LogoMotion: Visually-Grounded Code Synthesis for Creating and Editing Animation

Vivian Liu
Columbia University
New York, New York, USA
vivian@cs.columbia.edu

Rubaiat Habib Kazi
Adobe Research
Seattle, Washington, USA
rhabib@adobe.com

Li-Yi Wei
Adobe Research
San Jose, California, USA
liyiwei@acm.org

Matthew Fisher
Adobe Research
Palo Alto, California, USA
matfishe@adobe.com

Timothy Langlois
Adobe Research
Seattle, Washington, USA
tlangloi@adobe.com

Seth Walker
Adobe Research
San Francisco, California
USA
sewalker@adobe.com

Lydia Chilton
Columbia University
New York, New York, USA
chilton@cs.columbia.edu

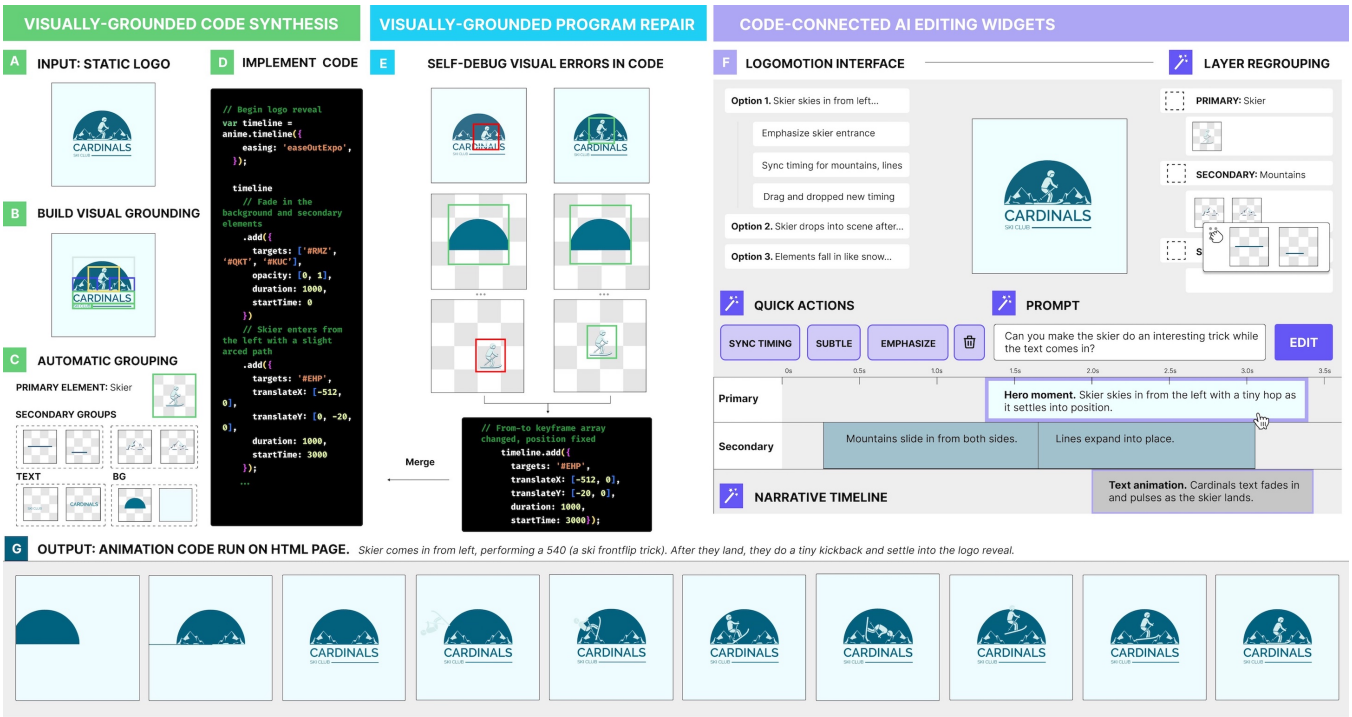


Figure 1: LogoMotion is an AI code generation approach that helps users create semantically meaningful animation for logos. LogoMotion automatically generates animation code for a logo design using visually-grounded code synthesis and program repair. This method performs visual analysis, instantiates a design concept, and conducts visual checking to generate animation code. Novices can use code-connected AI editing widgets to make targeted edits to the motion, grouping, and timing of their animation through familiar GUI controls.

Abstract

Creating animation takes time, effort, and technical expertise. To help novices with animation, we present LogoMotion, an AI code generation approach that helps users create semantically meaningful animation for logos. LogoMotion automatically generates animation code with a method called visually-grounded code synthesis and program repair. This method performs visual analysis, instantiates a design concept, and conducts visual checking to generate animation code. LogoMotion provides novices with code-connected

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CHI '25, Yokohama, Japan

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3714155

AI editing widgets that help them edit the motion, grouping, and timing of their animation. In a comparison study on 276 animations, LogoMotion was found to produce more content-aware animation than an industry-leading tool. In a user evaluation (n=16) comparing against a prompt-only baseline, these code-connected widgets helped users edit animations with control, iteration, and creative expression.

CCS Concepts

• **Applied computing** → **Media arts**; • **Computing methodologies** → *Natural language processing*; *Computer vision tasks.*

Keywords

animation, large language models, motion design, program synthesis, code generation, GPT, logos

ACM Reference Format:

Vivian Liu, Rubaiat Habib Kazi, Li-Yi Wei, Matthew Fisher, Timothy Langlois, Seth Walker, and Lydia Chilton. 2025. LogoMotion: Visually-Grounded Code Synthesis for Creating and Editing Animation. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3706598.3714155

1 Introduction

Motion suggests life [45]. Motion designers use animation to signal change, suggest relationships between objects, convey personality, and connect objects to the actions and activities they take on in the real world. They make cars drive, bumblebees fly, and stars twinkle, because people have visual expectations for how these objects should move. The dynamics of animation can further communicate meaning. A delivery truck driving in fast onto the screen can symbolize speedy service; the way a circle vertically moves can symbolize something as gentle as a sunrise or something as playful as a bouncing ball. Animation is delightful when it is semantically meaningful: when it "reflects and reinforces the semantic relationships" between design elements [44].

Currently, creating semantically meaningful animation is difficult for novices. Professional tools like After Effects allow this, but they can require technical expertise and fine-grained editing that can be time consuming. End-user animation tools such as Canva and Adobe Express are popular with novice designers and provide support for animation. However, their animation tools are largely based on animation presets and templates. For example, a novice could want to animate a logo for a Ski Club logo that has a skier, some mountains, lines, and text (as pictured in Fig. 1). They can apply an animation preset that automatically animates the page and has everything "come in from the left" or "fade in". They could also edit the speed or intensity of the motion in parameters or apply other preset motion styles such as "energetic" or "professional". However, while templates and presets can create smooth, well-executed animation, they can be limited at creating motion that truly reinforces the logo's visual message (e.g. making a skier ski in and do a flip).

Generative AI can move beyond rigid and rule-based systems like templates and presets to provide more flexible means of creation. It can generate animation code that is more complex and expressive than presets. Furthermore, AI has the multimodal capability to perform visual analysis and understand the semantics of an image, such as what each element depicts (a skier, a mountain) and what role each element plays on the canvas (foreground, background, groups). For example, an AI can visually analyze that the skier in the logo (Fig. 1-A) is the primary element and that there are a group of mountains. Using this analysis, it can come up with a design concept ("have the skier ski in from the left and do a flip into place, the mountains rise in, and the text enter as the skier lands") and implement code executing that concept. We call this **visually-grounded code synthesis**, a method that performs visual analysis and instantiates a design concept to guide the AI implementation of animation code . However, AI-generated code can have errors, and AI approaches should help users fix the errors they create. We introduce **visually-grounded program repair** to run AI-generated code, visually inspect the animation result for errors ("skier position is off"), and let the AI automatically debug itself. Together, visually-grounded code synthesis and repair generate an animation that is semantically meaningful to the logo design (Fig. 1-G).

Although this automatic logo animation approach often generates good results by itself, users should be able to edit and perfect it. Prompt-based editing of code has potential [11, 52] but can lack the control necessary for editing animation. A prompt can regenerate the entire animation code, change parts the user likes, or introduce new errors as the AI tries to fulfill the user's desired edit. To offer users more control over the editing of generated code, we introduce **code-connected AI editing widgets**. LogoMotion's editing UI contains familiar animation editing widgets such as a timeline, a layer panel, and quick actions ("more subtle", "faster", "synchronize timing"). When the user interacts with these widgets, LogoMotion translates the GUI interaction into targeted edits over the code corresponding to the motion, grouping, and timing of the animation. These code-connected AI editing widgets let users gain the expressiveness and controllability of a code representation while having a visual representation that lets them focus on the creative aspects of editing.

Our contributions are as follows:

- **Visually-Grounded Code Synthesis and Program Repair** automatically generates semantically meaningful animations for logo designs. This method performs visual analysis, provides a design concept, and applies visual checking over the canvas to generate animation code.
- **Code-Connected AI Editing Widgets** enable users to edit AI-generated code with widgets such as a narrative timeline, layer panel, and quick actions. These widgets bind GUI controls to targeted code edits, so users can edit the motion, grouping, and timing of their animation.
- Two technical evaluations showing that 1) in comparison study of 276 animations, visually-grounded code synthesis produces more semantically meaningful animations than an industry-leading animation tool, 2) visually-grounded program repair can solve 96% of detected errors within four attempts.

- A user study (n=16) comparing with a prompt-only baseline showed that code-connected AI widgets helped novices edit with control, iteration, and creative expression.

We conclude with a discussion of how LogoMotion's methods of combining code representations with visual understanding and code-connected widgets can apply to other design tasks beyond logo animation.

2 Related Work

2.1 Program Synthesis

Program synthesis, the formal name for code generation, is the idea that given a high-level specification of a problem, a search space of potential program solutions can be automatically searched to find a provably correct solution [26]. While program synthesis originated in the domain of formal methods and boolean SAT solvers, it has evolved greatly since the introduction of machine learning and large language models. The state of the art models for code generation include GPT-4, AlphaCode, CodeGEN, Code Llama, and Gemini [23, 39, 41, 47, 51]. These models generally take in a natural language specification of the problem (e.g. docstrings), test cases, and examples of inputs and outputs. They have shown remarkable ability at being able to solve complex programming problems at the level of the average human programmer [39]. Prompting for code generation generally differs from traditional prompting interactions, because code has underlying abstract syntactic representations, while natural language prompts can be more declarative and focused on conceptual intent [25]. Generating code often involves intermediate representations such as scratchpads [43], chain-of-thought, and chain-of-code operations [17, 38].

While code generation models have primarily been benchmarked on text-based programming problems (e.g. LeetCode problems), they have also been shown to capably handle visual tasks. ViperGPT demonstrated that a code generation model can be used to compose computer vision and logic module functions into code plans to answer visual queries [50]. HCI systems have shown that code generation models can be integrated within creative workflows to provide interactive assistance [12, 52]. Spellburst demonstrated how LLMs can help end users explore creative coding through natural language prompts and operations that merge Javascript code scripts [12]. BlenderGPT allows users to use prompts to handle 3D tasks such as scene creation, shader generation, and rendering [1]. While other works also build text representations of a visual output (e.g. a 3D scene hierarchy [30] or JSON representation of UI [21]), LogoMotion demonstrates how these text representations can be augmented with canvas understanding to visually ground code generation.

A recent direction within the program synthesis space has been the use of LLMs to inspect and edit the code they generate for program repair (also known as self-refinement or self-debugging) [15]. Traditionally, program repair approaches primarily focus on the text aspect of code (e.g. programming problems [22, 29]). Recent work has shown that screenshots can help self-revise front-end code [49]. Our work further extends program repair into the visual domain by showing how we can isolate visual differences in a layer-wise way that is specific to the layer-based organization of design elements on a canvas. Additionally, we show how visually-grounded program repair can be both automatically and interactively integrated within a user interface.

2.2 Animation Tools for Novices

Many prior works have introduced approaches for creating animation from static assets. Often, these approaches scope around specific artifacts such as character animation, dynamic illustrations, explainer videos, animated unit visualization, and 3D animation [14, 20, 32, 37, 40]. Systems focused on animated storytelling have explored how stories and scripts can be converted into animation. DataParticles [14] takes in stories and outputs animated unit visualizations, animating chunks of story with animation effects that connect text and visuals. Katika [32] takes in scripts and outputs amateur explainer videos, animating shots with motion bundles and SVG assets. While these systems show how animation can be guided by natural language, the mappings they make between language and animation are preset-based (e.g. "pop and pulsing", "highlight") and can be too limited in expressive range for logo animation.

Other HCI systems have explored how other modalities of input like sketch, texture, performance, deformation, and video [19, 20, 31, 34, 35] can drive animation. Draco and Kitty show how sketch input and kinetic textures can create path and particle motion for digital illustration [36]. Motion Amplifiers and Ma et. al. show how presets implementing animation design principles (e.g. squash-and-stretch, follow-through) [33] can help users create stylized 2D and 3D animation [37, 40]. Performance-based systems explored how gestures and acting can create interactive graphic overlays and layered character animation [20, 48]. Recent work such as DrawTalking [46] show how different modalities like sketch and speech can be combined during animation. Mixed-initiative interfaces for animation [27, 53] show that users can benefit from automatic support for tedious tasks such as object segmentation. However, the freeform nature of inputs like sketch, texture, or performance can be underconstrained for a professional use case like logo animation, and many of these prior approaches still primarily work around parameter-based editing. Lastly, other approaches allow users to transfer motion from input videos or source animations [34, 54]. However, these approaches make users start from a separate source of content and work backwards, which can be ill-posed for something as content-specific as logo animation.

2.3 Generative Tools for Animation

Generative models have introduced new approaches for animation. Techniques such as frame interpolation and embedding-based interpolation have provided new techniques and tools for creating stylized animation and morphing [16, 18, 42]. Models like AnimateDiff and motion LoRAs allow users to generate animated videos using prompts and learned motion effects [28]. Tools such as Runway Motion Brush [9] allow users to brush over parts of an image or video to animate. While these aforementioned tools are powerful at creating motion that matches the underlying content, they animate in pixel space and treat image and video as one flat layer, which make them ineffective at handling layered inputs such as layouts. LogoMotion shows how AI tools can handle inputs with layers by

automatically grouping them and letting users interactively edit this organization through layer panel interactions.

A separate line of work looks at generating code for motion design and animation. Spellburst [11] was an early work showing that users can generate creative code (p5.js) and edit by merging code representations. Keyframer [52] was another work that explored how novice and expert designers can generate CSS animation with LLMs and iterate through prompts and direct editing of code. This work also described that users had difficulty controlling the grouping and timing of animation using natural language. Both of these generative approaches operated without visual context–they understood the generated animation only in terms of its text representation (e.g. Keyframer used short text labels to describe the elements being animated). LogoMotion builds on these two works but is the first to leverage VLMs for animation and use visual analysis to inform code implementation. LogoMotion also introduces code-connected AI editing widgets, which let users edit animation at a higher level of abstraction than code and with more targeted control and intuitive GUI affordances than prior approaches [11, 52].

3 Formative Steps

To understand the scope of logo animation, we took a mixed methods approach: 1) interviewing motion designers and 2) analyzing existing end-user tools. We first conducted interviews with experts to inform our approach. Then, we additionally analyzed end user tools to understand the design patterns commercial tools offer novices to animation.

3.1 Formative Methodology

We interviewed motion design professionals (E1-E4), each of whom had at least 10 years of experience in logo and brand animation. E1, E2, E4 were recruited from a freelancing platform; E3 was recruited from within a design company. Each designer was interviewed and compensated for an hour of their time, during which they presented their 2-3 projects from their portfolio and spoke to their design process.

3.2 Formative Insights around Motion Design and Logo Animation

3.2.1 Motion designers strive to make the animation semantically meaningful. Motion designers often begin the animation process by receiving concepts, design briefs, and brand books from clients. These documents often help tie the important branding elements of the logo with semantic context about brand identity. For example, E1 described how clients would ask for them to make the text within the logo animation move playfully or have the graphic elements rain down the screen with Tetris-like game mechanics to create a nostalgic feel.

E4 described how it was especially important that the motion on the primary element match how that element might visually be expected to move. They stated, *"For logos, it is the brand identity. If it's a tree it needs to grow. If it's a wave, it needs to be waving. It has to be something specific to the logo."* They described animated templates as a design solution they know clients would try but added context about the limitations to templates: *"It's hard to generalize it [refers to a logo animation template]. You can just swap the logo in and out, but it has nothing specific to it.*

3.2.2 The design of logo animation is influenced by visual qualities such as layout, symmetry, layer relationships. E2 and E4 both showed animated logo reveals from their portfolios that had a *"zero-to-hero"* effect. In these, the primary element, the logo, often gets a hero moment, while the rest of the secondary elements would enter this hero moment with supporting animation. In this way, the *visual hierarchy* of a layout is a source of information that can inform the amount of motion applied on each element. Other sources of information include *positioning* and *grouping*. E1 showed a project where they animated a dotted border by staggering a fade-in effect across all the dots. Creating groups and applying animations to groups creates a sense of coordination and visual flow. E2 also showed a project where they animated two halves of an infinity logo to move symmetrically into place, showing how animation can reflect visual relationships like symmetry.

3.2.3 Professionals focus on visual flow and the technical details of motion dynamics. Professionals like E2 stated that they spent most of their time getting the visual flow right. This meant experimenting with different timings and easing functions. It can be hard even for an expert to get these aspects of motion dynamics to feel right, suggesting that this could be a tedious aspect of animation that would benefit from automatic support. E2 showed timelines they worked over in After Effects, which had dozens of keyframes that they kept track of by repeatedly playing back the animation as they created it.

3.3 Analysis of End User Tools for Layout Animation

To complement our expert interviews, we analyzed commercial tools novices use for animation. These tools included Adobe Express, Canva, Google Slides, Capcut, and Pinterest Shuffles [2–6]. In these tools, there are often animation panels with motion presets, design galleries of animated templates, and automatic animation techniques. Motion presets are familiar effects like fade, wipe, and dissolve that novices can edit in terms of duration, speed, and direction. In tools like Adobe Express, Canva, and Google Slides, a direction parameter often sets simple path motion like an element sliding in from left on entrance or exit. Some tools such as Powerpoint and Canvas allow users to define path motion with sketch input or preset paths. In Adobe Express, Canva, and Powerpoint, tools generally provide users with 1-2 dozen presets, often organized by motion labels like "energetic" or "professional". These tools often restrict novices to working within a few states such as on entrance, on main duration, looping, or exit [10]. Tools also offer galleries of animated templates, which have prebaked motion and placeholders users can swap out with their own content [5]. Lastly, tools such as Figma and Canva offer automatic animation techniques, so that novices do not have to individually animate each element. Figma Smart Animate and Powerpoint Morph find matching layers across slides and use property differences to automatically set animation [7, 8]. Canva Magic Animate animates layouts based on an input motion style and a computational understanding of the canvas layers [6].

From these formative steps, we derived the following guiding design principles:

DG1) *To create semantically meaningful animation, an AI approach should generate code that users can easily edit and engage with in natural language.* Users can benefit from having a design concept that guides the code implementation, which they can edit later on.

DG2) *To create animation customized for the design, an AI approach should apply visual analysis to guide the code implementation.* Users should be able to edit how an AI organizes the animation in terms of which elements get more emphasis and how elements group within the animation.

DG3) *To help novices edit their animation, an AI approach for animation should help visualize the animation and in what ways it can be edited.* Users should be able to understand the events that make up their animation and easily identify aspects that they can edit such as timing, emphasis, and speed.

DG4) *To prevent novices from having to understand the technical details of the code, an AI approach for animation should help users handle the implementation details.* Users should be able to edit with high-level controls in the form of GUI support that allows them to make targeted edits to their animation.

4 LogoMotion System

We present LogoMotion, an AI code generation and editing tool that helps novices create semantically meaningful animation. We walkthrough LogoMotion's approach with a Ski Club logo as a grounding example. First, LogoMotion helps novices automatically animate the logo by generating code using **visually-grounded code synthesis and program repair**. This method takes in a logo in the form of a layered PDF document as input and generates an HTML page and animation code as output. The animation that executes on the page is semantically meaningful to the logo design. For example, a skier element can ski in from the left and do a flip, and the title text can pulse in just as the skier lands. Second, novices can customize this animation using **code-connected AI editing widgets** we introduce such as a narrative timeline, layer panel, and quick actions. These code-connected AI editing widgets implement edits to the motion, grouping, and timing of the animation by doing targeted regeneration of the underlying code. For example, a user can edit the skier to have a different hero moment (e.g. skier skis in diagonally as if on a slope) or synchronize the timing of all secondary elements with the skier's entrance. We now walk through how users can create and edit animation with LogoMotion.

4.1 Input

As input, LogoMotion takes in a PDF that can consist of multiple image and text layers. Fig. 2 shows an example. The logo has a skier, two mountains, decorative lines, a colored background for image elements, and "CARDINALS" and "Ski Club" for text elements. This PDF is a layered design that has already been arranged by a designer, so each element has a pre-defined position and a layer index. LogoMotion starts with all layers not attached to any groups.

4.1.1 Preprocessing. To create a code representation of the canvas, LogoMotion uses a custom-written Extendscript program that takes a PDF imported in Adobe Illustrator and exports it into an HTML page. The Extendscript iterates through the text and image layers and converts each one into an HTML element with a unique ID ("#EHP"). It copies over bounding box information and z-order indices into the style attribute of each element. Fig. 2 shows how the skier image and text elements are exported as HTML. For text elements, LogoMotion also copies over the text content in alt text (alt="SKI CLUB") and sets a style attribute (class="text"). This HTML representation helps LogoMotion understand how to target image and text layers in the animation code.

4.2 Visually-Grounded Code Synthesis

During visually-grounded code synthesis, LogoMotion performs visual analysis to understand the semantics of the logo and instantiates a design concept to inform the implementation of animation code.

4.2.1 Layer Captioning. To understand what each element represents, LogoMotion first captions each layer. Fig. 2-Step 1 shows how a skier image is captioned as an "icon of person skiing". To caption, LogoMotion isolates each layer against a plain background, queries a VLM for a caption, and sets the caption to be the alt text of the HTML element.

4.2.2 Visual Hierarchy. LogoMotion analyzes the logo image to understand the visual hierarchy of the elements within it. This visual analysis can define which elements should get more emphasis (the hero moment in a logo) and which elements should get less (secondary supporting motion). LogoMotion passes the HTML code (which holds information about the element's caption, sizing, positioning, layering) and the logo image through a VLM call that outputs a JSON classifying each HTML element as primary, secondary, text, or background. These role classifications are added to each HTML element as a class attribute (e.g. class="primary"). LogoMotion chooses only one element as the primary element and applies additional visual analysis to understand which way it faces and how the element should enter. For example, if the skier element appears to face the right side of the canvas, it should ski and flip in from offscreen left (Fig. 2-Step 2).

4.2.3 Grouping Elements. In an animation, elements that conceptually relate ideally move in a coordinated way. To support this, LogoMotion automatically makes groups over the secondary elements and applies animations to groups. It does this by taking in the HTML and logo image and outputting a JSON that places layers in groups (e.g. secondary-group-1: #EHY, #OWE). Elements in the same group are then reorganized in the HTML to be under the same parent div element (e.g.). In Fig. 2-Step 2, we show how LogoMotion groups the two mountain elements, two line elements, and two text elements together. There can be any number of groups and grouping is especially effective a layout has many secondary elements, such as a border of repeating circles or a dozen stars.

4.2.4 Design Concept. With an HTML representation of the logo that is augmented with visual analysis, we are ready to animate. To create a semantically meaningful animation, LogoMotion instantiates an animation *design concept*. The design concept is a

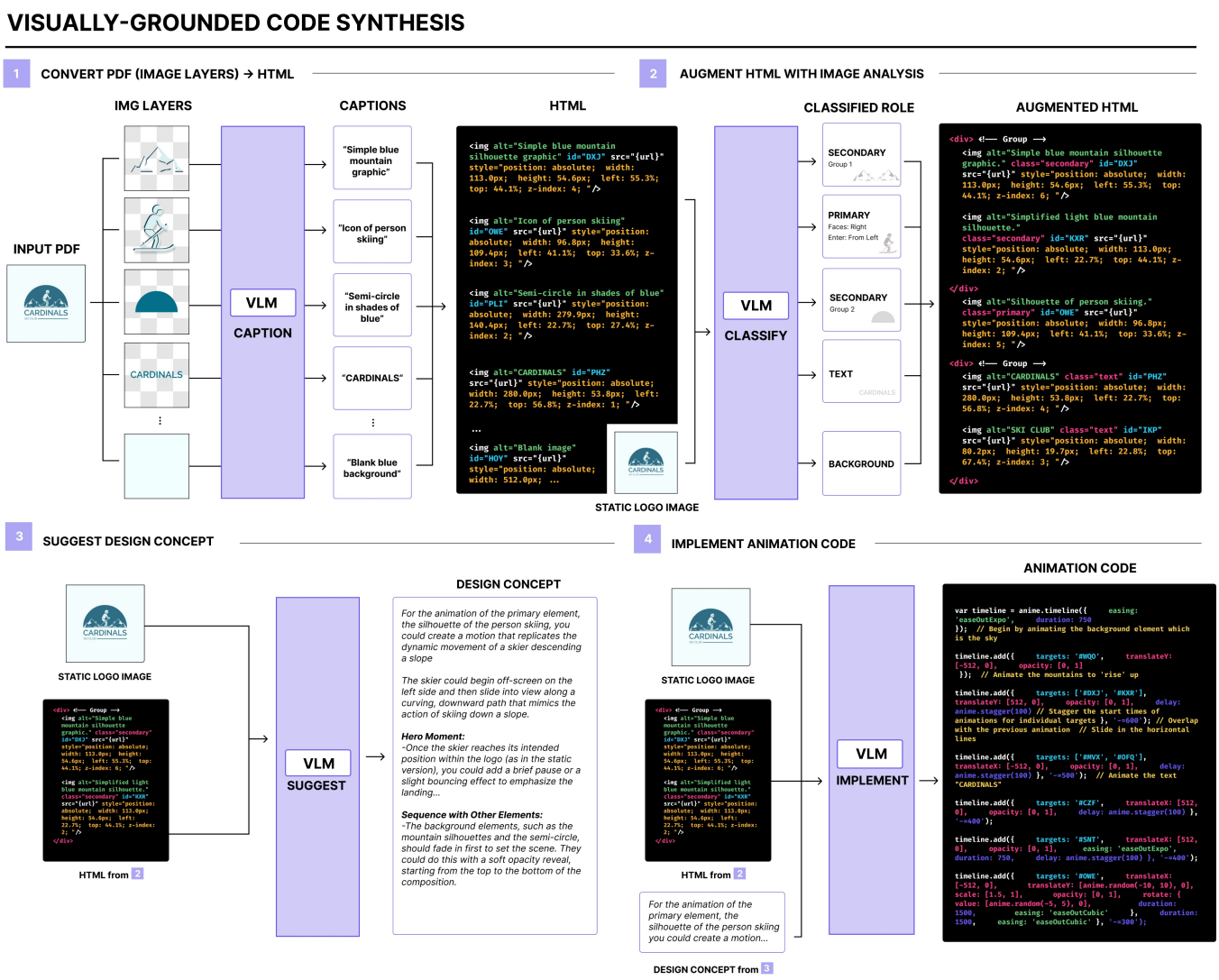


Figure 2: Visually-Grounded Code Synthesis Overview. In Step 1, a PDF of a logo is converted into an HTML representation of the canvas. In Step 2, LogoMotion augments the HTML with information from visual analysis steps. In Step 3, LogoMotion suggests a design concept for the animated logo. In Step 4, a VLM implements animation code for the design concept. This code animates the logo HTML.

pseudocode that semantically conditions the code generation. It suggests how each group of elements should animate, what hero moment can be applied to the primary element, and how all the elements sequence and time in. It can be multiple paragraphs, as shown in Fig. 2-Step 3: *"The skier could begin off-screen and slide into view along a curving, downward path that mimics the action of skiing down a slope. Once the skier reaches its intended position, add a brief pause or slight bouncing effect to emphasize the landing."* To generate the design concept, we provided a VLM with 1) the HTML from the previous step and 2) logo image. This is an example of our prompt:

```
This image is of a logo that we would like to animate.
Here is the HTML representation of this logo: <HTML>
We want to implement a logo animation which has a hero
    moment on the primary element. The primary element
    in this caption should animate in a way that mimics
    its typical behavior or actions in the real world.
    We analyzed the image to decide if it in its
    entrance it should take a path onto the screen or
    not: <entrance description>. Considering this
    information, suggest a motion that characterizes how
     this element ({primary element image caption})
    could move while onscreen. Additionally, suggest how
     this element should be sequenced in the context of
    a logo reveal with the other secondary, text, and
    background elements. (Note that the element is an
    image layer, so parts within it cannot be animated.)
```

4.2.5 Code Implementation. LogoMotion then implements the animation code using the 1) HTML that has been informed by visual analysis, 2) design concept, and 3) logo image. The prompt instructs the VLM (GPT-4o) to initialize a variable for a timeline and write animation events that target groups (HTML classes) and elements (HTML IDs). The generated code is then run on the HTML page, which includes anime.js as a library and presents all the layers on a canvas. LogoMotion can also be adapted to write in other animation frameworks (e.g. CSS animation). An example of how LogoMotion translates a description from the design concept into animation code is provided below.

```
timeline.add({
    targets: '#OWE', \\ Skier enters from the left, does
     a spin trick in midair
    translateX: [-512, 0], \\ Moves across from the left
    translateY: [0, -100, 0], \\ Jumping effect
    rotate: ['0deg', '360deg'] \\ Spin trick
    easing: 'easeInOutSine', \\ Easing for smooth
     animation
    duration: 1000, startTime: 3000}
```

4.3 Visually-Grounded Program Repair

AI-generated code can have errors and animation code in particular can have errors that render visually. For example, generated animation code can leave the skier element twenty pixels off from the intended position, because of an incorrect assumption AI makes about array syntax. To catch and fix these errors, LogoMotion use visual feedback from the canvas for self-debugging in a method we call **visually-grounded program repair**.

Once the animation code from the previous code synthesis stage is generated, it is automatically executed on the HTML logo. After the code is run, LogoMotion checks the last frame of the animation. It steps through each layer and programmatically compares the position, scale, rotation, and opacity for what it should be in the original logo layout. It tests for equivalence on attributes from the bounding box coordinates (left, top, width, height) and style properties (opacity). Because VLMs often make mistakes when provided unnecessary visual context [13, 24], LogoMotion goes layer by layer to reduce the information that the VLM needs to process.

Once an error is detected, we pass in the element ID(s), a bug description, animation code, and a pair of images capturing the difference. This image pair shows the element on its own–the rest of the layers are hidden before LogoMotion takes a screenshot. The images are labeled by "TARGET FRAME" and "LAST FRAME". Fig. 3 shows an example of a bug that renders visually: the skier is off position. LogoMotion then generates a code snippet to fix it and re-executes the animation with the proposed fix. If the animation error is fixed, the snippet is merged back in with a VLM call. We allow LogoMotion to try fixing up to four times. Though this step catches important errors that violate the original layout of the logo, it does not check for all possible errors (e.g. layer occlusions in intermediate frames). With the conclusion of checking, LogoMotion completes the automatic stages that generate a logo animation.

4.4 Editing with Code-Connected AI Editing Widgets

LogoMotion allows novice designers to edit their animation using **code-connected AI editing widgets** within the UI pictured in Fig. 4. The UI includes a canvas for viewing the animation, an Exploration Tree that tracks history, and the following code-connected AI editing widgets: 1) a narrative timeline, 2) a layer panel, 3) and a quick action panel. We call these widgets *code-connected* because when users make edits with them, LogoMotion implements the edit by regenerating part or all of the underlying animation code. We enable this connection between the editing widgets and the AI-generated animation code by processing the HTML code representation into the groups of the layer panel (Fig. 4-D) and the Javascript animation code representation into the narrative timeline (Fig. 4-C). The intuitive affordances of a GUI representation helps users edit the motion, grouping, and timing of the animation at a higher level of abstraction over the underlying code.

4.4.1 Narrative Timeline. The most prominent widget is the narrative timeline. It "narrates" what elements are doing during the animation. The timeline visualizes four tracks for Primary, Secondary, Text, and Background that populate with animation blocks. Each block is labeled with a description of the animation events that take place within the span of the block. For example, a block mapping to a group of mountains on the Secondary Track can have the label *"mountains rise in"*. The narrative timeline allows users to read the design concept that was generated in natural language rather than having to understand code. These labels can also be edited by users to change the animation events within that block. To create the narrative timeline, LogoMotion extracts the code comments attached to each block of animation code and uses them as labels. LogoMotion also defines the width and position of the block on the timeline by finding the start and end timestamps of each element and doing a merge-and-sort over all elements in a group. Each animation block captures metadata about the code, such as what code block it maps to, what elements it targets, and what its timestamps are.

Users can write prompts to edit the animation, and these prompts can be bound to selected animation blocks. For example, Fig. 4-E shows how a user can prompt *"can you make the skier do a front somersault..."* and select the skier block as shown in Fig. 4-E. Clicking *Edit Animation* triggers LogoMotion to regenerate the code, specifically targeting that block. Users can also select multiple blocks at once and write a prompt. For example, they can write *"make them all fall in like snow"* and have the edit apply across multiple selected groups. Lastly, the narrative timeline supports drag-and-drop, so users can edit start and end timestamps. Drag-and-drop requests are also routed to the VLM to implement, as exact start and end timestamps can be dependent upon other parameters like offsets ("+=200ms") and are not as simple to replace programmatically. For every prompt edit, LogoMotion uses GPT-4o and takes in the following inputs: block metadata, the animation code, HTML, and the logo image.

4.4.2 Layer Panel to Reorganize Animation. LogoMotion automatically decides groups during the code synthesis, but users should be able to edit these groups. LogoMotion lets users do so with a layer

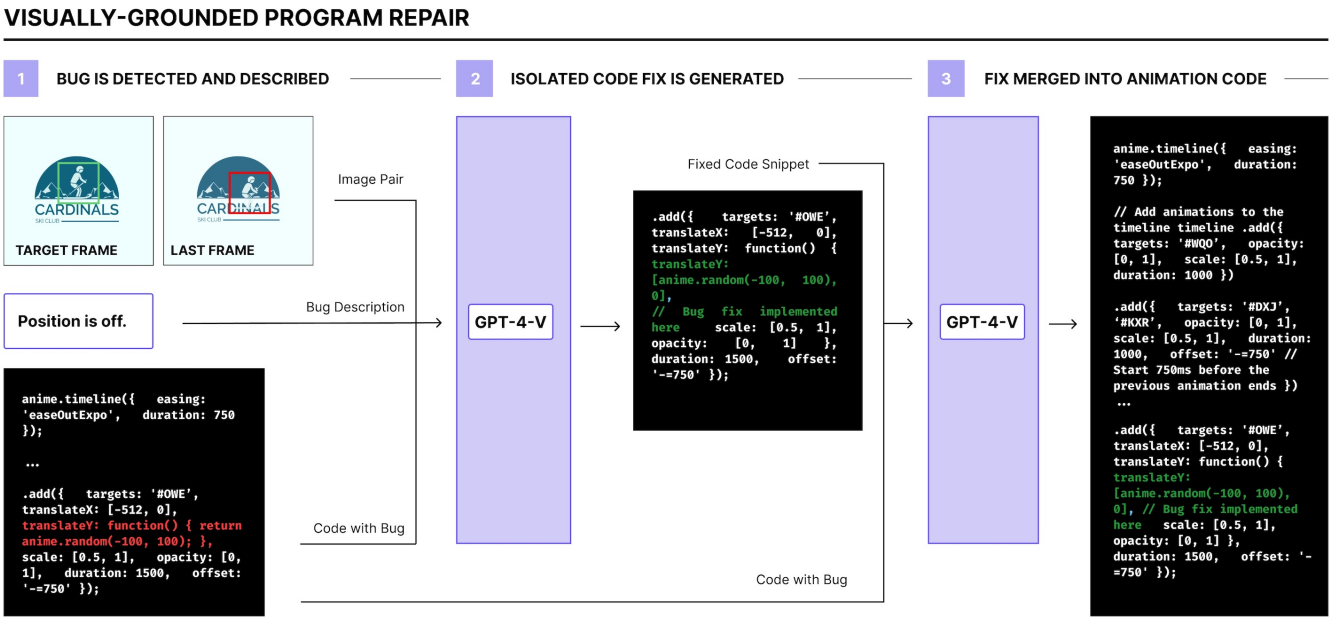


Figure 3: Visually-grounded program repair takes visual feedback from the canvas to self-debug animation code errors. It identifies bugs in animation code by checking for differences between elements in the target layout and elements in the last frame of the animation. If there is a visual error, a VLM receives 1) an image pair of the element with an error (the rest of the layout behind the skier element is pictured only for context), 2) a bug description, and 3) the original code. It outputs a code fix that is merged back into the animation code.

panel (Fig. 4-D) that reflects the groups of the underlying HTML representation. Users could rename layers to have new identifiers, rearrange layers to be in new groups, create a new group, or delete groups. Clicking *Regroup* would trigger LogoMotion to update the HTML code representation by calling a VLM to regenerate the animation code and reorganize it around the new groups. For example, at first, a user could have mountains, lines, and the semicircle group all doing different animations. The user could put them all within the same group and request a *Regroup* which would regenerate the code to make all elements in the same group synchronized to have the same timing and motion.

4.4.3 Quick Actions. To provide users with high-level controls and help users identify ways they can edit the animation, LogoMotion provides quick actions: *Subtle, Emphasize, Synchronize Timing, Slower, Faster* and *Delete* (Fig. 4-B). *Slower / Faster* helps users test different speeds, and *Subtle / Emphasize* helps users try different amounts of emphasis. *Synchronize Timing* allows users to synchronize animations to align with the same timing. *Delete* helps users identify and delete events within the animation. We derived this set of operations through experimentation and pilots with users, finding these to be the commonly requested actions. Quick actions can apply to selected blocks or to the entire animation timeline if no blocks are selected.

4.4.4 Interactive Version of Program Repair. Every time the code is regenerated, LogoMotion automatically runs program repair. In pilots, we found that users sometimes did not want to wait for program repair to complete, because it could take a few attempts. They wanted to be able to stop it when it was unnecessary, so we allowed them to exit out of the loop. We also limited the checking to the primary element. As an alternative, we provided a *Fix Issues* quick action to allow users to trigger visually-grounded program repair whenever they wanted.

4.4.5 Exploration Tree. To help users track version history, LogoMotion provides an Exploration Tree (Fig. 4-A). Users can choose one of four of LogoMotion's automatic animations and start editing. Each edit made becomes a new node on the Exploration Tree, and each node is labeled with what the edit was about (e.g. *"add oscillation to skier entrance"*). Each time a user clicks on a node, LogoMotion updates the narrative timeline with the animation code and the layer panel with the layer and groups at that version.

4.4.6 Code-Connected Widget Implementation Details. The system was implemented in Python / Flask and React/ Javascript. For VLMs, LogoMotion calls GPT-4v (for Evaluation 1 and 2) and GPT-4o (for Evaluation 3). LogoMotion stores the history of animation runs and repair history behind each run in JSON files. To implement visually-grounded program repair, LogoMotion copies each element over to a duplicate canvas that is hidden from the user, so that it can programmatically check layers and screenshot them without the user seeing any visual side effects on their canvas. LogoMotion checks the style attributes on each layer with a 1 percent tolerance level on pixel differences, so that program repair does not overtrigger on minute differences.

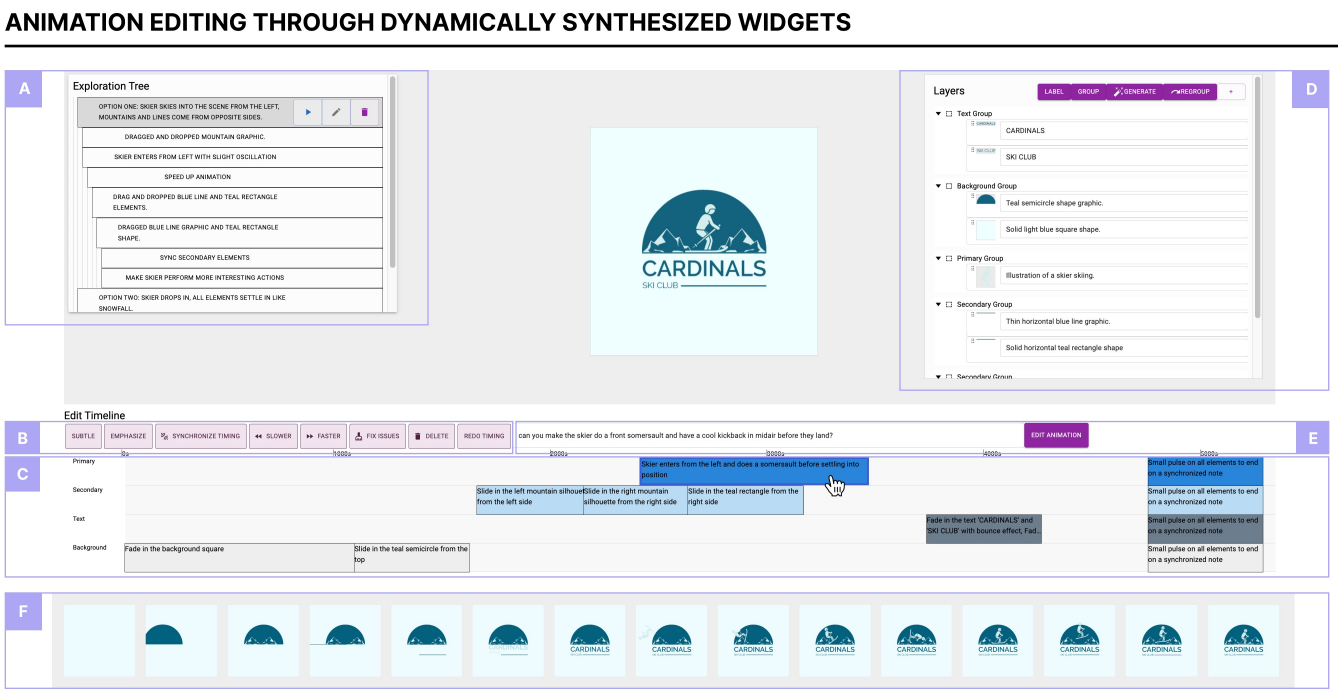


Figure 4: LogoMotion provides users code-connected widgets to edit their animation. It provides quick actions (B), a narrative timeline (C), a layer panel (D), and prompt-based interactions (E). These widgets help users make targeted edits to the motion, grouping, and timing of their animation. An example animation edit output is shown in (F).

Through this approach, LogoMotion allows users to automatically generate semantically meaningful animation using **visually-grounded code-synthesis and program repair** and interactively customize the animation using **code-connected AI editing widgets**.

5 Comparison Study on Automatic Outputs

We conducted three evaluations to understand the quality of LogoMotion: 1) a comparison study against an industry standard informed by professional logo animators 2) an empirical analysis of program repair testing different experimental settings, 3) an evaluation with novices (n=16) to understand LogoMotion's support for animation editing. These evaluations centered around the following research questions:

RQ1: To what extent does LogoMotion generate animation that is relevant to design elements on a canvas?
RQ2: What are the overall strengths and the weaknesses of LogoMotion at animation?
RQ3: What sorts of errors does LogoMotion tend to make in code?
RQ4: How capably can visually-grounded program repair debug errors and what settings of program repair impact performance?
RQ5: To what extent can LogoMotion's code-connected AI editing widgets help novices with animation editing?

The first evaluation is a comparison study comparing LogoMotion with Canva Magic Animate [6]. Magic Animate, is an AI-based tool for automatic animation that is one of Canva Pro's premium features. Magic Animate analyzes the design on a canvas and applies motion to match the content, font choices, images, and color. It provides a range of motion styles ("bold", "professional", "elegant") which can change the dynamics of the animation and cover a wide variety of use cases. We additionally created an ablated version of LogoMotion (LogoMotion-Ablated) to understand how higher-level VLM operators impacted the content-awareness of the approach. Specifically, we ablated Step 2 (visual analysis resulting in the Augmented HTML) and Step 3 (Design Concept) of our visually-grounded code synthesis stage. From experimentation, we found that some base amount of visual grounding (Step 1 - captions and layer information) was necessary for the approach to work, so we wanted to understand the effect of more versus less visual grounding. Our hypotheses for RQ1 were the following.

H1a: Compared to the other conditions, LogoMotion-Full would produce animations that were more content-aware.
H1b: Compared to the other conditions, LogoMotion-Full would be improved in terms of sequencing.
H1c: Compared to the other conditions, LogoMotion-Full would be better in terms of execution quality.

5.0.1 Methodology. We gathered a test set of 23 logo layouts that spanned different categories of objects (animate and inanimate), layouts, and use cases. Use cases included holiday greetings, school clubs, advertisements, and corporate branding. All logo layouts

were sourced from Adobe Express and Canva and are accessible online.

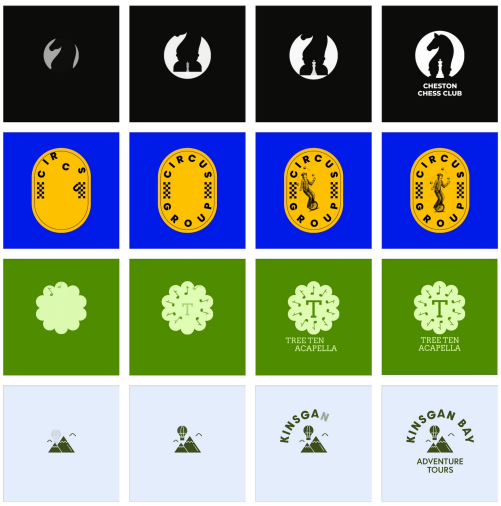
To understand the quality of the automatically generated animations, we ran the first two stages of LogoMotion to get four animations for each logo. We then ran LogoMotion-Ablated to get another four animations for each logo. Generating a set of four LogoMotion/ LogoMotion-Ablated outputs took approximately 12 minutes. To gather four animations for Magic Animate, we took the motion style they recommended for the layout as one animation and had an external logo animator pick the three next-best motion styles. The combination of a professional animator's decision on top of the fact that Magic Animate is an industry tool (engineered to have a polished outcome and wide applicability to different kinds of content) ensured we had a strong baseline to compare against.

To evaluate the animations, three professional designers were recruited to rate 276 animations (23 templates x 12 animations per logo) spanning the three conditions. Designers were introduced to the task with a remote call, calibrated with good and bad examples to understand the rubric, and compensated for their time. Each animation was presented in randomized order and rated on a scale of 1-5 for each of the following dimensions: 1) Relevance, 2) Sequencing, and 3) Execution Quality. Relevance describes how relevant the animation is to the subject matter of the logo. It is a measure of how specific the animation is to the design elements of the logo. Sequencing was a measure of how well the animation was sequenced in terms of coordination and timing across elements. Execution Quality judged the animation for how well it was executed and if it had any flaws. Each designer rated all 276 animations and took on average 2.83 hours to complete the task.

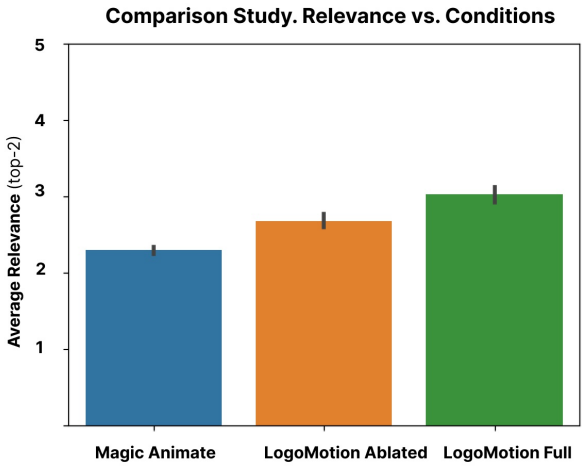
5.0.2 H1a. Relevance . We averaged across the ratings of three design professionals. LogoMotion was rated to have significantly more relevance to the subject matter of the animated logos than both Magic Animate and LogoMotion-Ablated (H1a, LogoMotion-Full: M = 3.05, σ = 0.64; LogoMotion-Ablated: M = 2.68, σ = 0.58; Magic Animate: M = 2.33 , σ = 0.33, $p \leq 0.001$). From this we confirm H1a; LogoMotion-Full was the top condition in terms of content-aware animations.

When sorted by average relevance across raters, the top rated animations tended to come from LogoMotion (15 of top 20) or LogoMotion-Ablated (5 of top 20). LogoMotion animations that were rated highly showed semantically meaningful motion. The video for this paper shows examples of lanterns blowing as if in slight wind, crabs crawling zigzag into the screen, and a hockey stick swung as if in play. Frames from animations are depicted in Figure 5a. In the first row, a black knight is translated into the canvas in an L-like motion as a bishop piece scales up. In the last row, we see a hot air balloon slowly rise in over the mountains, after which the logo title fades in letter by letter.

5.0.3 H1b. Sequencing . In terms of Sequencing, LogoMotion-Full was not significantly different from the other two conditions (H1b, LogoMotion-Full: M = 3.15, σ = 0.55; LogoMotion-Ablated: M = 3.18, σ = 0.41 ; Magic Animate: M = 3.12, σ = 0.43). Qualitatively, LogoMotion-Full and LogoMotion-Ablated were both capable of implementing the logical sequencing of a logo reveal. They could time primary elements before secondary elements or vice versa, but generally put the text last. LogoMotion generally sequenced layers



(a)



(b)

Figure 5: Comparison Study. a) Examples of LogoMotion-generated logo animations. These animations show how LogoMotion is able to create motion that is characteristic of the design elements, layout-aware, and logically sequenced. b) We report the average top-2 relevance across conditions. LogoMotion was rated to be significantly better in terms of relevance to canvas content.

in from bottom to top. Animations rated lower for Sequencing tended to have errors in layer ordering. Background elements were often left static but could also be used effectively in animations. In the paper's video figure, we show that in an animated logo for a martial arts club, a silhouette of a karate character pops in from the left. As it lands into place, the background shakes, as if from the impact of a karate kick. The interaction between these design elements was suggested by the design concept: *"for added impact, consider a screen shake or a vibration effect when the silhouette 'lands' to draw even more attention to the primary element's hero moment."*

	Relevance	Sequencing	Execution Quality
LogoMotion Full	**3.05****	3.15	3.25
LogoMotion Ablated	2.68	3.18	3.38
Magic Animate	2.33	3.12	3.22

Table 1: Relevance, Sequencing, Execution Quality. Average expert designer ratings evaluating automatically animated logos (n=276) for three conditions. ($^{**} = p \leq 0.001$)

LogoMotion was also capable of generating animations that reflected properties of gestalt like symmetry. For example, the symmetrical elements of the Circus example in Fig. 5a) animated in together, and the musical notes in the Acapella example animated in a contralateral, symmetrical way. LogoMotion could also create staggering within animation by mapping a generated animation function to a group of elements. This was how it implemented text animation such as a typewriter effect for arced text (Fig. Figure 5a-row 4) and could stagger an opacity animation across a group of stars to create a twinkling effect.

5.0.4 H1c. Execution Quality . LogoMotion-Full did not perform significantly differently from other conditions in terms of execution quality (H1b, LogoMotion-Full: M = 3.25, σ = 0.54; LogoMotion-Ablated: M = 3.38, σ = 0.46; Magic Animate: M = 3.22, σ = 0.39). LogoMotion-Ablated scored the highest on execution quality. Many animations for LogoMotion-Ablated tended to be conceptually similar (all elements fade in or translate into place from a slight displacement). Animations for LogoMotion-Ablated tended to be minimal in complexity and thus easy to execute. What brought LogoMotion-Full down in terms of execution quality was that sometimes LogoMotion-Full animated box-shadows and background-color changes, which made it past our program repair checker and were left unrevised. Lastly, for context, Magic Animate could at times receive lower scores on execution quality, because some of the animations it assigned produced a cropping effect that was an undesirable visual side effect.

5.0.5 Comparison Study Conclusion. This comparison study illustrated that LogoMotion could outperform another AI-based industry standard animation tool at producing animation specific to the design elements on the canvas. LogoMotion's animations were comparable in terms of execution quality and logical sequencing.

6 Technical Evaluation of Program Repair

Next, we conducted an evaluation for visually-grounded program repair stage to empirically understand what errors LogoMotion tended to make (RQ3) and how capable it was at self-debugging them (RQ4).

6.1 Methodology

Sixty-eight percent of the animation runs from the outset (after the first stage of code synthesis) were error-free and did not require program repair. The other 32% required the program repair. Within this stage, we modulated a 1) hyperparameter k and 2) whether or not image context was provided. k upper bounded the number of attempts a VLM could take to solve the bug and was varied from 1 to 4 attempts. Varying k is modeled after the pass@k methodology proposed by HumanEval [22], where k code samples are generated in attempts to solve a problem and the fraction of problems solved is the solve rate. In this case, the pass@k framework is applied in the context of program repair.

The second setting that we varied was whether or not image context about the visual error was provided. This image context (pictured in Fig. 3) is a labeled image pair showing a layer at its target position vs. in its last frame in the animation. We refer to Repair$_{+Imgs}$ as the setting with both visual context and bounding box information about the error. We refer to Repair$_{-Imgs}$ as the setting where no visual context and only bounding box information is provided.

For each setting, program repair was run k={1,2,3,4} times on animation code that had errors. Two animation code samples had to be excluded due to compilation errors that did not allow the program repair stage to complete. Overall, 112 samples of self-debugged animation code were generated for the Repair$_{+Imgs}$ condition, and 112 samples were generated for the Repair$_{-Imgs}$ condition.

6.2 Findings

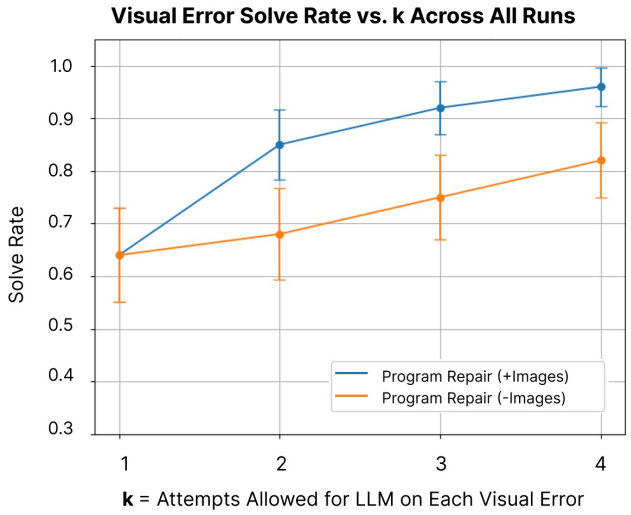


Figure 6: When LogoMotion is given more attempts to debug each error (increase in k), LogoMotion improves in solve rate. The trend is higher when image context is provided ($Repair_{+Imgs}$). Bars represent standard error.

6.2.1 RQ3. What errors does LogoMotion synthesis make? LogoMotion made 42 position-based errors in total. Position errors, which occurred when the left or top coordinate of the bounding box was off, were made in 30.4% of the runs. LogoMotion made 26 scale-based errors in total, erroring in 18.4% of the runs. These errors occurred when the width or height dimensions were off. We did not detect any opacity errors in our test set.

Next, we provide qualitative context as to how LogoMotion would make these errors. Common errors resulted from not following the from-to format that is common to animation libraries (CSS

Figure 7: Examples of animation code errors that LogoMotion would resolve with program repair: a position error (top left), rotation error (top right), scale errors (bottom row).

k	Solve Rate$_{+I}$	Solve Rate$_{-I}$
1	0.64	0.64
2	0.85	0.68
3	0.92	0.75
4	0.96	0.82

Table 2: Table reporting how solve rate changes when k increases across two settings. $SolveRate_{+I}$/$SolveRate_{-I}$ refers to the number of runs that were error-free after k attempts were made for each error $Repair_{+Imgs}$/ $Repair_{-Imgs}$ respectively.

and anime.js). In spite of the prompt suggesting a from-to format, keyframes would often be suggested with arrays that had over two values, so the element would not return back to its original position. For example, if the generated animation set the translateX values [10,-10, 0]–the element would end with a -10 offset relative to its correct position. Examples of errors LogoMotion could make are depicted in Figure **??**.

Position errors could also occur when there was an inconsistent application of absolute and relative percentages. For example, a line layer in an animation could be instructed to stretch in from 0% outwards to 100%. This 100 percent was intended to be with respect to the element but actually rendered with respect to the full canvas dimensions. Another frequent error was when the VLM assigned a looping animation, which would instantiate a small periodic action with the loop parameter set to true. Looping animation events generally left the elements at small deltas from their intended positions but were easily resolved.

6.2.2 RQ4. How well can LogoMotion fix its errors? Many errors were simple enough that they would only take only one attempt from LogoMotion to solve. Figure 6 shows that as k increases, so too does the solve rate on the visual bugs for both $Repair_{+Imgs}$ and $Repair_{-Imgs}$ By $Repair_{+Imgs}$ at k=4, 96% of the runs are resolved. There is a boost in performance from passing in visual context. Applying a Fisher's exact test (considering the small sample size of debug runs), we created a contingency table, where one variable was the presence of image context and another was success or failure at debugging, aggregated across k. We found that the odds ratio was 1.97 ($p = 0.08$), suggesting that the odds of successfully fixing errors with visual context nearly doubles the odds of fixing errors without visual context.

6.2.3 Visually-Grounded Program Repair Conclusion. Our analysis suggests that LogoMotion can effectively make use of the visual context it receives during program repair to self-debug. Providing the model with more attempts can improve the solve rate of LogoMotion on visual errors.

7 User Study with Novices

We conducted a user study to address the following research question: *RQ5) To what extent can LogoMotion's code-connected widgets help novices with animation editing?*

7.0.1 Participants. We recruited participants through mailing lists at two universities, online communities for design tools, and from within a design company. Our target population were users who have design backgrounds and an interest in animation, but not necessarily the interest to engage with professional-grade tools. In total, sixteen participants were recruited (n=16, 6 female, 9 male, 1 non-binary, age 20-32). Fourteen participants characterized themselves as novices to animation, while two participants characterized themselves as people who had dabbled in animation as a hobby before (tried tutorials but did not master the software). Of these sixteen participants, seven had some prior experience with design. The study took 60-75 minutes for participants to complete, and participants were compensated $20 for their time. Prior to the study, participants were sent an IRB information sheet about the experiment and a consent form.

7.1 Experimental Design

We conducted a within-subjects experiment and created an ablated version of our interface to use as a baseline. The baseline presented the user with only natural language interaction through prompt. We justify this as our baseline for the following reasons: 1) a prompt-only interface is one of the norms for generative interaction, 2) this baseline still had access to the same intermediate representations of the canvas (augmented HTML, generated code, logo image), 3) timelines can add overhead for novices compared to the simplicity of a prompt box, 4) LogoMotion's interface affordances can raise expectations even though LogoMotion's code-connected widgets do not behave as deterministically as traditional GUI.

Participants were given a brief introduction to the logo animation. They were instructed that their task was to animate two logo layouts given LogoMotion and the baseline. Prior to each task, an experimenter gave them a walkthrough of each interface with an

example logo. The experimenter demonstrated the prompt box, narrative timeline, quick actions, and layer regrouping. To minimize for ordering and learning effects, the ordering of the conditions was counterbalanced and randomized across participants.

Participants picked a different layout to work with between tasks, selecting from a set of options. In total, participants animated 24 unique templates (Figure 8) spanning a wide range of design elements, use cases, and layouts. To focus on the editing aspect, four logo animations for each layout were pre-generated–but not cherrypicked. Participants had 25 minutes to customize one of the four logo animations, though they could stop whenever they felt done. During the task, participants were instructed to think aloud. Between tasks, participants filled out a brief survey with Likert-scale questions. After the tasks, a semi-structured interview was conducted to understand participant experiences. Usage logs additionally captured interaction traces with both interfaces.

7.2 Results

Overall, many participants enjoyed working with LogoMotion, found the system highly performant, and saw its potential for animation.

7.2.1 Ability. Fourteen of 16 participants using LogoMotion and 13 of 16 participants using the baseline agreed (≥ 5 out of a 7-point Likert scale) that the system helped them create animations they would have otherwise not been able to create. P2 said, *"It's really cool especially for someone who couldn't do it without this tool and wouldn't spend a lot of time with Youtube videos or tutorials to do the very basics."* P16 said, *"I really liked it, I enjoyed it. It was really fun. I wouldn't be able to make something to this perfection if I were to make it from scratch using something like After Effects–it would have taken two hours to do it."*

Participants spent more time editing and iterating with LogoMotion than with the baseline. On average, participants spent 8.53 minutes with the baseline and 13.15 minutes with LogoMotion, a difference that was significant ($p = 0.017$). Participants often stopped earlier in the baseline than with LogoMotion because they ran out of ideas for how to edit the animation (P9), had difficulty coming up with the vocabulary to describe the animation they wanted (P1, P6, P8, P13), and because they said further iteration on timing would be frustrating (P4).

7.2.2 Exploration. One aspect participants liked in particular was LogoMotion's ability to provide multiple options for inspiration. P4 was animating a Casino logo (pictured in Fig. 8) when they saw in the narrative timeline that one option animated the primary element to have a card flipping motion. *"I like that I could see the timeline, so when I look at different options I could see what was happening under the hood. I figured out that I could use the keyword flip because some other one [animation] used the word flip."* The narrative timeline gave novices understanding about how the system was arriving at the animation concepts and and helped users ideate new ways to animate.

The version history captured by the Exploration Tree allowed us to measure exploration and iteration. For each participant, we measured the number of edits made and depth of iteration (longest path from base animation to last edited animation). Participants explored significantly more animations with LogoMotion compared to the baseline, averaging 11.19 animations with LogoMotion versus 7.13 animations with the baseline ($p \leq 0.01$). Participants were also able to carry out more rounds of iteration with LogoMotion, creating deeper chains of edits. On average, the maximum depth of iteration in the full condition was 6.56 animation edits in length compared to 4.75 animation edits in length, a difference that was statistically significant ($p \leq 0.05$). We visualize the distribution of animations explored and iteration depth across participants in Fig. 9.

Figure 8: Examples of logos participants animated spanning a diverse range of use cases, layouts, and content.

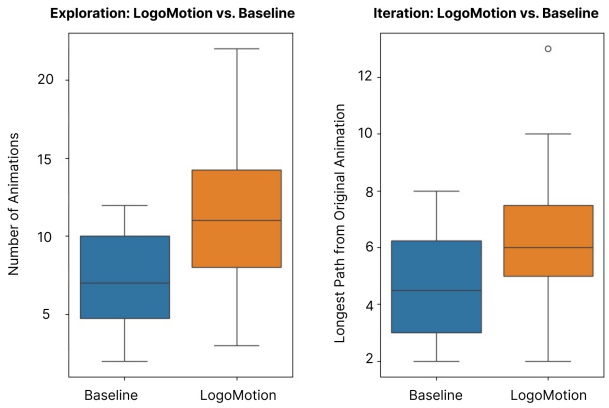


Figure 9: Boxplots show the distribution of animations explored and the iteration depth. Participants explored significantly more animations and had longer lengths of iteration with LogoMotion over the baseline.

7.2.3 Types of Customization. Participants were expressive in their natural language editing requests. P14 edited a logo featuring a manta with the prompt: *"During hero moment, stingray floats backward as if fighting against a current"*, going for an interaction between the primary element and background. In animating a logo for Piper Plumbing, P1 wrote (*"water droplet appears as if it is an actual drop of water coming from the pipeline, after the wheel has turned once"*) and was pleased to see that LogoMotion could fulfill

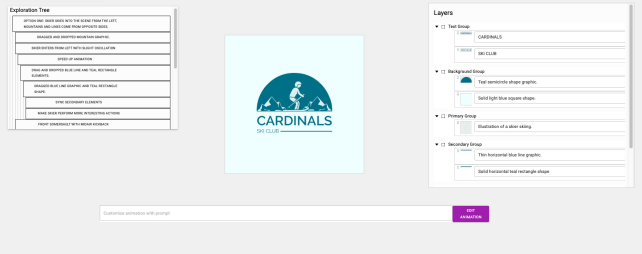


Figure 10: Our baseline condition included the Exploration Tree, Prompt Box, and Layer panel. It provided prompting as the main form of interaction.

such a complex edit. P16 iterated on getting a primary logo element to have a "gentle baby walk" entrance into the frame using both prompts and the "Subtle" quick action.

8 Discussion

We have introduced LogoMotion, VLM-powered code generation approach that helps users create semantically meaningful animation using 1) visually-grounded code synthesis and repair and 2) code-connected editing widgets. We now discuss how the merits of these contributions and how they generalize beyond logo animation.

8.1 Generalizability of Visually-Grounded Code Synthesis and Repair

Visually-grounded code synthesis is powerful because it unlocks how code generation approaches can be guided by an AI's ability to analyze images. LogoMotion shows how to do this by building an HTML representation that is visually aware of what is on the canvas. The HTML representation reflects the visual hierarchy of the elements in the image and captures grouping information in the code hierarchy organization. LogoMotion demonstrates how a VLM can be applied to understand layer content, automatically make groups, and update the HTML representation when users interact with a layers panel. This is relevant to any design tool that has a layer panel. In tools like Adobe Illustrator and Figma, users engage with hundreds of layers and groups, and LogoMotion shows how an AI approach can solve classic painpoints like layer organization and group layers for users. Secondly, LogoMotion shows how a code generation approach can be guided by a design concept. This is powerful because it allows animation to move beyond templates and presets –LogoMotion outperforms an industry approach Magic Animate in this respect–and move towards semantically meaningful animation. Visually-grounded code synthesis can also extend to other animation tasks like slide animation (where visual hierarchy and grouping is key) or kinetic typography (where the motion must match the semantic meaning of the text).

LogoMotion's visually-grounded program repair mechanism also has high generalizability. It can be effective for many other tasks where code renders visual output such as UI mockups, front-end prototyping, and game design. For example, if a UI prototype is generated from a visual mockup, it can be checked for if whether components meet the visual specification of the mockup. Furthermore, for this checking process to be able to isolate visual differences, visual reflection has to be able to support different "beams of focus". LogoMotion shows how to implement visually-grounded program repair with a layer-wise beam of focus and effectively solve 96% of detected errors within four attempts. Though in LogoMotion we only checked the last frame of the animation against the target layout, in future work, we can check intermediate frames of animation to build a deeper understanding of motion.

8.2 Generalizability of Code-Connected AI Editing Widgets

Code-connected AI editing widgets allow users to enjoy the benefits of a code representation (controllability, greater expressive range) without having to engage with the technical details of implementation. LogoMotion's narrative timeline widget allowed users to pair the expression gained from prompts with the other modes of controls that come from GUIs (e.g. selection, reordering, drag-and-drop). Each time the animation was edited, the AI implementation would automatically produce a well-eased and smooth animation, without requiring any tedious manipulation of timestamps or fine-grained editing that is common with traditional tools. The narrative timeline that we introduce is also a novel improvement upon traditional timelines, which are not capable of describing the events that take place within their keyframes and blocks. Narrative timelines can generalize to other design tasks such as video editing. Overall, these code-connected AI widgets are new components that open up interactions beyond the standard chatbot paradigm.

8.2.1 Semantic vs. Specific Controls for Animation Editing. LogoMotion allowed us to explore when it is best to support users with generative, semantic-based controls (prompts), when it is best to support users with granular specific controls (timelines), and when there are opportunities to bind these two types of controls together within an interface. The obvious benefit of semantic controls is that it opens up animation to novices–expressing animation is as simple as expressing a thought or story. Furthermore, in language it is easier to have conceptual reach and to divergently explore animations. Participants used properties of language like metaphors (*"gentle baby walk"*) and action description (*"swimming as if fighting against a current"*) to create connections that reinforced the high-level visual message. However, language as the sole mode of expression is insufficient–it lacks the control necessary for animation, which has many dimensions that rely on perceptual feel such as timing, dynamics, synchronization, grouping, and interactions. These aspects are hard for novices to find the language for; a novice can write that two objects should overlap in timing, but overlap by how much, and which element should come first? If something should be faster, by how much? What does it mean for two elements to "come in together"? Quickly, it becomes unwieldy to find the right balance between underspecification and overspecification in natural language. LogoMotion's code-connected components allowed users to divert many of these editing goals to GUI interactions that are second nature (e.g. seeing and setting the absolute timing on elements, defining what elements group together in the organization of the animation). At the same time, LogoMotion demonstrated how semantic and specific controls can be paired together to regenerate the underlying code. Point-and-selection of blocks from the

narrative timeline could be bound to semantic intents like quick actions and prompts.

8.3 Limitations and Future Work

We discussed LogoMotion outputs with professional motion designers, who provided insights about LogoMotion's limitations. While of quality, LogoMotion does not produce animations that are on a professional level. The animations do not at a meta-level guide the viewer's eye. The animations also often unevenly apply fundamental principles of animation. They mentioned how the the design concept could excessively apply overlapping action and could miss the opportunity to create complex interactions between groups. LogoMotion also does not support complex motion paths, morphing, and physics-based animation. We made efforts to explore path motion but found that it was hard to integrate an SVG path defined outside of the design concept into the generated code. Motion paths that can define primary and secondary motion, incorporate richer curved motion, and open up the opportunity for direct manipulation represent an important next step.

Another limitation is that LogoMotion scoped around logos. To focus on creating semantically meaningful animation, we made design decisions specific to logos (e.g. there is generally one hero element that should get more emphasis in the animation). However, other formats have different motion design principles. We experimented with having LogoMotion animate social media posts as well as a few webpages. We found that in these formats where there is more text, the animation had to be more subtle to balance readability and advertisement. This helped further establish the importance of having layer panel interactions and quick actions (Synchronize, Make Subtle). To adapt to other domains like CSS animation or vector animation, LogoMotion would likely have to adapt to generate and check on color and shape change. It is possible that the focus on the entrance description and use of the word "motion" predisposed LogoMotion to animate in terms of motion rather than these other properties.

Lastly in terms of our user evaluation, we only tested LogoMotion with sixteen people. For three of the early participants, the drag-and-drop interaction was too sensitive, so accidental interactions had to be excluded from analysis. Another limitation was that some participants had less fruitful experiences with LogoMotion because logo they chose to work with did not present many opportunities for animation. The logo elements, being all raster images, did not support deformable motion and could not be broken further down into parts. LogoMotion ideally could have given feedback for what edits would be more possible to fulfill and what was out of bounds.

9 Conclusion

LogoMotion presents an VLM-powered tool that allows users to automatically and interactively animate logos and create semantically meaningful motion. Its approach has three stages: visually-grounded code synthesis, visually-grounded program repair, and code-connected widgets. Within this editor, a narrative timeline, layer panel, and quick actions helped users edit for motion, grouping, and timing of their animation. We show in evaluations that LogoMotion outperforms a state-of-the-art industry tool at producing animation that is relevant to its design elements. LogoMotion was found to be expressive and enjoyable for novices to create and edit logo animation.

References

[1] 2023. BlenderGPT. https://github.com/gd3kr/BlenderGPT
[2] 2023. Capcut Templates. https://www.capcut.com/templates
[3] 2023. Motion Array Premiere Templates. https://motionarray.com/browse/premiere-pro-templates/
[4] 2023. Pinterest Shuffles. https://www.shffls.com/
[5] 2024. Adobe Express Animated Templates. https://www.adobe.com/express/create/logo/animated
[6] 2024. Canva Magic Animate. https://www.canva.com/help/using-magic-animate/
[7] 2024. Figma Smart Animate. https://help.figma.com/hc/en-us/articles/360039818874-Smart-animate-layers-between-frames
[8] 2024. Powerpoint Morph. https://support.microsoft.com/en-us/office/use-the-morph-transition-in-powerpoint
[9] 2024. Runway Motion Brush. https://academy.runwayml.com/gen2/gen2-motion-brush
[10] Adobe. 2022. Animate text, videos and photos in Adobe Express. https://helpx.adobe.com/express/using/animation.html
[11] 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 (UIST '23)*. ACM, 1–22. doi:10.1145/3586183.3606719
[12] 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 (UIST '23)*. ACM. doi:10.1145/3586183.3606719
[13] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. arXiv:2302.04023 [cs.CL] https://arxiv.org/abs/2302.04023
[14] Yining Cao, Jane L E, Zhutian Chen, and Haijun Xia. 2023. DataParticles: Block-based and Language-oriented Authoring of Animated Unit Visualizations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (, Hamburg, Germany,) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 808, 15 pages. doi:10.1145/3544548.3581472
[15] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching Large Language Models to Self-Debug. arXiv:2304.05128 [cs.CL]
[16] Xinyuan Chen, Yaohui Wang, Lingjun Zhang, Shaobin Zhuang, Xin Ma, Jiashuo Yu, Yali Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. 2023. SEINE: Short-to-Long Video Diffusion Model for Generative Transition and Prediction. arXiv:2310.20700 [cs.CV] https://arxiv.org/abs/2310.20700
[17] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. arXiv:2303.04864 [cs.LO]
[18] Katherine Crowson. [n. d.]. Alembics/disco-diffusion. https://github.com/alembics/disco-diffusion
[19] Richard C. Davis, Brien Colwell, and James A. Landay. 2008. K-sketch: a 'kinetic' sketch pad for novice animators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. Association for Computing Machinery, New York, NY, USA, 413–422. doi:10.1145/1357054.1357122
[20] Mira Dontcheva, Gary Yngve, and Zoran Popović. 2003. Layered acting for character animation. *ACM Trans. Graph.* 22, 3 (July 2003), 409–416. doi:10.1145/882262.882285
[21] Peitong Duan, Jeremy Warner, Yang Li, and Bjoern Hartmann. 2024. Generating Automatic Feedback on UI Mockups with Large Language Models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24, Vol. 4)*. ACM, 1–20. doi:10.1145/3613904.3642782
[22] Mark Chen et. al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]
[23] OpenAI et. al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[24] Alessandro Favero, Luca Zancato, Matthew Trager, Siddharth Choudhary, Pramuditha Perera, Alessandro Achille, Ashwin Swaminathan, and Stefano Soatto. 2024. Multi-Modal Hallucination Control by Visual Information Grounding. arXiv:2403.14003 [cs.CV] https://arxiv.org/abs/2403.14003
[25] Alexander J. Fiannaca, Chinmay Kulkarni, Carrie J Cai, and Michael Terry. 2023. Programming without a Programming Language: Challenges and Opportunities

for Designing Developer Tools for Prompt Programming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>Hamburg</city>, <country>Germany</country>, </conf-loc>) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 235, 7 pages. doi:10.1145/3544549.3585737

[26] Sumit Gulwani, Alex Polozov, and Rishabh Singh. 2017. *Program Synthesis*. Vol. 4. NOW. 1–119 pages. https://www.microsoft.com/en-us/research/publication/program-synthesis/

[27] Aditya Gunturu, Yi Wen, Nandi Zhang, Jarin Thundathil, Rubaiat Habib Kazi, and Ryo Suzuki. 2024. Augmented Physics: Creating Interactive and Embedded Physics Simulations from Static Textbook Diagrams. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 144, 12 pages. doi:10.1145/3654777.3676392

[28] Yuwei Guo, Ceyuan Yang, Anyi Rao, Yaohui Wang, Yu Qiao, Dahua Lin, and Bo Dai. 2023. AnimateDiff: Animate Your Personalized Text-to-Image Diffusion Models without Specific Tuning. arXiv:2307.04725 [cs.CV]

[29] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. arXiv:2105.09938 [cs.SE]

[30] Han Huang, Fernanda De La Torre, Cathy Mengying Fang, Andrzej Banburski-Fahey, Judith Amores, and Jaron Lanier. 2023. Real-time Animation Generation and Control on Rigged Models via Large Language Models. *arXiv preprint arXiv:2310.17838* (2023).

[31] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.* 24, 3 (July 2005), 1134–1141. doi:10.1145/1073204.1073323

[32] Amir Jahanlou and Parmit K Chilana. 2022. Katika: An End-to-End System for Authoring Amateur Explainer Motion Graphics Videos. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 502, 14 pages. doi:10.1145/3491102.3517741

[33] Ollie Johnston and Frank Thomas. 1981. *The illusion of life: Disney Animation*. Disney Editions.

[34] Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. 2012. Cliplets: juxtaposing still and dynamic imagery. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (Cambridge, Massachusetts, USA) *(UIST '12)*. Association for Computing Machinery, New York, NY, USA, 251–260. doi:10.1145/2380116.2380149

[35] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: bringing life to illustrations with kinetic textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 351–360. doi:10.1145/2556288.2556987

[36] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: Bringing Life to Illustrations with Kinetic Textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 351–360. doi:10.1145/2556288.2556987

[37] Rubaiat Habib Kazi, Tovi Grossman, Nobuyuki Umetani, and George Fitzmaurice. 2016. Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4599–4609. doi:10.1145/2858036.2858386

[38] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2023. Chain of Code: Reasoning with a Language Model-Augmented Code Emulator. arXiv:2312.04474 [cs.CL]

[39] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (Dec. 2022), 1092–1097. doi:10.1126/science.abq1158

[40] Jiaju Ma, Li-Yi Wei, and Rubaiat Habib Kazi. 2022. A Layered Authoring Tool for Stylized 3D Animations. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 383, 14 pages. doi:10.1145/3491102.3501894

[41] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. *ICLR* (2023).

[42] Simon Niklaus, Long Mai, and Feng Liu. 2017. Video Frame Interpolation via Adaptive Convolution. arXiv:1703.07514 [cs.CV] https://arxiv.org/abs/1703.07514

[43] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show Your Work: Scratchpads for Intermediate Computation with Language Models. arXiv:2112.00114 [cs.LG]

[44] Val Ralabate. 2014. UI Animation and UX: A Not-So-Secret Friendship. *A List Apart* (2014). https://alistapart.com/article/ui-animation-and-ux-a-not-so-secret-friendship/#section4 Accessed: 2024-09-09.

[45] Adrianna Ratajska, Matt I Brown, and Christopher F Chabris. 2020. Attributing Social Meaning to Animated Shapes: A New Experimental Study of Apparent Behavior. *The American Journal of Psychology* 133, 3 (10 2020), 295–312. doi:10.5406/amerjpsyc.133.3.0295

[46] Karl Toby Rosenberg, Rubaiat Habib Kazi, Li-Yi Wei, Haijun Xia, and Ken Perlin. 2024. DrawTalking: Building Interactive Worlds by Sketching and Speaking. arXiv:2401.05631 [cs.HC] https://arxiv.org/abs/2401.05631

[47] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[48] Nazmus Saquib, Rubaiat Habib Kazi, Li-Yi Wei, and Wilmot Li. 2019. Interactive Body-Driven Graphics for Augmented Video Performance. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3290605.3300852

[49] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2Code: How Far Are We From Automating Front-End Engineering? arXiv:2403.03163 [cs.CL]

[50] Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. ViperGPT: Visual Inference via Python Execution for Reasoning. *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2023).

[51] Gemini Team. 2023. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL]

[52] Tiffany Tseng, Ruijia Cheng, and Jeffrey Nichols. 2024. Keyframer: Empowering Animation Design using Large Language Models. arXiv:2402.06071 [cs.HC]

[53] Nora S. Willett, Rubaiat Habib Kazi, Michael Chen, George Fitzmaurice, Adam Finkelstein, and Tovi Grossman. 2018. A Mixed-Initiative Interface for Animating Static Pictures. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 649–661. doi:10.1145/3242587.3242612

[54] Sharon Zhang, Jiaju Ma, Jiajun Wu, Daniel Ritchie, and Maneesh Agrawala. 2023. Editing Motion Graphics Video via Motion Vectorization and Transformation. *ACM Transactions on Graphics* 42, 6 (Dec. 2023), 1–13. doi:10.1145/3618316