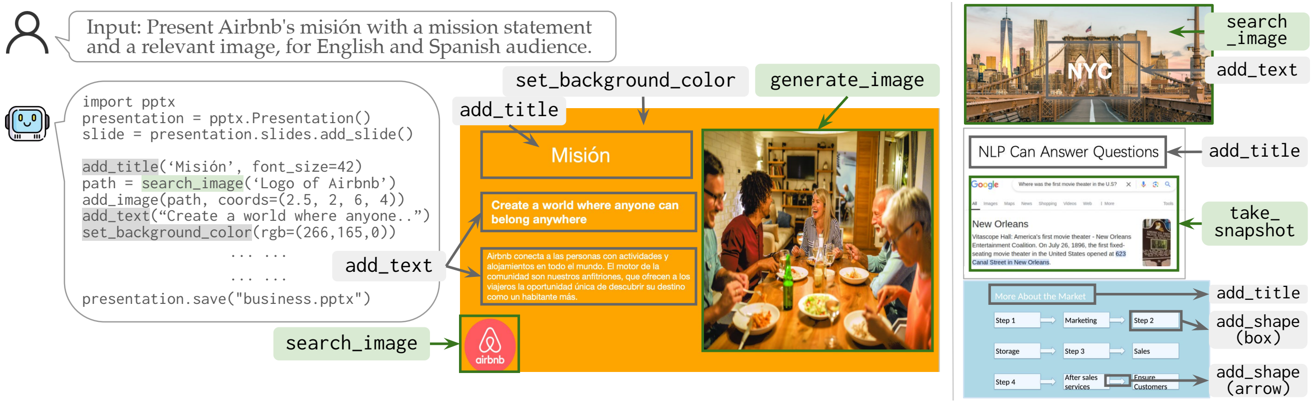
AUTOPRESENT: Designing Structured Visuals from Scratch

Jiaxin Ge[1]* Zora Zhiruo Wang[2]*
Xuhui Zhou[2] Yi-Hao Peng[2] Sanjay Subramanian[1] Qinyue Tan[2]
Maarten Sap[2] Alane Suhr[1]† Daniel Fried[2]† Graham Neubig[2]† Trevor Darrell[1]†

[1]University of California, Berkeley [2]Carnegie Mellon University



Input: Present Airbnb's misión with a mission statement and a relevant image, for English and Spanish audience.

```
import pptx
presentation = pptx.Presentation()
slide = presentation.slides.add_slide()

add_title(‘Misión’, font_size=42)
path = search_image(‘Logo of Airbnb’)
add_image(path, coords=(2.5, 2, 6, 4))
add_text(“Create a world where anyone..”)
set_background_color(rgb=(266,165,0))
                 ... ...
                 ... ...
presentation.save("business.pptx")
```

Misión

Create a world where anyone can belong anywhere

Airbnb conecta a las personas con actividades y alojamientos en todo el mundo. El motor de la comunidad son nuestros anfitriones, que ofrecen a los viajeros la oportunidad única de descubrir su destino como un habitante más.

NYC

NLP Can Answer Questions

Where was the first movie theater in the U.S?

New Orleans

Vitascope Hall: America's first movie theater - New Orleans Entertainment Coalition. On July 26, 1896, the first fixed-seating movie theater in the United States opened at 623 Canal Street in New Orleans.

More About the Market

Step 1 → Marketing → Step 2
Storage → Step 3 → Sales
Step 4 → After sales services → Ensure Customers

Figure 1. **Automatically generating slides from natural language instructions.** We propose AUTOPRESENT, a tool-augmented code generation method that follows natural language instructions to design slides from scratch, as shown in the examples. This allows for precise control over all elements, including textual content, images, visual layouts, coloring, and more.

Abstract

Designing structured visuals such as presentation slides is essential for communicative needs, necessitating both content creation and visual planning skills. In this work, we tackle the challenge of automated slide generation, where models produce slide presentations from natural language (NL) instructions. We first introduce the SLIDESBENCH *benchmark, the first benchmark for slide generation with 7k training and 585 testing examples derived from 310 slide decks across 10 domains.* SLIDESBENCH *supports evaluations that are (i) reference-based to measure similarity to a target slide, and (ii) reference-free to measure the design quality of generated slides alone. We benchmark end-to-end image generation and program generation methods with a variety of models, and find that programmatic methods produce higher-quality slides in user-interactable formats. Built on the success of program generation, we create* AUTOPRESENT, *an 8B* LLAMA*-based model trained on 7k pairs of instructions paired with code for slide generation, and achieve results comparable to the closed-source model* GPT-4O. *We further explore iterative design refinement where the model is tasked to self-refine its own output, and we found that this process improves the slide's quality. We hope that our work will provide a basis for future work on generating structured visuals. Our code, data, demo, and video demonstrations are publicly available at* https://github.com/para-lost/AutoPresent

1. Introduction

Designing structured visuals such as presentation slides from scratch is an essential skill for effective communication and conveying complex ideas [30]. Among various forms of visual communication, creating a compelling set of slides is a challenging problem, requiring content creation (text, pictures, diagrams, and more) and visual planning skills, to ensure the slides are well designed [25] and convey insights with clarity [3, 36]. Even human experts may need to spend hours iterating and polishing their slide decks [10] to produce high-quality designs with clear insights. While digital agents have demonstrated impressive capabilities in tasks such as software engineering [49], web navigation [46, 54], and free-form image design generation [8, 32], their creative capabilities in generating semi-

*Equal Contribution.
†Equal Contribution.

structured communicative media like slide decks has not been extensively tested. Therefore, we ask: *Can we employ powerful AI agents to create high-quality presentation slides that are well-structured and insight-revealing?*

In this work, we formulate the natural language (NL) to slide generation task. At a high level, the user provides the system with a natural language instruction about the desired slide, and the system then generates an editable presentation, as shown in Figure 1. We consider three types of user instructions: (1) *detailed instruction with images*. (2) *detailed instructions only*. (3) *high-level instructions*, reflecting varying levels of design freedom.

Since there are no existing tools for quantifying agent performance in slide generation tasks, we propose the SLIDESBENCH benchmark (§2) as a training source and test bed for method comparisons. SLIDESBENCH contains $7k$ training examples and 585 testing examples of varied instruction difficulties, constructed from 310 publicly available slide decks from 10 different domains, including art, business, and technology. To evaluate generated slides, we introduce two sets of evaluation metrics: *reference-based* metrics to examine position, content, and color match against the reference slide; and *reference-free* metrics inspired by slide design principles [5, 9, 34, 39, 44] to measure the design quality of agent-created slides alone, given that many good designs for the same instructions may vary from the reference slide.

To enable controlled and structured slide generation, we propose to create slides using program generation, where a model first generates a program from the natural language instruction, and then the program is executed to get the slide. We apply this approach to large language models (LLMs; LLAMA [11], DeepseekCoder [16], CodeLlama [33], GPT-4O [2]) and vision-language models (VLMs; LLAVA [51]). As illustrated in Figure 1, given a natural language instruction, the model first generates a Python program and then executes it to obtain a PPTX slide. We find that small models such as LLAMA (8B) and LLAVA (7B) are often unable to produce executable code. While GPT-4o can produce reasonable slides, it still exhibits a substantial gap in design quality compared to human-generated slides (§5). By further conducting iterative refinement, we find that models can self-refine and further improve slide quality. We also find that code generation approaches substantially outperform end-to-end image generation methods (Stable Diffusion [32], Dall-E [8]).

To further enhance the current model's ability to generate high-quality slides, we present our open-sourced AUTOPRESENT (8B) model (§4.2) which is fine-tuned from LLAMA 8B on the SLIDESBENCH training set. AUTOPRESENT achieves state-of-the-art performance among small open-sourced models and approaches the performance of the closed-sourced model GPT-4o. Since directly generating a long program is difficult for current models [14], we further create the SLIDESLIB library to simplify the program generation process. SLIDESLIB contains high-level functions that are *basic* such as `add_title`, and *image-related* such as `search_image` and `generate_image`. We show that LLMs and VLMs generally perform better when given access to SLIDESLIB.

Our main contributions can be summarized as follows:

- We formulate the NL-to-slide generation task and build SLIDESBENCH, the first benchmark for slide generation, which contains $7k$ training and 585 test examples and supports automatic evaluations.
- We leverage NL-to-program generation methods with refinement to produce high-quality slides, and benchmark diffusion models, VLMs, and LLMs.
- We train an 8B parameter open-source LLM, AUTOPRESENT, that approaches the performance of GPT-4o, and design a programmatic tool library SLIDESLIB that facilitates slide program generation across models.

2. SLIDESBENCH

In this section, we describe the creation of the SLIDESBENCH benchmark. Each instance consists of a natural language instruction to create a slide, and the slide itself (in PPTX format) as a reference. SLIDESBENCH includes three scenarios of varying difficulty levels designed to evaluate models with different user input. We describe the slide data collection (§2.1), three task setups (§2.2), and the annotation process (§2.3).

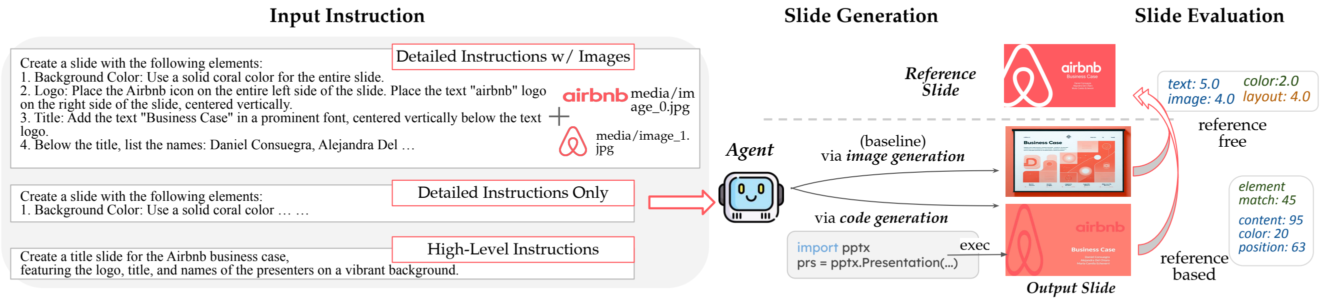
2.1. Slides Data Collection

We search the web and collect presentation slide decks from 10 domains, including art, marketing, environment, technology, etc. To select the highest-quality slide decks from each domain, we manually go through the relevant slide decks and conduct initial processing, by checking if all its slides (i) have visually structured layouts, and (ii) extractable media such as images (if any). For the slide decks with all slides satisfying (i) and (ii) in each domain, we incorporate one slide deck into the test set, and others into the training set. This results in a total of 10 and 300 slide decks (in PPTX format) for testing and training, each containing 20 slides on average. To respect the rights of the slide creators, we do not redistribute the slides. Instead, we provide a list of URLs for the slides that we used so that others can download the slides directly from the original website. We also provide an opt-out mechanism for any creator who does not want their slides in the dataset. We provide implementation details in §A.

2.2. Three Task Setups

We formulate the task as an NL-to-slide generation process. Given the reference slide, we curate three versions of natural language instructions, as shown in Figure 2, to represent slide generation tasks under varied difficulty levels. We introduce each setup below.

Detailed Instructions with Images The first and easiest setting is to provide the models with all the necessary in-



Create a slide with the following elements:
1. Background Color: Use a solid coral color for the entire slide.
2. Logo: Place the Airbnb icon on the entire left side of the slide. Place the text "airbnb" logo on the right side of the slide, centered vertically.
3. Title: Add the text "Business Case" in a prominent font, centered vertically below the text logo.
4. Below the title, list the names: Daniel Consuegra, Alejandra Del …

Create a slide with the following elements:
1. Background Color: Use a solid coral color … …

Create a title slide for the Airbnb business case,
featuring the logo, title, and names of the presenters on a vibrant background.

```
import pptx
prs = pptx.Presentation(...)
```

Figure 2. **Illustration of SLIDESBENCH.** Each example of SLIDESBENCH consists of three instructions: Detailed Instructions with Images, Detailed Instructions Only, and High-Level Instructions. The model is tasked to generate a slide based on the instruction, and the generated slide is evaluated on the metrics suite, which contains both the reference-free metrics and the reference-based metrics.

formation and assets to produce the reference slide, including text and image content, formatting and layout specifications. This setting evaluates models' visual planning abilities, such as arranging spatial layouts, maintaining formatting consistency, balancing content proportions, and emphasizing key elements.

Detailed Instructions Only Since a user may not specify, or know exactly what images to put on a slide, we propose a *detailed instruction only* setting, where we provide the same natural language instruction provided in the *detailed instruction with images* setting, but replace the provided images with their natural language descriptions (e.g., "two people shaking hands") generated by `gpt-4o-mini`. We then instruct the models to obtain the images using image searching or image generation tools. This setting further challenges models to interpret complex or compositional descriptions of images and obtain visuals that align with the slide context.

High-Level Instructions In contrast to users who have a concrete target slide in mind and can spell out all detailed instructions, some users may only be able to express their needs on a high level. We thus devise a *high-level instruction* setting, where the natural language instructions are rather high-level and only provide a general topical idea of the slide, such as "create a title slide for Airbnb," instead of detailing what logos and text to add and where, as exemplified in Figure 2. Models in this case need to both acquire or create content, and arrange the elements properly.

2.3. Example Annotation

To annotate the dataset, we collect natural language instructions paired with each slide. For each slide, we create three versions corresponding to the three setups in §2.2.

Detailed Instructions with Images To produce detailed instruction with images, we use a scalable approach combining human-written examples and model-generated annotations. For each slide deck, we first write instructions for three example slides manually — including all necessary information (content, layout, formatting) to reproduce the slide, and providing paths to the images used in the slide (e.g., `media/image_0.png`), as shown in Figure 2 (top). We then use these (human-written instruction, reference slide) pairs as few-shot examples to prompt LLM (specifically, `gpt-4o-mini`) to generate natural language instructions for each slide in the current slide deck.[*] Further, for the test set, we manually examined and refined the instructions by correcting incorrect specifications, adding missing content, and removing unnecessary or untrue content.

Detailed Instructions Only To produce detailed instruction only, we replace the image paths (e.g., `media/image_0.png`) with the natural language descriptions of the images(`‘‘an artistic, colorful background’’`). These descriptions are generated by `gpt-4o-mini`. For the test set, we manually refine the instructions to ensure that they do not refer to unavailable image paths (e.g., removing phrases like "use the provided images"), as shown in Figure 2 (middle).

High-Level Instructions To create high-level instructions, we start with a similar approach by manually annotating three examples and then prompting the model to generate for all slides. Human-written instructions only provide a topical description of the slide and intentionally leave out specific content or layout details. This process ensures that the generated instructions remain concise and general, as shown in Figure 2 (bottom).

Overall, the instructions have an average of 115.6, 118.3, and 26.6 words under *detailed instruction with images*, *detailed instruction only*, and *high-level instructions* settings respectively, accompanied by an average of 1.1, 0.0, and 0.0 provided images.

3. Evaluation Metrics

In this section, we describe the evaluation metrics that we designed for SLIDESBENCH. We propose two sets of evaluation metrics: reference-based metrics for measuring models' instruction-following abilities (§3.1), and reference-free metrics to examine the design quality of model-generated content (§3.2). We also use executability to examine the success rate of each model (§3.3).

3.1. Reference-Based Metrics

Inspired by Design2Code [38] metrics, we implement four dimensions to examine the similarity between the model-

[*] Including the three slides with human-written instructions, to ensure instructions for all slides are consistent in style and specificity.

produced slides and the reference slide.

Element matching For the slide layout, we measure the total sizes of matched elements (in generated and reference slides) divided by the total sizes of all element, where each textbox, image, or shape constitutes an element. More concretely, we accurately parse out each element in the generated and reference slides, and compute their maximum matching using the `match` library.

Content similarity For each pair of matched elements, we compute their content similarity. If the reference element is text, we calculate the textual similarity using the cosine similarity of the embeddings produced with sentence-transformer with the default `all-MiniLM-L6-v2` model [29]. If the reference element is an image, we calculate the CLIP score [19] of the image in two elements. We report the average content similarity across all matched element pairs, if either element contains a non-empty text string or an image component.

Color similarity We also measure the coloring similarity using the CIEDE2000 color difference formula [26], to quantify the perceptual difference between the colors. For every matched element pair, we measure the text font color similarity and element background color (if any). We additionally measure the color similarity between the background color of generated and reference slides.

Position similarity In addition to content and formatting, we also calculate the positional similarity between each pair of matched elements. More concretely, we follow Si et al. [38] to normalize the element coordinates to $[0, 1]$ by the slide page length and width. We compute the Manhattan distance between the elements and formulate positional similarity as $sim(r, g) = 1 - max(abs(x_r - x_g, y_r - y_g))$.

Note that a low text, color, or position similarity score could come from differences in text, color, and positions, or derivative errors caused by the inaccurate element-matching process (e.g., it may match the title box in the generated slide to a content textbox in the reference slide, which has different content or coloring requirements).

3.2. Reference-Free Metrics

A well-designed slide generated by models may look very different from the reference slide. Therefore, we also propose four reference-free evaluation metrics, to independently assess the design quality of model-generated slides. To establish the metrics, we surveyed a wide range of literature on slide design principles [5, 9, 34, 39, 44], and summarized four major points as below and detailed in Table 1:

Text Using concise texts is important for slides to engage with the audience. An ideal slide should have a clear title, concise main content, and readable formatting.

Image Using appropriate visuals can engage audiences. We hence measure if models can find high-quality images and properly use them to enhance the slide quality.

Layout Slide layout is crucial to create visual balance. We examine whether all elements are within the slide, have no overlap, and align properly with the relevant elements.

Metric	Criteria
Text	The title should be simple and clear to indicate the main point. For main content, avoid too many texts and keep words concise. Use a consistent and readable font size, style, and color.
Image	Use high-quality images with a reasonable proportion.
Layout	Elements should be aligned, do not overlap, and have sufficient margins to each other. All elements should not exceed the page.
Color	Use high-contrast color especially between the text and the background. Avoid using high-glaring colors.

Table 1. Reference-free metrics, all evaluated in 0-5 scale.

Color Vivid and consistent color use in slides can help deliver insights. We check if the slide uses high-contrast colors to facilitate visibility, and avoid high-glaring colors to discourage user engagement.

Validation of Reference-Free Evaluation For all the metrics, we provide the image version of the slide and ask the `gpt-4o` model to produce a score between 0–5. To examine the reliability of this model-based evaluation, we conduct a human study and compare the intraclass correlation coefficient (ICC) between two human annotators and model evaluation, on all ground-truth slides. Our examination gives high ICC scores across all four metrics: 73.8%–85.3%, which are well within the range of what is typically considered "high agreement". In experiments in later sections, we scale these 0-5 scale scores to the 0–100 range to enable comparisons on this more standard scale. [†]

3.3. Executability

Particularly for methods based on code generation (§4.1), we additionally measure the execution success rate to account for invalid programs. Concretely, we count the percentage of successfully executing programs generated by models among all examples. We report reference-based and free scores for executable slides only, to fairly compare their design quality. But we report 'Overall' scores for all slides by assigning zeros to non-executing slides, to account for execution failures. We report all metrics for successfully executing and all slides in §E.

4. Method

We introduce our main method — slide generation via code generation, optionally using our SLIDESLIB toolkit (§4.1). Then, we present AUTOPRESENT, trained on $7k$ slides, that achieves performance on par with strong GPT model (§4.2).

4.1. Slides via NL-to-Code Generation

Generating Python Programs Given natural language instructions in §2, models are tasked with generating Python programs using publicly available libraries such as

[†] We still evaluate with 0-5 scale to maintain a robust, human-aligned evaluation process.

python-pptx. The model receives two (natural language instruction, Python program) pairs as in-context examples, followed by the test instruction, and generates a Python program which is then executed and will ideally yield a PPTX file containing the requested slide.

Generating Programs with SlidesLib Nonetheless, the programs above could be very long and complex (170 lines on average), which could be challenging for models to generate entirely correctly, as shown in previous work [14]. To address this, we design SlidesLib, a library that provides easier-to-use interfaces for several common actions such as setting a title or setting background color. Using SlidesLib, the average program length is reduced to 13 lines, significantly easing the generation task. As shown in Table 2, SlidesLib includes 4 functions for basic operations and 3 functions for image search and generation, these functions allow models to produce more concise and modular programs. To enable the model to generate programs using SlidesLib, we follow the visual programming method [43] by providing a prompt that includes the documentation of the functions and two in-context examples. See more SlidesLib details in §B.

Function	Description
add_title	Insert a title in the slide.
add_text	Insert text at a specific location.
add_bullet_points	Insert a textbox with bullet points.
add_image	Insert image at a specific location.
generate_image	Call an image generator (Dall-E 3) given a query.
search_image	Search for an image on a search engine (Bing).
search_screenshot	Display a query on a web browser (Google Chrome) and take a snapshot of the search result.

Table 2. Basic (top) and image-specific (bottom) functions provided by SlidesLib.

4.2. AutoPresent

Using the slides in the training set of SlidesBench, we construct (natural language instruction, program) pairs to form training data to train an open-sourced 8B model, AutoPresent. This model is based on the Llama-3.1-8B-Instruct and trained using LoRA [21] with a rank of 128. To create (natural language instruction, program) training pairs, we generate two versions of canonical program solutions for each slide:

(i) Basic Python Programs We derive canonical programs (that is, programs that can be executed to reproduce the slide) without SlidesLib. To do this, we manually design an extraction script that (i) extracts each element (e.g., text and image) on the given slide, and (ii) produces a rule-based program that adds each element to the slide. After extracting and adding each element to the slide, the resulting program accurately reproduces the original slide.

(ii) SlidesLib Python Programs We also generate canonical programs using SlidesLib, by transforming snippets from the programs above into SlidesLib function calls. To reproduce images in *detailed instruction only* and *high-level instructions* settings, we generate a short caption for each image and provide it to GPT-4o to generate the program for producing that image using search_image or generate_image functions. More details of this automatic program generation process are in §B.2.

After obtaining three instructions and two program versions for each example, we construct four versions of the training data, each with $7k$ examples:

1. (detailed instruction with images, python program)
2. (detailed instruction with images, SlidesLib program)
3. (detailed instruction only, SlidesLib program)
4. (high-level instructions, SlidesLib program)

These training sets allowed us to train four specialized models that address different challenges, which we report in Table 3 (1,2) and Table 4 (3,4).

4.3. Iterative Refinement

Slide generation is by nature an iterative process and often requires visual-based refinements after the first draft. To enable models to refine slides as humans do, we explore an iterative refinement procedure, where the model is tasked to self-refine the slide it generated. Specifically, in the setting using SlidesLib, we provide GPT-4o (capable of consuming slide images) with the original language instruction, the program it generated in the first pass, and a snapshot of the rendered slide; the model is then asked to generate a new program based on these information to refine the slide quality by tweaking colors, spacing, and other aspects of the slide. See the prompts of this process in §D.

5. Experiments and Results

We first introduce the experimental setup (§5.1), then present the results under various scenarios (§5.2).

5.1. Experimental Setup

Code Generation Approaches For code generation approaches, we sample $n = 3$ responses and iteratively go through them, using the first successfully executing program as the final output of the model. If none of the n responses execute successfully, we count it as an execution failure. In addition to AutoPresent (§4.2), we benchmark several LMs out-of-the-box, including open-weights Llama 3.1 (8B, Instruct), the code generation models DeepseekCoder-7B-v1.5 and CodeLlaMa-7B-Instruct, the vision-language LLaVa v1.5 model (with a Vicuna-7B-v1.5 LM backbone); and the proprietary GPT-4o model (the gpt-4o-2024-08-06 checkpoint).

Method	Execution%	Reference-Based				Reference-Free				Overall
		element	content	color	position	text	image	layout	color	
Reference	100.0	–				59.7	81.5	73.5	65.7	–
End-to-end Image Generation										
Stable-Diffusion*	**100.0**	74.5	33.4	9.0	75.0	19.6	45.1	36.9	40.5	48.0
DALLE 3*	**100.0**	75.5	39.9	9.2	76.1	32.7	87.3	56.7	53.4	50.2
Code Generation w/o SLIDESLIB										
LLaVA (7B)	11.3	61.9	**97.3**	6.2	70.8	41.6	**100.0**	29.2	25.7	6.1
CodeLLaMA (7B)	5.1	63.6	94.0	11.2	74.0	52.0	43.0	48.0	40.0	3.1
LLaMA (8B)	2.1	74.0	94.6	12.5	**81.2**	50.0	8.3	50.0	50.0	1.3
GPT-4o	89.2	83.3	91.6	10.5	77.0	51.9	72.8	53.7	54.7	55.1
AUTOPRESENT (ours)	79.0	67.7	79.7	10.9	75.9	45.3	62.7	54.2	60.9	45.2
Code Generation w/ SLIDESLIB										
LLaVA (7B)	20.0	80.5	80.5	3.5	64.0	37.5	48.0	29.5	43.5	9.7
CodeLLaMA (7B)	48.7	80.3	89.8	9.4	69.3	45.9	66.8	45.1	49.9	30.3
LLaMA (8B)	54.4	78.3	91.2	7.5	69.5	46.0	68.2	47.6	53.1	33.5
GPT-4o	86.7	**86.2**	92.5	12.7	76.3	**54.6**	83.7	**70.5**	59.4	**58.0**
AUTOPRESENT (ours)	84.1	84.2	92.2	**18.1**	67.2	47.8	73.2	58.6	**64.7**	55.0

Table 3. **Results with *detailed instructions with images*.** We found that small models like LLAVA (7B) and LLAMA (8B) can barely generate any slides, while AUTOPRESENT (8B) generates slides on par with GPT-4o. All the models still underperform humans.

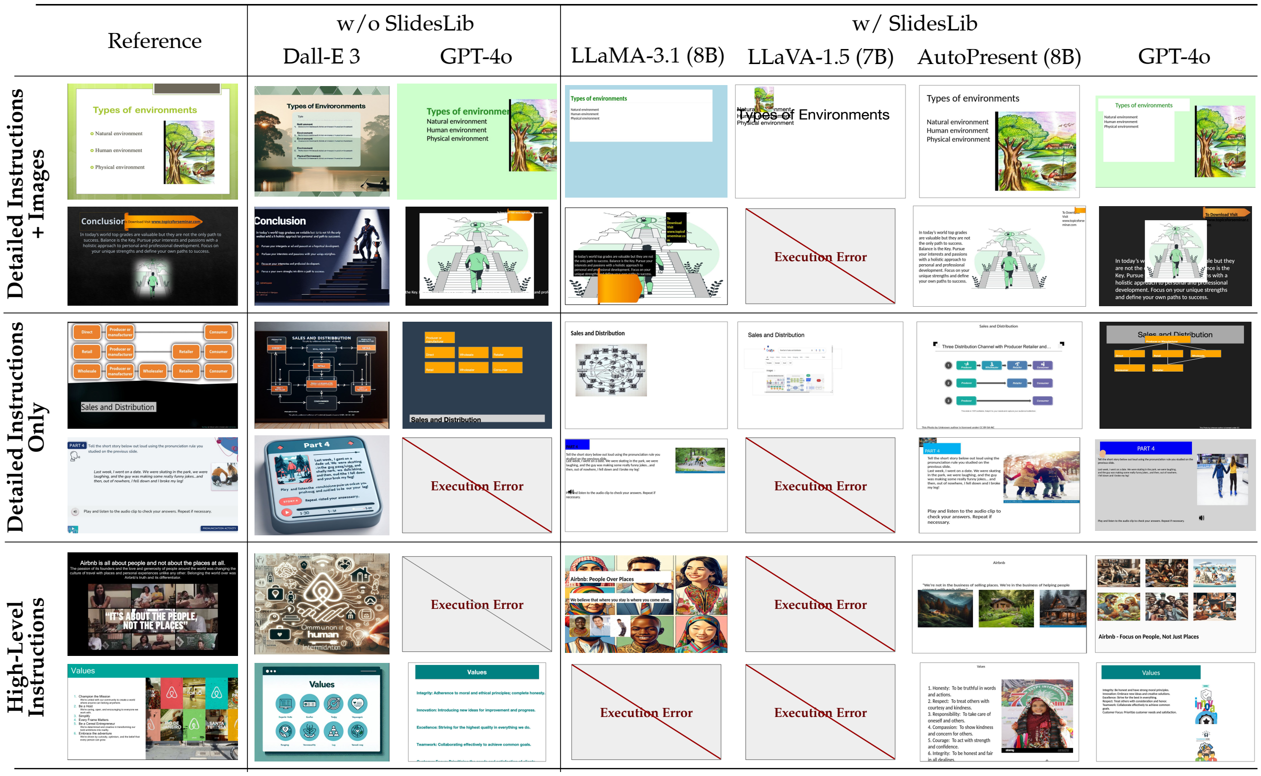


Figure 3. **Examples of slides generated by different methods in three scenarios.** End-to-end image generation methods fail to generate structured slides. Small open-sourced models like LLAMA and LLAVA can barely generate any usable slides, while AUTOPRESENT produces quality slides. Adding SLIDESLIB improves GPT-4o's performance on *detailed instruction only* and *high-level instruction* tasks.

End-to-End Image Generation We compare code generation with end-to-end neural image generation methods, which are a common way to produce visuals. These methods are good at creating scenic or artistic images, but may be imprecise in content (esp. text) and do not support easy further modification by users. We benchmark Stable-Diffusion 2 [32] and DALL-E 3 [8] by asking them to output slides given the natural language instructions. We adjust

Method	Detailed Instructions Only				High-Level Instructions			
	exec	ref-based	ref-free	overall	exec	ref-based	ref-free	overall
End-to-End Image Generation								
SD2	**100.0**	48.0	35.5	48.0	**100.0**	47.7	31.5	47.7
DALLE 3	**100.0**	50.2	57.5	50.2	**100.0**	50.7	53.6	52.2
Code Generation w/o Library								
LLaVA	17.9	56.9	47.4	9.3	19.5	50.2	47.3	9.5
DeepseekCoder	2.6	59.6	37.5	1.3	22.6	57.6	43.0	11.4
CodeLLaMA	-	-	-	-	21.0	57.9	54.4	12.2
LLaMA	4.6	61.4	35.1	2.8	8.7	55.6	50.1	4.8
GPT-4o	50.3	**66.8**	50.0	28.7	70.8	**60.3**	57.0	39.7
Code Generation w/ Expert-Designed Library								
LLaVA	17.4	58.2	33.8	8.0	25.1	50.1	36.7	10.9
DeepseekCoder	24.1	57.1	43.4	12.1	31.8	53.0	48.7	16.2
CodeLLaMA	-	-	-	-	35.9	56.6	53.4	20.3
LLaMA	60.5	61.7	56.6	37.4	76.9	56.8	58.3	43.7
GPT-4o	87.7	64.2	**65.8**	**56.3**	97.4	60.1	**71.2**	**58.5**
AUTOPRESENT	89.2	61.9	58.7	55.2	86.6	55.2	61.5	47.8

Table 4. Results under *detailed instruction only* and *high-level instructions* settings. We assign 100% execution success rates for all end-to-end image generation methods because they do not generate programs and would not suffer from execution errors.

our reference-based evaluation procedure by first segmenting slide images into elements using Tesseract OCR[40] and further parse out the texts of the elements, then applying the default calculation process as in §3. For the *detailed instruction with images* setting, we also report the results of the end-to-end image generation methods, marked with a "*" indicating that they don't actually use image inputs.

5.2. Quantitative Results and Analysis

Table 3 shows the result of *detailed instruction with images* scenario and Table 4 shows the result of *detailed instruction only* and *high-level instructions* scenarios.

In the top row of Table 3, we first measure the scores of the reference slides, which shows that the quality of the human-created slides is among the highest.

Compared to the scores achieved by GPT-4O, smaller open-source models such as LLAMA 3.1 and LLAVA barely produce any working slides out of the box. Although the significant gaps of 49.9–55.0 points exist in the *detailed instruction with images* setting, this gap shrinks to 22.2–34.6 when no visuals are provided a priori, in *detailed instruction only* and *high-level instructions* scenarios (Table 4). This demonstrates significant challenges in obtaining images in slides. In contrast to the low performance of open-weight models out-of-the-box, AUTOPRESENT's performance approaches that of GPT-4O.

End-to-End Image Generation When no visuals are provided, end-to-end image-generation methods perform worse than the best code-generation approaches in both the reference-based and reference-free metrics, especially in generating accurate content. These methods also often produce creative figures without aligning with the design principles of slides, indicating its poorer controllability.

Effect of Library SLIDESLIB brings observable gains in LLAMA and LLAVA in all three scenarios by at most 34.0 points; and similarly increases the strong GPT-4O performance across scenarios, especially when no images are provided. This suggests the benefits of generating more modular and concise programs for structured visual design.

VLM vs. LLM When no helper functions are presented, the one VLM that we tested (LLAVA) outperforms its LLM counterpart LLAMA in all scenarios by 5.1–7.5 points. However, LLAVA shows limited ability in using functions presented in context, as demonstrated by the large margin the library-augmented LLAMA has over LLAVA (12.1–26.2). All LLMs (LLAMA, GPT-4O) perform worse when the instructions become less specified (*detailed instruction with images* → *detailed instruction only* → *high-level instructions*). Nonetheless, SLIDESLIB can greatly mitigate this degradation due to the loss of input specificity, and help models produce better outcomes across all three scenarios.

5.3. Qualitative Case Study

We illustrate several models-produced slides in Figure 3. For end-to-end image generation methods, the design is more creative and often more attractive, but the text does not constitute meaningful words, or even the characters themselves are not valid.

On the other hand, code generation methods, especially weaker LLAMA and LLAVA models, suffer more from visual layout — elements often overlap with each other or exceed the canvas, making it challenging for the audience to obtain all information clearly.

In contrast, AUTOPRESENT generates slides with appropriate layouts without undesirable element overlaps. In addition, they better follow the user instructions and are not overly creative like the image generation methods.

5.4. Perceptual Evaluation

We performed a qualitative evaluation on 10 randomly selected slides from each domain generated by GPT-4o, Llama-8B, and AUTOPRESENT under the *detailed instruction with images* and *detailed instruction only* settings. We also add the ground-truth reference slide to evaluate the performance gap between current models and human slide creators. We shuffle these slides and ask the annotators to rank each slide from 1-5 based on how likely they would be to use the slide. For the *detailed instruction with images* setting, we collect 25 responses in total, and for *detailed instruction only*, we collect 16 responses in total. We provide more details of the evaluation process in §F.

The result is shown in Figure 4. By performing the paired t-test, we found differences between the models pairs in terms of user preferences, as shown in Table 13: (1) In both settings, AUTOPRESENT and GPT-4o perform statistically significantly better than LLAMA. (2) In *detailed instruction with images* setting, GPT-4o and AUTOPRESENT has no significant differences (3) In the *detailed instruction only* setting, AUTOPRESENT is slightly worse than gpt-4o, aligning with our quantitative evaluations in Table 4. All three models still have an overall performance gap compared with human-designed slides, indicating room for improvement on the slide generation task.

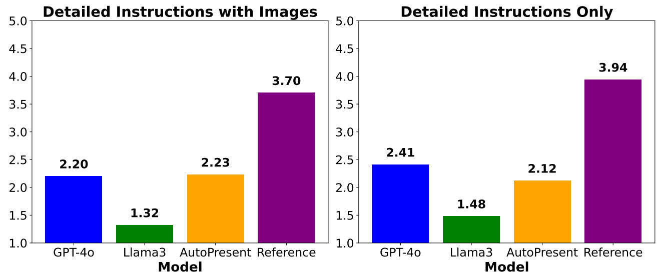


Figure 4. **Perceptual evaluation results on detailed instruction (1) with images and (2) only settings.** We ask the users to score the quality of each slide from 1-5 and report the average score of each model. The user reported preference on GPT-4o and AUTOPRESENT compared with LLAMA, while still having a gap with human-designed slides.

Iteration	Detailed + Images	Detailed Only	High-Level
0	58.0	56.3	58.5
1	59.5	59.5	59.8
2	59.3	**60.1**	61.3
3	**60.1**	59.4	**61.4**

Table 5. **Overall scores after applying multi-rounds of refinement in the three scenarios**, demonstrating that refinement boosts performance in all three scenarios.

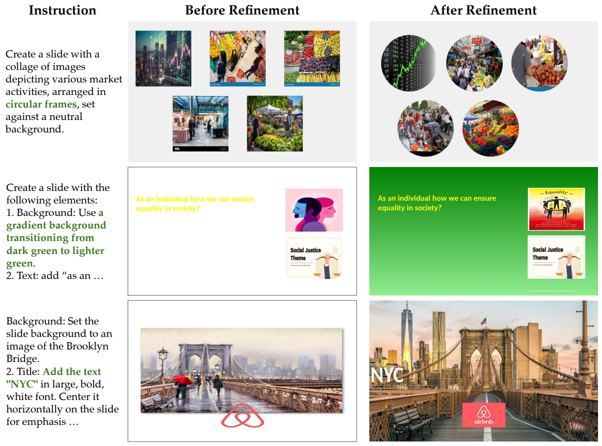
5.5. Result after Iterative Refinement

Finally, as shown by Table 5, we find that refinement improves model performance on all three challenges. By doing an ablation on the round of iterations, we find that while continued refinement often increases the scores, the first iteration usually gives the biggest performance improvement. We present representative cases after doing one round of refinement in Figure 5, which indicates that refinement can improve content layout and detailed controls on coloring and sizing.

6. Related Work

Language and Vision Model-Based Agents Agents based on large language models (LLMs) [2, 11] and vision-language models (VLMs) [4, 51] have been widely adopted in various tasks such as web navigation [24, 52, 54], software engineering [49, 50], and web development [27, 38]. Creation of presentation materials is another common task [10] that has both similarities and differences from these more widely examined tasks.

Generating Programs for Vision Tasks End-to-end image generation models such as diffusion [20, 32, 48] and GAN [15, 35] are widely used at producing scenic images, yet falling short on more structured visuals such as websites and slides [38]. Generating programs (i.e., image-editing actions) is a useful means to get structured visuals [14, 18, 41, 43, 45], including Tikz figures [6, 7], SVG [31, 37], posters [51], and user interfaces [28, 38]. However, they often require detailed inputs and are limited to specific, simple figure types, so they are still far from creating complex, editable presentation slides from scratch. Our work extends this line of research by formulating and benchmarking the natural-language-to-slide generation task.



Create a slide with a collage of images depicting various market activities, arranged in **circular frames**, set against a neutral background.

Create a slide with the following elements:
1. Background: Use **a gradient background transitioning from dark green to lighter green**.
2. Text: add "as an ...

Background: Set the slide background to an image of the Brooklyn Bridge.
2. Title: **Add the text "NYC"** in large, bold, white font. Center it horizontally on the slide for emphasis ...

Figure 5. **Auto-refinement results with GPT-4o**, where the model further addresses some previously neglected instructions (marked in green), such as shape, background color, and text.

Automatic Slide Generation Previous works on slide creation mostly focus on basic extraction from provided documents [12, 22, 23, 36, 42] or having models generate content given a topic [1, 3, 47] without addressing how to organize content visually. More recently, some benchmarks [17, 53] and methods [13] have emerged that follow detailed instructions for slide editing (e.g., adjust the font size of the title from 20 to 24) of an existing slide. In contrast, we synthesize more complex and structured programs that can generate slides from scratch, including content creation, visual arrangement, and fine-grained editing, instead of refining an existing slide.

7. Conclusion and Limitations

In this work, we address the challenge of creating structured visuals from scratch. Specifically, we introduced SLIDESBENCH, the first benchmark for automatic slide generation with evaluation metrics based on and free of reference slides. We benchmark multiple end-to-end image and program generation approaches, and demonstrate that AUTOPRESENT with SLIDESLIB achieves comparable performance with the top GPT-4O model. Our further exploration in iterative refinement also reveals certain effectiveness in self-refinement. This work is an initial step towards automated generation of structured visuals. Specifically, it focuses on single-slide generation and produces full slide code in a single pass, without leveraging iterative design workflows. Future research could address these limitations by expanding to full slide decks, adopting gradual and interactive slide generation, and incorporating slide-specific features like animations. Further, integrating more design principles, such as optimizing for attention capture and information clarity, would be crucial for making generated slides more impactful and effective.

Acknowledgment

We would like to thank Yutong Bai for helping us draft Figure 1 and providing feedback on the paper, David Chan for providing detailed suggestions on the introduction, and Frank Xu and Sean Welleck for discussions at the initial stage of this project, and Junyi Zhang and Haven Feng for feedbacks on the project.

References

[1] Gamma - ai-powered document creation and storytelling platform. Accessed: 2024-11-14. 8

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2, 8

[3] Shaikh Mostafa Al Masum, Mitsuru Ishizuka, and Md Tawhidul Islam. 'auto-presentation': a multi-agent system for building automatic multi-modal presentation of a topic from world wide web information. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 246–249. IEEE, 2005. 1, 8

[4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022. 8

[5] Angie Arriesgado. 45 tips to speed up your powerpoint design workflow, 2019. 2, 4

[6] Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. *ArXiv*, abs/2310.00367, 2023. 8

[7] Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. *arXiv preprint arXiv:2405.15306*, 2024. 8

[8] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2(3):8, 2023. 1, 2, 6

[9] The LinkedIn community. How do you use design principles and best practices to evaluate and improve your slide layout and formatting?, 2024. 2, 4

[10] decktopus. Top presentation statistics for 2021, 2021. 1, 8

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 2, 8

[12] Tsu-Jui Fu, William Yang Wang, Daniel McDuff, and Yale Song. Doc2ppt: Automatic presentation slides generation from scientific documents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 634–642, 2022. 8

[13] Apurva Gandhi, Thong Q Nguyen, Huitian Jiao, Robert Steen, and Ameya Bhatawdekar. Natural language commanding via program synthesis. *arXiv preprint arXiv:2306.03460*, 2023. 8

[14] Jiaxin Ge, Sanjay Subramanian, Baifeng Shi, Roei Herzig, and Trevor Darrell. Recursive visual programming. In *European Conference on Computer Vision*, pages 1–18. Springer, 2025. 2, 5, 8

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 8

[16] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. 2

[17] Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Nan Duan. Pptc benchmark: Evaluating large language models for powerpoint task completion. *arXiv preprint arXiv:2311.01767*, 2023. 8

[18] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. 8

[19] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021. 4

[20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 8

[21] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021. 5

[22] Yue Hu and Xiaojun Wan. Ppsgen: Learning-based presentation slides generation for academic papers. *IEEE transactions on knowledge and data engineering*, 27(4):1085–1097, 2014. 8

[23] Min-Yen Kan. Slideseer: A digital library of aligned document and presentation pairs. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 81–90, 2007. 8

[24] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024. 8

[25] Wenyuan Kong, Zhaoyun Jiang, Shizhao Sun, Zhuoning Guo, Weiwei Cui, Ting Liu, Jianguang Lou, and Dongmei Zhang. Aesthetics++: Refining graphic designs by exploring design principles and human preference. *IEEE Transactions on Visualization and Computer Graphics*, 29(6):3093–3104, 2022. 1

[26] M Ronnier Luo, Guihua Cui, and Bryan Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre*

Foundation, Colour Society of Australia, Centre Français de la Couleur, 26(5):340–350, 2001. 4

[27] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021. 8

[28] Yi-Hao Peng, Faria Huq, Yue Jiang, Jason Wu, Xin Yue Li, Jeffrey P Bigham, and Amy Pavel. Dreamstruct: Understanding slides and user interfaces via synthetic data generation. In *European Conference on Computer Vision*, pages 466–485. Springer, 2025. 8

[29] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. 4

[30] Garr Reynolds. *Presentation Zen: Simple ideas on presentation design and delivery*. New Riders, 2011. 1

[31] Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556*, 2023. 8

[32] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 1, 2, 6, 8

[33] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 2

[34] Hitesh Sahni. Presentation design: A step-by-step guide, 2024. 2, 4

[35] Axel Sauer, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis. In *International conference on machine learning*, pages 30105–30118. PMLR, 2023. 8

[36] Athar Sefid and Jian Wu. Automatic slide generation for scientific papers. In *Third International Workshop on Capturing Scientific Knowledge co-located with the 10th International Conference on Knowledge Capture (K-CAP 2019), SciKnow@ K-CAP 2019*, 2019. 1, 8

[37] Pratyusha Sharma, Tamar Rott Shaham, Manel Baradad, Stephanie Fu, Adrian Rodriguez-Munoz, Shivam Duggal, Phillip Isola, and Antonio Torralba. A vision check-up for language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14410–14419, 2024. 8

[38] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering? *arXiv preprint arXiv:2403.03163*, 2024. 3, 4, 8

[39] Think Outside The Slide. Latest annoying powerpoint survey results, 2019. 2, 4

[40] Ray Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, pages 629–633. IEEE, 2007. 7

[41] Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular visual question answering via code generation. *arXiv preprint arXiv:2306.05392*, 2023. 8

[42] Edward Sun, Yufang Hou, Dakuo Wang, Yunfeng Zhang, and Nancy XR Wang. D2s: Document-to-slide generation via query-based text summarization. *arXiv preprint arXiv:2105.03664*, 2021. 8

[43] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898, 2023. 5, 8

[44] Utrecht University. Four principles for making a good powerpoint presentation, 2024. 2, 4

[45] Zhiruo Wang, Graham Neubig, and Daniel Fried. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Forty-first International Conference on Machine Learning*, 2024. 8

[46] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. 1

[47] Thomas Winters and Kory W Mathewson. Automatically generating engaging presentation slide decks. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 127–141. Springer, 2019. 8

[48] Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svgdreamer: Text guided svg generation with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4546–4555, 2024. 8

[49] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. 1, 8

[50] John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, et al. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*, 2024. 8

[51] Tao Yang, Yingmin Luo, Zhongang Qi, Yang Wu, Ying Shan, and Chang Wen Chen. Posterllava: Constructing a unified multi-modal layout generator with llm. *arXiv preprint arXiv:2406.02884*, 2024. 2, 8

[52] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, pages 20744–20757. Curran Associates, Inc., 2022. 8

[53] Zekai Zhang, Yiduo Guo, Yaobo Liang, Dongyan Zhao, and Nan Duan. Pptc-r benchmark: Towards evaluating the robustness of large language models for powerpoint task completion. *arXiv preprint arXiv:2403.03788*, 2024. 8

[54] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. 1, 8