 1-5 their response to three questions: 1) How satisfied are you with how far you were able to get with your animation in 15 minutes?, 2) How helpful do you think this system was for helping you brainstorm different ideas?, and 3) How helpful do you think this system was for helping you refine your animations?

(with 1 being very unsatisfied, 3 being neutral, and 5 being very satisfied). We then followed with several open-ended questions, including: Do you think you discovered any strategies for prompting that seemed to work better? Are there features you would want this tool be able to do? Could you imagine using this type of technology for animation design in the future? All participants received two $12 cafeteria vouchers for their time.

A.5 Full Prompt for Generating CSS Animation Code with an Input SVG

```
You are writing CSS for animating the SVG contained within the <> symbols below. The design
    should meet the following user specification:

###
<User-entered prompt>
##

Please follow these rules in writing the CSS:
1. Contain the CSS code snippet in a <style> element.
2. In the CSS code snippet, do not use animation shorthand and use the property "animation-
    name".
3. If there is any transform: rotate() or transform: scale() in the code snippet, set
    transform-origin: center and transform-box: fill-box.
4. The animation should repeat forever with animation-iteration-count: infinite.
5. The CSS for a code snippet should be nested within a parent with the class design-n
    where n corresponds to the index of the snippet counting up from <number of existing
    designs>. ONlY add this parent-class to CSS rules. DO NOT add the parent class to
    keyframes.
6. A code snippet should be followed by a short explanation summarizing how the design is
    distinct. The explanation should be no more than 15 words long, should be descriptive
    rather than technical, and should be contained in an <explanation> tag.
7. Only write CSS. Do not return any SVG or additional text.

If the user asks for more than one design, follow these addition rules FOR EACH DESIGN:
1. Generate independent CSS code snippets and explanations for each design.
2. End each CSS code snippet and explanation with the delimiter -----. Ensure the delimeter
    has 5 dashes.

<>
 <The SVG Code>
```

```
<>
```

If the user is working in a new iteration (e.g., the generated code should build off of a selected design), we also include these instructions for extending the existing CSS code:

```
For all generated designs, start from the existing add any new CSS BELOW the existing CSS.

"""<Existing CSS>"""

Refactor the existing CSS to apply the corresponding design-<number-of-existing-designs>
    class.
RETAIN ALL LINE OF IN THE EXISTING CSS. DO NOT DROP ANY LINES.
Check your work and ensure that the existing CSS is represented in the designs.
```

A.6   Generated CSS Properties

Table 5 presents the range of GPT-4 generated CSS property types in its responses to user prompts:

A.7   CSS Errors

We found 15 instances of syntactically incorrect generated CSS from GPT-4 out of the 233 total prompts entered during the study (a 6.7% error rate) . Errors in CSS will still render the SVG within Keyframer, but the image will not appear animated. A summary of the errors identified are shown below. Note that we are using GPT-4 as is without any fine-tuning. Table 6 summarizes the CSS errors identified.

Table 5. Generated CSS Properties

CSS Property	Example Values
Timing	
duration	5s
timing-function	linear, ease-in-out
delay	1.5s
Transforms	
scale, scaleX, scaleY	scale(1.5)
translate, translateX, translateY	translateY(-100%)
rotate, rotateX, rotateY	rotate(10deg)
Other Appearance	
opacity	0.5
fill	cyan, #0000FF
visibility	hidden, visible
filter	brightness(100%)
font-family	Courier, monospace
animation-	
direction	alternate, alternate-reverse, forward
play-state	running
fill-mode	both

Table 6. CSS Syntax Errors

Type	Count	Example
Classname applied to keyframe	6 (40.0%)	**.design-9 @keyframes planet-rotate {}** instead of **@keyframes planet-rotate {}**
Class notation instead of id	3 (20.0%)	**.design-0 .flame** instead of **.design-0 #flame**
Incorrect class refactoring	2 (13.3%)	**.design-0 #rocketship {} .design-3 #exhaust {}** (both should use **.design-3**)
Typo	2 (13.3%)	**<stle>** instead of **<style>**
Invalid CSS	1 (6.6%)	**#sparkle-1, #sparkle-2, #sparkle-3 { animation-delay: 0.5s, 1s, 1.5s}**
Undefined variable	1 (8%)	**calc(30deg * var(—random))** when **random** has not been defined