LogoMotion: Visually Grounded Code Generation for Content-Aware Animation

Vivian Liu
vivian@cs.columbia.edu
Columbia University

Rubaiat Habib Kazi
rhabib@adobe.com
Adobe Research

Li-Yi Wei
lwei@adobe.com
Adobe Research

Matthew Fisher
matfish@adobe.com
Adobe Research

Timothy Langlois
tlangloi@adobe.com
Adobe Research

Seth Walker
swalker@adobe.com
Adobe Research

Lydia Chilton
chilton@cs.columbia.edu
Columbia University

ABSTRACT

Animated logos are a compelling and ubiquitous way individuals and brands represent themselves online. Manually authoring these logos can require significant artistic skill and effort. To help novice designers animate logos, design tools currently offer templates and animation presets. However, these solutions are limited in their expressive range. Large language models have the potential to help novice designers create animated logos by generating animation code that is tailored to their content. In this paper, we introduce LogoMotion, an LLM-based system that takes in a layered document and generates animated logos through visually-grounded program synthesis and program repair. In the program synthesis stage pictured above, multimodal LLM operators take in visual context and handle the construction of a text representation of the canvas, conceptual grouping of elements, and implementation of animation code.

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 ACM 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.

Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...
https://doi.org/10.1145/1122445.1122456

CCS CONCEPTS
- Applied computing → Media arts; Computing methodologies → Natural language processing; Computer vision tasks.

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

ACM Reference Format:

1 INTRODUCTION
Motion suggests life, and as such, motion is a dimension we add to our designs to make them more dynamic and engaging. Animation is a special type of design form which we have created to help us take static designs into more media-rich and interactive contexts. A specific type of animated content that we frequently create is the animated logo. Animation allows logos, which have been defined as the “visual figureheads” of brands [25], to better integrate within videos, livestreams, websites, and social media. A well-executed animation can quickly engage an audience, introduce the brand or individual online, and elevate content to have more visual interest.

Authoring an animated logo is challenging. Logos are often more than just a pairing of icon with text. Because they can have different layouts, layers, color, and typography, they can take on great variety and be complex artifacts to animate. For a novice designer, it can be difficult to understand which design elements should be animated, in what sequence, and how to build up compelling and believable motion. There are many facets of motion to consider such as speed, timing, positioning, duration, easing, and motion personality (e.g. a playful bounce vs. a strong entrance). Additionally, when logos have more design elements, designers also have to understand how groups of elements can synchronize to coordinate motion and orchestrate a visual flow.

While there is a great demand for animated content, it is difficult for people outside of motion design to develop this kind of expertise. Design tools such as Adobe Express, Canva, and Figma often provide solutions in the form of animated templates and automatic animation techniques [10, 12, 13]. Templates pre-populate logo layouts with animations that users can customize. They illustrate how users can apply motion presets (e.g. slide, flicker, or fade) onto logo elements to create professional-looking animations. However, templates do not always adapt to every use case. When users make edits (e.g. add/remove/replace elements) to customize logo templates, they can easily break the seamless and professional look the templates were originally packaged with. An alternative to templates are automatic animation techniques, which globally apply rules and heuristics to animate canvases [12]. For example, all elements on a page can be directed to slide in from one side or sequentially fade into place. While templates and automatic techniques can get users to a starting point fast, neither solution works with a recognition of the user’s content, which is something that can be enabled by emerging technologies.

Large language models (LLMs) present the potential for content-aware animation. They can generate animation code that is specific to the design elements and their layout on the canvas. Code is a text representation that is often used to drive animation [18, 33, 53], because it can concisely specify how elements interact over time and space on a canvas. Because LLMs encode a vast amount of world knowledge, they can draw upon actions and activities related to the content being animated and generate a near infinite number of animations. This open-ended generative capacity can go beyond the scope of what templates, presets, and rule-based techniques usually cover.

Recent advancements have made LLMs more multimodal, such that they can take in both text and image as inputs, and provide visually-grounded responses. This make LLMs more applicable in domains like animation where a visual understanding of the canvas matters. It opens up the potential for users to provide images of their layout to an LLM and receive animations tailored to their layout and design elements. For example, if a novice designer wanted to animate a taxi, they could use an LLM to generate code to drive a taxi onto the canvas. This code could translate the taxi object along the x-axis before easing it into the center of the canvas to imply a stop-and-go motion befitting of taxis.

In this paper, we present LogoMotion, an LLM-based method that automatically animates static layouts in a content-aware way. LogoMotion generates code in a two-stage approach involving visually-grounded program synthesis and program repair. The first stage introduces multimodal LLM operators that take in visual context and handle the 1) construction of a text representation of the canvas, 2) conceptual grouping of elements, and 3) implementation of animation code. The second stage of our approach introduces a technique for visually-grounded program repair, which helps LLMs check what they have generated against the original layout and debug differences in a targeted layer-wise fashion.

Our contributions are as follows:

- LogoMotion, an LLM system that uses visually-grounded code generation to automatically generate logo animations from a PDF. The system identifies the visual content in each layer, infers the primary and secondary elements, and creates groups of elements. Based on this, the system suggests a design concept (in text) and uses the LLM to generate animation code. Users can optionally improve the animation by editing or adding their own design concept.
- Visually-grounded program repair, a mechanism that lets the LLM automatically detect and debug visual errors within its generated animation code, creating a feedback loop between LLM-generated code and its visual outputs.
- A technical evaluation of 276 animations showing that compared to Canva Magic Animate and an ablated version of the system (without stages for hierarchy analysis and design concept suggestions), the full pipeline of LogoMotion produces animations that are more content-aware.
• A qualitative evaluation of novice users showing that LogoMotion is able to quickly achieve their desired animation with minimal reprompting.

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 [30]. 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 [42, 47, 49, 52, 57]. These models generally take in a natural language specification of the problem (e.g. docstrings), test cases, and examples of inputs and outputs. These models have shown remarkable ability at being able to solve complex programming problems at the level of the average human programmer [42]. 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 [26]. Converting a user intention into code often involves intermediate representations such as scratchpads [48] and chain-of-thought / chain-of-code operations to derive and implement a technical specification [23, 41].

While code generation models have primarily been benchmarked on text-based programming problems (e.g. LeetCode problems), they have also 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 that derive answers to visual queries [56]. HCI systems have also shown that code generation models can be integrated within creative workflows and provide interactive assistance [17, 58]. Spellburst demonstrated how LLMs can be purposed to help end users explore creative coding, a form of generative art, by writing prompts in natural language and merging underlying code representations [17]. BlenderGPT is an open-source plugin that allows users to translate a prompt into actions within Blender involving scene creation, shader generation, and rendering [6]. Design2Code recently illustrated that front-end programming can also be generatively created by finetuning code models and applying self-revision prompting [54]. However, Design2Code is currently outperformed by state-of-the-art LLMS (GPT-4V). As in these earlier works, code generation models often compose abstractions from libraries that were written to programmatically create visuals (bpy, CSS, p5.js) [27, 53].

A recent direction within the program synthesis space has been program repair through self-refinement. Program repair refers to automatic approaches for bug fixing, and self-refinement is the idea that LLMs can inspect and edit their code [22]. However, these approaches have generally been focused on text-based tasks and programming problem benchmarks [21, 32]. Our work shows how self-refinement can be extended into the visual domain by detecting visual errors at the layer level and providing image "diffs" that describe the bug for visually-grounded program repair.

2.2 Creativity Support Tools for Animation

Animation is a highly complex creative task. Tools that support it can be as novice-friendly as Google Slides [7] or as steep in learning curve as Adobe After Effects [1] and Autodesk Maya [3]. Animation spans a broad range of creative tasks, from conceptualization (scriptwriting, creating animatics) to asset creation (graphic design and storyboarding) to motion design (particle, primary, and path motion) [35]. Research tools often help users with the end-to-end creation of a target artifact. For example, Katika is an end-to-end tool that helps users create animated explainer videos by converting animation scripts into shot lists and finding relevant graphic assets and motion bundles [34]. Other systems have helped users create animated unit visualizations [20], 3D animations [46], and kinetic illustrations [38] by basing interactions around fundamental animation principles [36]. These principles help maximize the effect of animation by separating out dimensions such as primary and secondary motion, staging, timing, anticipation.

Many approaches focus on the specific task of converting static assets to animated ones by designing ways to define motion. Motion can be derived from a number of places: it can be customized from templates [10, 12], isolated from videos [37], orchestrated through particle and path motion [2, 39], or directed through language-based transformations [20, 43]. Templates and page-level animations are popular in commercial tools such as Adobe Express, Canva, Capcut, and Pinterest Shuffles [4, 5, 8–10], because they allow users to explore a diverse range of animation possibilities while reducing manual effort—users do not have to animate each element independently. Templates for video and animation have been found to be helpful for introducing novice designers to expert patterns within a design space, adding structure to their creative process, and boosting the overall quality of their creations [40, 60].

2.3 Generative Tools for Design

Generative AI technologies have popularized natural language as a new form of interaction for content creation. LLMs [19] have shown promise in brainstorming support [44], script and writing assistance [29, 59, 61], and sensemaking [55]. Text-to-image models [14, 15, 50, 51] have shown to be effective at visual asset generation for visual blending [28], news illustration [44], storyboarding [59], product design [45], world building [24], and video generation [43]. Generative technologies have also begun to be applied to motion design and animation [31, 43].

The closest generative work to ours is Keyframer, a study of how novice and expert designers prompt GPT for animations. A major finding is that 84% of prompts were semantic in nature—users wanted to describe high level directions like “make the clouds wiggle” more often than low level prompts like changing the opacity. This clearly shows that people want semantically relevant animations—motion that characterizes how that element might move in real life.

LogoMotion studies a similar problem (animating digital layouts, but in the logo domain), and we also use LLMs for code synthesis. However, we build upon this direction by introducing a pipeline that performs code synthesis and program repair in a visually-grounded manner. Keyframer generated animation code without using visual context from the canvas and had less built-in support.
for the grouping and timing of design elements. The preprocessing and image understanding implemented within LogoMotion helps it come up with sophisticated design concepts that specify hero moments for the primary element and handle the sequencing of other design elements (e.g. synchronized secondary elements, text animation). Furthermore, we compare our approach to state-of-the-art baselines and show a significant improvement in content awareness.

3 FORMATIVE STEPS
In this section we introduce the design challenges and principles that surround our problem statement of logo animation. We took a mixed methods approach to our formative work: collecting and affinity mapping exemplars of the class, reading design literature about logos, interviewing motion designers, and analyzing existing end-user tools [10, 12, 13]. This step also guides the design and technical choices of our method.

We interviewed 4 professional motion designers (E1-E4) with at least 10 years of experience with motion design tools to understand these different methods (e.g. templates, manual authoring) and conducted needfinding around logo animations. Three motion designers interviewees (E1, E2, E4) authored logo and brand animations professionally.

3.0.1 Motion Presets. Consumer design tools [10, 12, 13] usually give users control over animation using motion presets. For example, design tools from iMovie to PowerPoint universally support simple animation presets like fades, slides, and wipes. These presets allow users to customize animation by changing motion properties such as speed, direction, and duration. A common design pattern is to also parameterize an animation with intro, looping, and outro animations, to help users structure different states within their animation [16]. Examples of animation presets and how presets can be parameterized and customized is pictured in Figure 2.

3.0.2 Heuristics and Grouping. Design tools often also analyze the layout of the canvas and apply heuristics or rules to page-level animations [12]. For example, all non-background elements can animate in from the edges, fall in from one direction, or fade in sequentially. These sorts of motion styles depend upon an understanding of the canvas in terms of layout hierarchy (background vs. foreground), element groupings (symmetrical elements, element copies), object salience (primary vs. secondary) and element type (text vs. image). These kinds of automatic approaches make it easier for users to reach complex animations while abstracting away technical complexities such as layered timelines and easing curves for the user. Thus, an analysis of the visual hierarchy of elements within a canvas is necessary to guide their timing and synchronization within the animation. In our analysis of professionally created animated logos and animated logo templates, we found that image elements tend to enter first and settle into place. Text animation tends to be last so that the viewer can take in the full effect of the visual messaging.

3.0.3 Limitation of Templates. Consumer design tools offer a wide range of animated template galleries designed by professionals. While templates can help users get to a polished and editable starting point fast (E1, E2), they do not always easily adapt to the user’s content or use case (E2). For example, Figure 2 shows a scenario where a user has picked an animated template with a preset “drift in from right” animation to start from. They can swap out the placeholder content with their own image assets and content and start customizing the template to their use case. However, even though the image assets are of the same class (a car), the built-in animation does not apply (their car slides in backwards). They have to discard the default animation and work backwards within the controls that they have to make believable and natural-looking animation, which can nullify the point of using an animated template. To mitigate this, template galleries and the template designers behind them often populate and update template galleries with hundreds of templates to cover all possible use cases. This combined with the fact that logo animation has so many dimensions (image animation, text animation, layout possibilities) makes templates an exhaustive design solution to maintain. Templates can be usable yet brittle when users make too many edits away from the defaults.

E4 expressed the limitations of templates and their desire for motion designers to create characteristic and semantically meaningful motion in their work.

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. 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. -E4
3.0.4 Manual authoring with design tools. In contrast to consumer tools, the professional designers we spoke to spent significant effort focusing on the visual flow and sequencing of their animations. They took motion design briefs from clients and made sure that the keyframing, transforming, and easing of elements looked natural and professional (E1, E2). While a hand-crafted approach enables animations to be bespoke and tailored to specific content and narrative goals, it requires significant manual effort, time, and expertise.

From this formative understanding of the design landscape for animated logos, we synthesized the following design principles that guide our approach.

- **Design Principle 1. Animation should be content-aware.** Animations should be customized to the subject matter of the logo: be it by applying characteristic motion to the hero elements, creating a pacing and visual flow that matches the overall messaging, or showing layout awareness.

- **Design Principle 2. Animation should respect layout hierarchy.** The animation should reflect the visual hierarchy of the layout. Primary elements should have the most animation (a hero moment), and secondary elements should have less or more subtle animation to not detract from the primary element. Elements that conceptually group together should be coordinated in motion.

- **Design Principle 3. Animations should have logical sequencing.** Create a sense of visual flow and sequencing fitting for a logo reveal. The approach should help users automatically create well-eased and well-paced animations.

4 LOGOMOTION SYSTEM

We present LogoMotion, a LLM-based method that automatically animates logos based on their content. The input is a static PDF document which can consist of image and text layers. The output is an HTML page with JavaScript code that renders the animation. The pipeline has three steps: 1) **preprocessing (for visual awareness)**, which represents the input in HTML and augments it with information about hierarchy, groupings, and descriptions of every element, 2) **visually grounded code generation**, which takes the preprocessed HTML representation and the static image of the logo and outputs JavaScript animation code, and 3) **visually-grounded program repair**, which compares the last frame of the animation to the target image and does LLM-based self-refinement if there are visual errors on any layer.

4.1 Input

A user begins by importing their PDF document into Illustrator. Within Illustrator, using ExtendScript, they can export their layered document into an HTML page. We use HTML as a fundamental representation to suit the strengths of an LLM and construct a text representation of the canvas. The HTML representation includes the height, width, z-index and top- and bottom- positions of every image element. Text elements are represented as image layers. Each word is captured as a separate image layer, and its text content is the alt text caption, except in the case of arced text (e.g. logo title in Figure 1), where each letter is a separate image layer. Every element is given a random unique ID. This representation allows the LLM to understand what layers make up the logo image.

The ExtendScript script automatically extracted the bounding boxes and exported each layer into two PNG images: 1) a crop around the bounding box of the design element and 2) a magnified 512×512 version of the design element, which was used for GPT-4-V for captioning.

4.2 Preprocess Visual Information

Given a basic HTML representation of the logo layout, the system does several pre-processing steps to add semantic information about the logo’s visual content.

4.2.1 Image descriptions. To provide information about what each layer depicts, we use GPT-4-V to isolate each layer against a plain background and produce descriptive text. We put this in the alt text HTML attribute. This is pictured in Step 1 of Figure 3.

4.2.2 Visual Hierarchy. To provide a visual hierarchy of elements, we give GPT-4-V the HTML representation of the canvas and the logo image and ask it to classify each element as one of four categories: primary, secondary, text, or background. This step outputs a new HTML file which includes the role classification in the class name of every element (class="primary", class="secondary", etc.). From our formative work on logo animation, we learned that generally logos have one primary element that deserves the most attention in animation. Thus, we restrict the LLM to select exactly one primary element.

Because primary elements will get characteristic motion applied to them, we need some extra information to describe their motion. This includes the orientation of their image to determine what direction they would come in. For example, a car facing left should drive in from the left, a car facing forward should start small and slowly enlarge as if it is driving towards you. We save this information in a variable (=entrance description=) that is used later in the suggestion of a design concept for the animation.

4.2.3 Grouping Elements. In addition to providing a hierarchy, we needed to understand which elements visually and conceptually group together. There are usually many secondary elements that have symmetry, similar positions, or other visual similarities that make them necessary to animate in together. For example, many stars in the night sky should twinkle or two mountains should rise together. To create groups, we called GPT-4-V to make subgroups over the elements that were tagged as secondary. We reorganized the text representation of the canvas such that groups of secondary elements were placed together where they are the children of a parent <div> element. The output of this step is shown as the AUGMENTED HTML in Step 2 of Figure 3.

4.2.4 Design Concept. From early explorations with the system, we realized that to get the system to produce coherent animations that told a story, we needed to provide a design concept to relate all the elements together. Thus, before the code generation step, we requested the LLM to return a natural language description of the animation. This stage encouraged the model to interpret the logo and connect image elements to relevant animation actions that they might take on in the real world. For example, a flower could bloom from the center of a screen by fading and scaling in, or a skier could ski in from the left side of the screen and rotate one turn...
Figure 3: Program Synthesis Overview. In Step 1, a PDF of a logo is converted into an HTML representation of the canvas. LogoMotion has pre-processing steps to caption each image element, extract its bounding box, and assign a z-order as per the layer ordering of the document. In Step 2, the HTML is augmented with information about visual hierarchy of the logo layout (e.g., what are primary / secondary elements, what elements group together). In Step 3, a design concept for the animated logo suggested. In Step 4, the LLM implements animation code for the design concept that will animate the logo HTML.

to suggest a flip. Secondary and text elements were also instructed to be given a narrative description of their animation, such that their animations would not clash with the primary elements. See the output of Step 3 from Figure 3 for an (excerpt) example of a design concept.

To automatically generate a design concept, we prompted GPT-4-V with an HTML file (augmented with the visual information) and an image of the logo, and asked it to write a design concept with the 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 and the other elements. (Note that the element is an image layer, so parts within it cannot be animated.)
The output design concept was saved as a variable to input in the code generation step. Although our pipeline creates a design concept, this can be edited, or iterated on by users if they want more control and interaction with the system.

4.3 Visually-Grounded Code Synthesis

The HTML, design concept, and initial image are then passed as inputs to the code synthesis stage. We use GPT-4-V to implement a code timeline using anime.js, a popular Javascript library for animation [11]. While we chose to use anime.js, the generated code manipulated elements through their transforms (position, scale, rotation), easing curves, and keyframes, which are abstractions that generalize across many libraries relevant to code-generated visuals (e.g. CSS). This was the prompt we used to generate animations.

Implement an anime.js timeline that animates this image, which is provided in image and HTML form below.

```html
<HTML>
Here is a suggested concept for the animated logo:

```
<DESIGN CONCEPT>
Return code in the following format.

```javascript
timeline
 .add({
 // Return elements to their original positions
 // Make sure to use from-to format
 // Use -512px and 512px to bring elements in from offscreen.
 })```

This animation timeline was then executed by using Selenium, a browser automation software that automatically drives a Chrome browser.

4.3.1 Visually Grounded Program Repair. A necessary stage after program synthesis is program repair. AI-generated code is imperfect and prone to issues with compilation and unintended side effects. Additionally, because our approach generates animations for every element, multiple animations have to compile and execute properly. This increases the chances that there will be at least one point of failure in the final animation. To prevent these errors, we introduce a mechanism for visually-grounded program repair to fix “visual bugs” for when elements do not return to their original positions after the animation. We detected these bugs by calculating differences between the bounding boxes between the last frame in our timeline and the target layout. Specifically, we checked for correctness in left, top, width, height, and opacity. This allowed us to check for issues in position, scale, and opacity directly and rotation indirectly.

We next handled bug fixing in a layer-wise fashion. The inputs to this step were: the element that was detected to have an issue (its element ID), the animation code, and two images representing the element at its target layout and where it actually ended in the last frame. GPT-4-V was then prompted to return an isolated code snippet describing the issue and a corrected code solution. This code snippet was then merged back into the original code using another GPT-4-V call. We checked the bounding boxes of images and text elements in a layerwise fashion. This reduced the amount of information the LLM had to process and allowed it to more easily visually analyze the difference between the two images. Once the bug fixes were complete, we generated the final output of LogoMotion: an HTML page with JavaScript code written with anime.js. An overview of the inputs and outputs of this stage is illustrated in Figure 4.

4.3.2 Interactive Editing. LogoMotion is a prompt-based generative approach. Within the operators, there are many places where design norms for different formats could change. For example, the duration of the animation, the level of detail described by the design concept, and the different kinds of groupings that can be made over a layout (e.g. group by proximity, group by visual similarity, group by regions of the canvas) are all free variables that could be considered high-level controls. To explore how users engage with these high-level controls, we also present users the possibility to interactively iterate upon these automatically generated logos. Interactive editing exposes a simple prompt to users which allows them to describe how they want the animation to change. Similar to the visual debugging stage, GPT-4-V converts this requested change (this time user-provided) into an isolated portion of code, which is merged back into the original timeline. The benefit of having the previous version of animation code to merge into means that users do not have to respecify how the other elements move.

5 EVALUATIONS

We conducted three evaluations to understand the quality of our LLM system: 1) a comparison study against an industry standard and baseline informed by professional animated logo designers 2) an empirical analysis of program repair testing different experimental settings, 3) an evaluation with novices to understand LogoMotion’s capacity for customization. These evaluations are centered around the following research questions:

RQ1: Across a wide range of designs, to what extent does LogoMotion support content-aware animation?  
RQ2 What are the overall strengths and the weaknesses of LogoMotion at animation?  
RQ3 What sorts of errors does LogoMotion tend to make?  
RQ4 How capably can visually-grounded program repair debug such errors and what settings of program repair impact performance?  
RQ5 To what extent can user interaction and iteration improve the quality of the automatically generated animated logos?

The first evaluation is a comparison study comparing LogoMotion animations against Magic Animate [4] which automatically recommends animations to all elements on a page after a user selects a page-level style (e.g. "Bold", "Professional", "Elegant"). We additionally compare LogoMotion with an ablated version of our system (which we will refer to as LogoMotion-Ablated). We ablated parts of our system that 1) conceptually grouped HTML and 2) suggested a design concept to see how these higher-level LLM operators impacted content awareness. Our exact hypotheses for RQ1 were the following.

H1a Compared to the other conditions, LogoMotion would produce animations that were more content-aware.

H1b Compared to the other conditions, the LogoMotion would be improved in terms of sequencing.

H1c Compared to the other conditions, LogoMotion would be better in terms of execution quality.
5.0.1 Methodology. We first gathered a test set of 23 templates that spanned different categories of objects (animate and inanimate), layerings, and use cases. Use cases included holiday greetings, school clubs, advertisements, and branding. All templates were sourced from Adobe Express and Canva and are accessible online.

Each template was exported as a PDF and then converted into an HTML representation using the methods mentioned in subsection 4.2. We ran LogoMotion to get four animations for each template. We then ran LogoMotion-Ablated to get another four animations for each template. In general, generating a set of four LogoMotion/LogoMotion-Ablated outputs took approximately 12 minutes. To have an industry standard set to compare against, we also collected four options from Magic Animate, which is an AI-based tool for automatic animation that is one of Canva Pro’s premium features. Their page-level presets are conceptually similar to templates (e.g., all elements fade in from one side of the canvas, elements wipe into place). We took first their recommended animation presets for the layout and then had an external designer pick the next best three presets, ensuring 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 runs per template) 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 content-aware the animation approach is. 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.

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 its ablated version (H1a, LogoMotion-Full: $M = 3.05, \sigma = 0.64$; LogoMotion-Ablated: $M = 2.68, \sigma = 0.58$; Magic Animate: $M = 2.33, \sigma = 0.33, p \leq 0.001$). 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 tend to come from LogoMotion (15 of top 20) or LogoMotion-Ablated (5 of top 20). LogoMotion animations that were rated highly tended to show motion that was archetypical of their subject. The video for this paper shows examples of lanterns blowing as if in slight wind, crabs crawling zig-zag across the screen, and skiers skiing downhill at a diagonal. Frames from examples are depicted in Figure 5. In the first row, a black knight is translated into the canvas in an L-like motion, like a chess piece being set down, 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,
Figure 5: 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.

Figure 6: We report the average top-2 relevance across conditions. LogoMotion was rated to be significantly better in terms of relevance.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Relevance</th>
<th>Sequencing</th>
<th>Execution Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogoMotion Full</td>
<td>3.05</td>
<td>3.15</td>
<td>3.25</td>
</tr>
<tr>
<td>LogoMotion Ablated</td>
<td>2.68</td>
<td>3.18</td>
<td>3.38</td>
</tr>
<tr>
<td>Magic Animate</td>
<td>2.33</td>
<td>3.12</td>
<td>3.22</td>
</tr>
</tbody>
</table>

Table 1: Relevance, Sequencing, Execution Quality. We took the top-2 from each condition within a template. Ratings were averaged across design professionals. ($^*$ = $p \leq 0.001$)

5.0.4 H1c. Execution Quality. In terms of Execution Quality, the full LogoMotion pipeline did not perform significantly differently from the other conditions (H1b, LogoMotion-Full: $M = 3.25$, $\sigma = 0.54$; LogoMotion-Ablated: $M = 3.38$, $\sigma = 0.46$; Magic Animate: $M = 3.22$, $\sigma = 0.39$).

LogoMotion-Ablated scored the highest on execution quality. Many animations for this condition tended to be conceptually similar (all elements fade or translate into place from a slight displacement) and were thus minimal in animation complexity and easy to execute. One factor that impacted execution quality was that LogoMotion could at times produce visual flaws that our bounding box checker was unable to catch. For example, LogoMotion could sometimes suggest animation code that targeted attributes like background color or outer glow. These would make it past our program repair stage (which did not check those properties) and bring down the execution quality ratings.

5.1 Evaluation: Program Repair

We next conducted an evaluation specifically centered around the visually-grounded program repair stage to methodically understand the sorts of errors LogoMotion would make (RQ3) and was capable of handling.
### Table 2: Table reporting how solve rate changes when k increases across two settings.

<table>
<thead>
<tr>
<th>k</th>
<th>Solve Rate_{+k}</th>
<th>Solve Rate_{−k}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td>2</td>
<td>0.85</td>
<td>0.68</td>
</tr>
<tr>
<td>3</td>
<td>0.92</td>
<td>0.75</td>
</tr>
<tr>
<td>4</td>
<td>0.96</td>
<td>0.82</td>
</tr>
</tbody>
</table>

of debugging (RQ4). We provide empirical analysis of this stage testing different experimental settings.

#### 5.2 Methodology

68% of the animation runs from the outset (after program synthesis) were error-free and did not require program repair. The other 32% required the program repair stage. 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 an LLM 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 [21], 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 / self-refinement.

The second setting that we varied was whether or not image context about the visual difference was provided. This image context pictured in Fig. 4 was a labeled image pair showing a layer at its target position vs. in its last frame in the animation). We refer to the setting with context about the visual difference (labeled image pair showing layer at target image and last frame) and bounding box information as Repair_{+imgs} and the setting with only bounding box information as Repair_{−imgs}.

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-refined animation code were generated for the Repair_{+imgs} condition, and 112 samples were generated for the Repair_{−imgs} condition.

#### 5.3 Findings

##### 5.3.1 RQ3. What errors does LogoMotion synthesis make? LogoMotion made 42 position-based errors in total. Position errors were made in 30.4% of the runs, meaning that almost all runs with errors detected a position error. These errors occurred when the left or top coordinate of the bounding box was off. LogoMotion made 26 scale-based errors in total, erroring in 18.4% of the runs, meaning that scale errors were less common than position errors. These errors occurred when the width or height dimensions of the bounding box were off. We did not detect any opacity errors in our test set.

Common errors resulted from not following the from-to format that is common to animation libraries (CSS 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.

Another type of position error would occur when there was 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’s width or height, but was rendered to be 100 percent (absolute with respect to the canvas). An example of this mistake within the LLM response is provided below.
Another type of error that was frequently encountered was when GPT would return a looping animation. Looping animations, as briefly mentioned in our formative steps, are a common design pattern to animation, and they would be instantiated by defining a small periodic action with the loop parameter set to true. Loop elements generally left the elements at small deltas from their intended positions but were easily resolved.

5.3.2 RQ4. How capably does LogoMotion fix its errors? Many errors were simple enough that they would only take only one attempt from LogoMotion to solve. This is pictured in Figure 8, by the predominance of the green bar for "Solved in 1" for each value of \( k \). Note that Figure 7 normalizes the number of elements, because it reports the proportion of animation code runs made error-free, while Figure 8 aggregates across all errors on all design elements. This distinction is important because the one run that could not be resolved (Figure 7, \( k=4 \)) had many elements whose individual errors were not resolved (Figure 8, \( k=4 \)), making the solve rate different at \( k=4 \) across the graphs.

Figure 7 shows that as \( k \) increases, so does the solve rate on the visual bugs for both Repair\(_{1Imgs} \) and Repair\(_{1Imgs} \). By Repair\(_{1Imgs} \) at \( k=4 \), 96% of the runs are resolved. There is a noticeable boost in performance from passing in visual context. The visual context was helpful generally when the error was visually apparent. Anecdotally, we found that when the difference was very slight (small pixel differences), GPT-4-V would respond and say that it could not tell the difference, presumably using the bounding box information and bug description to make the necessary changes. The bounding box information alone is still a form of visual grounding, because it provides the type of visual difference (e.g. position / scale) and quantitative values from the canvas, rather than relying GPT-4-V to come up with and potentially hallucinate differences.

6 EVALUATION WITH NOVICES

Lastly, we show the potential of LogoMotion for user interaction by showing LogoMotion to novices. We wanted to understand (RQ5) can user interaction and iteration improve the quality of automatically generated logos? More concretely, how usable did novices find the results, and how many rounds of iteration did novices take to reach a good design outcome?

6.0.1 Methodology. We recruited five animation novices from within a university. We had novices pick two templates to start with from a gallery of options. Of the LogoMotion-generated results, we asked how many they would consider publishable results and take as is and how many they would consider working with to edit and fix. If they chose an option to iterate upon, they provided prompts into a textbox, which would trigger edits to the underlying animation code.

6.0.2 Results. We found that many novices liked the system generations as is. In total, of the 16 animations that the novices saw, 10 were said to be usable as is. However, they also liked the ability to rapidly customize them and test smaller iterations around the design concepts they captured. For example, if the logo was of a skier skiing in from the left, they tried to ski them in at a lower angle or from a different entrance. They appreciated the ability to make these edits and get quick previews without having to do any direct manipulation with regards to timing, easing, or the rearrangement of other elements (all of which was preserved in the underlying code timeline). Novices liked to be able to use natural language to make edits that would target many different dimensions at once (visual changes, grouping changes, ordering changes). For example, novices put in prompts like, "have a slower delay at the beginning and don’t move the bird at the end", "make the text show quicker and don’t have a pulse animation at the end", or "the elements in the yellow banner should come in at the same time".

Novices appreciated the multiple of four options they got for each logo. One novice said that they did not realize there were so many options to pull a layout apart in terms of layers and directions until they saw four different animations making it happen. Novices also found inspiration in the galleries and would merge concepts from different options into new ones.

There were two cases where novices faced difficulties getting what they wanted from the system. The first was when they wanted to get complex interactions between two objects (e.g. having a shape interact with a nearby letter). The second was when edits novices made created new visual bugs that their exploration process would be derailed by. Nonetheless, all novices were satisfied with at least one animated logo after making just one to two rounds of iteration.

7 DISCUSSION

We have introduced an automatic approach for animation, which shows how LLMs can be applied to highly complex visual tasks involving canvas awareness, sequencing, and coordinated motion.

7.1 Breaking Away from Templates

Templates and presets are the standard approach to animation for novice designers and everyday creators. Beyond logos, they are the norm for applying motion to content of all kinds (e.g. slide presentations, videos, motion graphics, etc.), but from our needfinding, we found that they can be brittle and lack flexibility. LLMs can greatly open up the space of motion design to novice designers. They can start with an animation that is customized to their content rather than working backwards to make a template work for them. This approach can benefit both novice designers, who can have an alternative to searching for templates and template creators, who have to continually come up with new ways to populate template galleries. Furthermore, LogoMotion showed that novices could quickly find animation options that they were satisfied with and customize animations with ease using natural language.

7.2 Generating Code Around Visuals

We focused on code generation for animated logos, a highly specified kind of design artifact for which we made design assumptions (e.g. the presence of a primary element). However, there are many more design patterns and types of logos and layouts that LogoMotion could expand upon. For example, LogoMotion’s LLM method could also be applied to layouts such as social media square posts (e.g. flyers), email banners, and digital flyers. Each format has different "rules" and design norms. For example, the zero-to-hero effect...
that was necessary for logos is not as relevant for these other forms of visual communication, which can be more text heavy and prioritize the message getting across (therefore necessitating more minimal or subtle animation). A next step for this work could be to look at how the prompts, which are embedded within this pipeline, can be changed depending on the layout type to make the pipeline more flexible overall.

Additionally we found that the design concept stage was important in many respects. This stage created a tighter coupling between the image layers and their semantic content and gave the code more scaffolding for how it could be implemented in terms of translation, easing, timing, and duration. Additionally, in our early exploration with prompts in the pipeline, we found that when queried the model multiple times on the same layout, without a design concept, the LLMs would tend to produce same or similar animations run after run (e.g. a car would slide in from the left side of the screen each time). The design concept stage helped diversify the range of design options presented to users. It is also a novel instance of how a technical specification for code can be provided in narrative form (a story for the design elements) rather than as a technical specification or pseudocode, as it is often done in code generation literature.

7.3 Limitations
There are many more technical techniques such as path motion, morphing, and physics-based animation that we did not explore through LogoMotion. These more advanced techniques were out of scope of the LLM’s capabilities but are important to motion designers and highly relevant to what makes animated logos truly professional.

Other limitations were specific to the program repair stage. One was that the errors we checked for were primarily transform-based (e.g. position, scale, opacity). We did not anticipate that the LLM would return complex animations that could also target other properties such as background color and drop shadow. These sorts of errors, while easy for the human eye to perceive, would make it past our checker and bring down the execution quality of the animation. Additionally, the program repair stage checked the target layout only against the last frame—not the animation as a whole. While we briefly explored methods that involved optical flow and methods for measuring perceptual difference across frames, we found that it was difficult to define visual errors in the intermediate frames (e.g. hard to quantify what is a bad momentary overlaps or when design elements block other important elements).

8 CONCLUSION
LogoMotion presents an LLM-based method that automatically animates logo layouts (layered documents) with animation code that is specific to the contents of the layout. In two stages for visually-grounded program synthesis and program repair, we show how multimodal LLM operators can be chained to produce intermediate representations that help LogoMotion draw on the visual context in the canvas and world knowledge within the LLM before animation code is finally generated. We show in evaluations that LogoMotion outperforms state-of-the-art industry features in producing content-aware animation and can produce high quality results for novices that they can customize with ease.

REFERENCES