SlideCoder: Layout-aware RAG-enhanced Hierarchical Slide Generation from Design

Anonymous ACL submission

Abstract

Manual slide creation is labor-intensive and requires expert prior knowledge. Existing natural language-based LLM generation methods struggle to capture the visual and structural nuances of slide designs. To address this, we formalize the Reference Image to Slide Generation task and propose Slide2Code, the first benchmark with difficulty-tiered samples based on a novel Slide Complexity Metric. We introduce SlideCoder, a layout-aware, retrieval-augmented framework for generating editable slides from reference images. SlideCoder integrates a Color Gradient-based Segmentation algorithm and a Hierarchical Retrieval-Augmented Generation method to decompose complex tasks and enhance code generation. We also release SlideMaster, a 7B open-source model fine-tuned with improved reverse-engineered data. Experiments show that SlideCoder outperforms state-of-the-art baselines by up to 40.5 points, demonstrating strong performance across layout fidelity, execution accuracy, and visual consistency. Our code is available at `https://anonymous.4open.science/r/SlideCoder`.

1 Introduction

Slide creation is essential in academic and professional communication for visually conveying complex ideas. However, manual design is labor-intensive and time-consuming (Al Masum et al., 2005). While templates offer some relief, they enforce fixed layouts and styles, limiting flexibility.

Recent progress in Large Language Models (LLMs) (Nam et al., 2024; Ge et al., 2023) has sparked interest in automatic slide creation. AutoPresent (Ge et al., 2025), an early study on the Natural Language (NL) to slide generation task, fine-tunes a LLAMA-based model (Grattafiori et al., 2024) on the diversified SLIDESBENCH dataset. It translates NL instructions into Python code, which invokes SLIDESLIB, a high-level API built on python-pptx (Canny, 2023), to construct each slide. This pipeline reduces manual effort and streamlines design workflows.

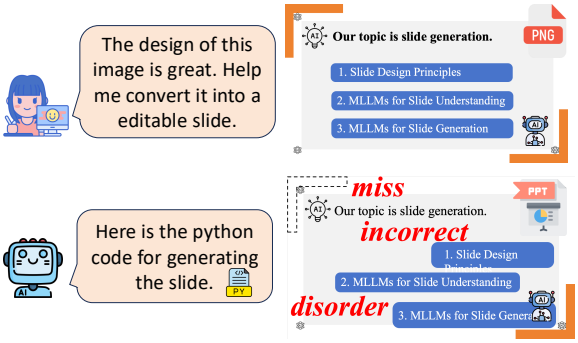

Figure 1: Illustration of slide generation scenarios from design and mistakes made by MLLMs.

Despite Autopresent's capability to generate slides from natural language input, several significant challenges remain unaddressed.

First, natural language inherently lacks an accurate description of slide visual design (e.g., color, layout, and style) and users sometimes directly input the design image for slide generation. For example, as shown in Figure 1, a user sees a nice design from non-editable slides (png and pdf format) or other source like webpage design, and hopes to convert it into an editable slide (pptx format). Or the user lacks the skills to make slides, they can generate the slide by input their design image. In these scenarios, the Multimodal Large Language Models (MLLMs) are needed to understand the design and generate slides.

Second, MLLMs face limitations when handling complex slides, particularly those incorporating diverse element types and high element density. As illustrated in Figure 1, these discrepancies can be divided into three categories: *miss*, which stands for the complete omission of certain visual or textual elements (e.g., the top left corner of the shape is missing); *incorrect*, referring to de-

viations in visual styles or attributes from those specified or expected in the reference slides (e.g., title is not bold); and *disorder*, which describes significant differences in spatial arrangements and alignment of elements compared to the original layout (e.g., the three subheadings are not properly positioned and aligned.).

Third, MLLMs' insufficient comprehension of the python-pptx library leads to the generation of syntactically invalid or non-executable code. Autopresent (Ge et al., 2025) attempts to address this issue by constructing SLIDESLIB, a simplified library built upon python-pptx, encapsulating commonly used operations into a set of high-level APIs. However, this operation inherently restricts the flexibility and comprehensiveness of slide generation. Specifically, SLIDESLIB currently supports only five basic operation types, which neglects more intricate layouts and design requirements commonly encountered in realistic scenarios. Consequently, presentations produced by this approach tend to be overly simplistic, inadequately capturing complex human intentions and detailed visual expectations.

To address the aforementioned limitations, we introduce SlideCoder, a layout-aware RAG-enhanced hierarchical slide generation framework, which can understand the complex slides and python-pptx library accurately. First, we formulate a novel task, **Reference Image (RI) to slide generation**, i.e., automatically generating the code for replicating the slide, which is visually consistent with RI. To evaluate the performance of SlideCoder under complex slide scenarios, we propose a novel Slide Complexity Metric (SCM), and construct a new benchmark Slide2Code with different difficulty levels based on SCM. Second, we develop a novel **Color Gradients-based Segmentation** algorithm (**CGSeg**) that effectively decomposes slide images into semantically meaningful regions. Besides, we propose the **Layout-aware Prompt**, which integrates the position information of elements to enhance MLLM's understanding of slide layout. Third, we propose a novel **Hierarchical Retrieval-Augmented Generation (H-RAG)-based Code Generation** method, which employs a dual-level retrieval-augmented knowledge base (Cuconasu et al., 2024; Fan et al., 2024) to explicitly enhance MLLMs' understanding of the python-pptx library. At the higher level, a Shape Type Knowledge Base (TS-KB) systematically classifies slide elements and standardizes their descriptions using python-pptx API terminologies. At the lower level, a Operation Function Knowledge Base (OF-KB) captures precise syntactic patterns and invocation paradigms of python-pptx library functions.

To further enhance the MLLM's ability to generate high-quality slides, we build a PPTX reverse-engineering tool to construct high quality training data for fine-tuning a 7B model SlideMaster based on Qwen-VL-7B (Bai et al., 2025), which can approaches the performance of the closed-sourced model GPT-4o (Achiam et al., 2023). Our contributions are summarized as follows:

- We define reference image (RI) to slide generation task and propose a novel Slide Complexity Metric (SCM), based on which we construct Slide2Code, the first difficulty-leveled benchmark with 300 samples.
- We propose SlideCoder, which consists of a novel Color Gradients-based Segmentation algorithm (CGSeg), a Layout-aware Prompt and a Hierarchical Retrieval-Augmented Generation (H-RAG)-based Code Generation method for enhancing the MLLM's understanding on the complex slides and python-pptx library.
- We train SlideMaster, a 7B open-source model approaching the performance of GPT-4o. To enable effective fine-tuning, we also build a comprehensive PPTX reverse-engineering tool for precise code generation.

2 Related Work

2.1 Multimodal Large Language Models for Code Generation

The multimodal large model demonstrates excellent capabilities in visually rich code generation scenarios, such as UI code generation (Xiao et al., 2024; Wan et al., 2024a; Yun et al., 2024; Wan et al., 2024b), SVG code generation (Rodriguez et al., 2025; Nishina and Matsui, 2024; Wu et al., 2024; Xing et al., 2024), and visually rich programming questions (Li et al., 2024; Zhang et al., 2024a; Ma et al., 2025). However, MLLMs are not yet capable of plug-and-play use across tasks and still produce subtle errors, therefore, some studies explore their code repair abilities (Yang et al., 2024; Yuan et al., 2024; Zhang et al., 2024b).

2.2 Slide Generation and Understanding

Previous work on slide generation has predominantly focused on basic content extraction from

input documents. With the recent advancements in large language models (Fu et al., 2022; Hu and Wan, 2014; Kan, 2007; Sefid and Wu, 2019), several studies have begun to explore LLM-based slide generation. For example, (Zheng et al., 2025) utilizes LLMs to generate slides based on pre-defined slide templates and user-provided text. (Ge et al., 2025) introduces the task of natural language (NL) to slide code generation, aiming to organize visual slide content through textual input. However, its use of coarse-grained natural language descriptions and a native agent design significantly limits the quality of the generated slides.

3 Slide2Code Benchmark

We construct the Slide2Code benchmark to evaluate the performance of multimodal large language models (MLLMs) on the Reference Image (RI) to slide generation task. Each instance includes a reference slide image and its corresponding PPTX slide. Slide2Code enables comparison of MLLM backbones under varying complexity. §3.1 formally defines the task, §3.2 describes our unified complexity scoring system based on element quantity, diversity, and visual density, and §3.3 details data collection and sampling.

3.1 Task Description

This work addresses the task of Reference Image (RI) to slide generation, where the input is a slide's reference image I_0 and the goal is to generate Python code using the python-pptx library. Let F_0 denote the original slide file corresponding to I_0. Given a generation framework G and Multimodal Large Language Models (MLLMs) M, the generated code $C_g = G_M(I_0)$ can be executed to obtain a new slide file F_g, whose rendered image is denoted as I_g. As the original code C_0 for F_0 is unavailable, we assess the performance of G and M by comparing (I_0, F_0) and (I_g, F_g).

3.2 Slide Complexity Metric

To evaluate slide complexity, we propose a Tri-Metric Slide Complexity Metric (SCM) that integrates production difficulty and visual complexity. Due to the mismatch between visual appearance and construction effort, for example, inserting a visually complex image may require minimal operations. To adress this, we assess slides using: (1) element count, (2) element type count (e.g., textbox, placeholder), and (3) Element Coverage Ratio. The first two reflect operational cost, the third captures visual richness. Since reference complexity labels are not available, we evaluate the relative complexity of sample i within a collection $Y = \{1, 2, 3, ..., N\}$.

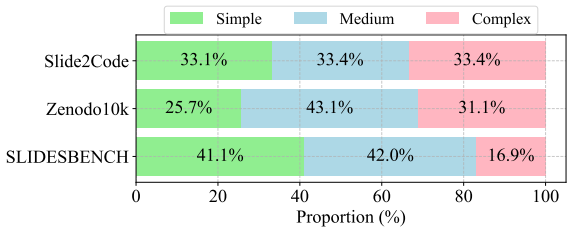

Figure 2: Proportion of samples across three levels in the Slide2Code, Zenodo10k, and SLIDEBENCH datasets.

Let c_i be the number of elements and e_i the number of distinct element types in sample i. The Element Coverage Ratio v_i is the proportion of activated color grids to total grids in the image of sample i, computed via the gradient-based segmentation algorithm CGSeg (see §4.1 for details).

Each raw dimension score $x_i \in \{c_i, e_i, v_i\}$ is normalized as $\tilde{x}_i = \sigma\left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\right)$, where μ and σ^2 denote the mean and variance over all samples in set Y, respectively. Here, $\sigma(\cdot)$ is the sigmoid function (Han and Moraga, 1995), and ϵ is a small constant for numerical stability. The final complexity score for slide i is computed via a weighted aggregation: $z_i = \alpha \cdot \tilde{c}_i + \beta \cdot \tilde{e}_i + \gamma \cdot \tilde{v}_i$, where $\alpha + \beta + \gamma = 1$ and the weights α, β, γ reflect the importance of production effort and visual complexity. This metric shows a strong correlation with human judgment, as detailed in Section §5.4.

3.3 Data Collection

To construct a comprehensive benchmark that captures diverse slide characteristics, we randomly sample approximately 32,000 Zenodo10k (Zheng et al., 2025) slide instances, the largest publicly available slide dataset, to construct the slide set Y as described in §3.2. To enhance diversity and allow comparative analysis, we additionally incorporate SLIDEBENCH samples in Y. This unified set is then used to calculate the normalized complexity scores z for all slides. KMeans algorithm is used to obtain three clusters, whose cluster centers are sorted in order of z to define the simple, medium, and complex levels. From each cluster, we randomly select 100 representative samples from Y to form the final Slide2Code benchmark.

Figure 2 shows that both Zenodo10k and

SLIDEBENCH contain a significantly larger proportion of simple and medium slides. In contrast, Slide2Code exhibits a more balanced composition across all three levels, allowing a more equitable evaluation of slide generation models under varying structural and visual complexities.

4 Methodology

In this section, we introduce SlideCoder, a unified end-to-end framework for generating Python-executable slide code from reference images (RIs). We assume a scenario where a user provides a design layout ("*Design* ") and embedded visual elements such as pictures or background images ("*Pictures* "). SlideCoder comprises three core modules. First, a **Color Gradients-based Segmentation** (CGSeg) algorithm segments the input *Design* into semantically meaningful regions. Second, a **Hierarchical Retrieval-Augmented Code Generation** module, consisting of three collaborative agents **Describer**, **Coder**, and **Assembler**, generates the slide code. Third, a **Layout-aware Prompt** mechanism enhances the Assembler agent to ensure spatial consistency and syntactic correctness. Finally, based on this framework, we fine-tune a 7B open-source model, named SlideMaster.

4.1 Color Gradient-based Segmentation

To reduce the difficulty of MLLM in understanding complex slide design, we proposed CGSeg, a recursive color gradient-based segmentation algorithm to divide slide design into blocks. As shown in Algorithm 1, CGSeg starts by dividing the input image (Figure 4a) into a grid and computing the Sobel magnitude for each block to measure the intensity of the color gradient (lines 4–5). Blocks with gradient magnitudes significantly higher than the median are marked as activated block (lines 6–14), as visualized in Figure 4b. To group visually coherent regions, CGSeg applies a flood-fill (Burtsev and Kuzmin, 1993) operation to the binary activation mask (line 15), identifying connected regions corresponding to sub-images (line 16), as shown in Figure 4c. These sub-images are further segmented recursively to ensure a hierarchical decomposition of the image I_m, along with the corresponding positional information p_m (lines 1–3 and 17–23), with the final segmentation result shown in Figure 4d. This recursive structure allows CGSeg to adaptively refine segment granularity based on local visual complexity, which is crucial for handling slides with heterogeneous layout densities.

Algorithm 1 Color Gradient-based Segmentation (CGSeg)

Require: Image I, Grid size g, Depth D, Max depth $D_{\max}$, Threshold T
Ensure: List of segmented sub-images
1: **if** $D = D_{\max}$ **then**
2: **return** $\emptyset$
3: **end if**
4: $G \leftarrow \textsc{Split}(I, g)$ `// g × g grid blocks`
5: $C \leftarrow \textsc{GradMag}(G)$ `// gradient magnitudes`
6: $C_{\text{mid}} \leftarrow \textsc{Median}(C)$
7: $M \leftarrow \mathbf{0}^{g \times g}$ `// binary mask`
8: **for** each c_{ij} in C **do**
9: **if** $c_{ij} > T \cdot C_{\text{mid}}$ **then**
10: $M_{ij} \leftarrow 1$ `// activate the block`
11: **else**
12: $M_{ij} \leftarrow 0$
13: **end if**
14: **end for**
15: $M \leftarrow \textsc{Fill}(M)$ `// flood-fill`
16: $M_s \leftarrow \textsc{Regions}(M)$ `// split connected regions`
17: $R \leftarrow \emptyset$
18: **for** each m in M_s **do**
19: $I_m, p_m \leftarrow \textsc{Crop}(I, m)$ `// get sub-image`
20: add I_m and p_m to R
21: $R' \leftarrow \textsc{CGSeg}(I_m, g, D{+}1, D_{\max}, T)$
22: add all in R' to R
23: **end for**
24: **return** R

4.2 Hierarchical Retrieval-Augmented Code Generation Module

4.2.1 Generation Process

We design three collaborative MLLM agents whose code generation processes are augmented by H-RAG. **Describer** is responsible for generating a global *Design* description (Overall Description) as well as block descriptions (Block Description) for each segmented blocks. Based on block and their associated block description, **Coder** produces corresponding code snippets. Subsequently, **Assembler** generates the complete slide code by layout-aware prompt, which will be elaborated in §4.3, along with the *Pictures* provided. Executing this code produces a slide that structurally and visually aligns with the Reference Image(RI). If the generated code is not executable **Assembler** applies a self-refinement mechanism to correct syntax errors, where errors serves as the feedback to prompt the MLLM to re-generate the code.

Beyond the above inputs, each agent draws knowledge from distinct bases according to its role. The form and origin of the knowledge used in each agent's prompt are detailed in §4.2.2.

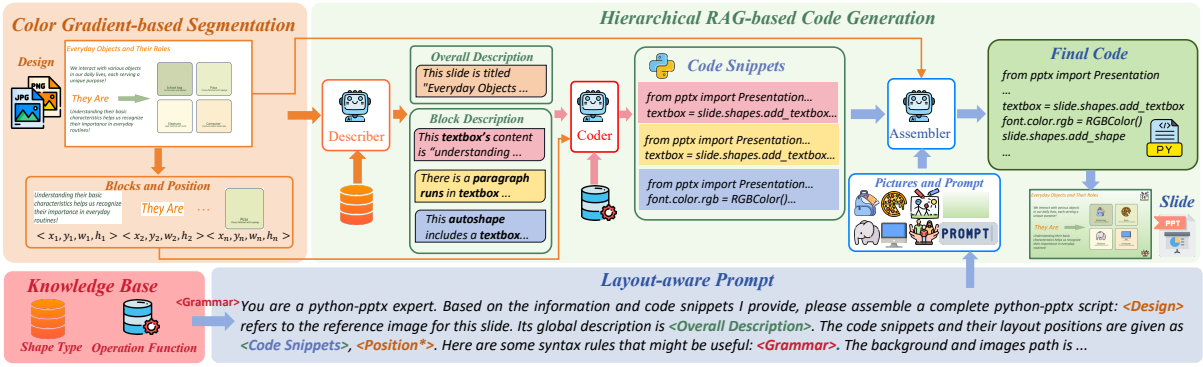

Figure 3: The framework of SlideCoder.

(a) Input Image (b) Activated Grid Blocks (c) Flood-filled Regions (d) Final result

Figure 4: An example of CGSeg applied to a slide reference image. The algorithm begins by computing color gradients (a-b), fills them (c), and recursively segments sub-regions (d).

4.2.2 Hierarchical Retrieval-Augmented Generation

Hierarchical Retrieval-Augmented Generation(H-RAG) comprises a Shape Type Knowledge Base and an Operation Function Knowledge Base. The former contains descriptions of objects from the python-pptx documentation, used in **Describer** to guide standardized description generation. For example, in "This ***autoshape*** includes a ***textbox***...", both terms are object names from the documentation. The latter includes full syntax specifications (e.g., parameters, return values, etc.). Appendix F details their structure.

We employ BGE M3-Embedding (Chen et al., 2024) to embed entries and build a vector-based retrieval database. For a prompt p, its vector q_p is computed, and cosine similarity $\cos(q_p, k_i)$ is used to match k_i. The top-k relevant entries are inserted into p. Given the size of the Shape Type Knowledge Base, all entries are included in **Describer** to ensure complete type coverage.

In the hierarchical pipeline, agents collaborate progressively. **Describer** retrieves object types from the Shape Type Knowledge Base to identify elements in block images and output standardized descriptions. **Coder** uses these to query the Operation Function Knowledge Base and generate code snippets. **Assembler** uses these snippets to retrieve full syntax patterns and generate executable code.

4.3 Layout-aware Prompt

After **Coder** completes the generation of code snippets for blocks, **Assembler** is applied to assemble these code snippets for generating the final slide in an accurate manner. The assembly prompt needs to meet the following requirements: (1) ensure that each block appears in the correct position in the final slide; (2) avoid syntax errors in the merged code and ensure code context consistency.

To achieve above goals, layout-aware prompt injects the layout position using python-pptx standard positioning units (inches) to ensure the position correctness and retrieve the grammar **<Grammar>** from Knowledge Base to avoid syntax errors and code conflicts. Since the resolution of the *Design* differs from the actual slide layout size, we apply proportional scaling to the Position (<x, y, w, h>) extracted from Color Gradients-based Segmentation (CGSeg) algorithm to map it onto the slide coordinates, denoted as **<Position*>**. Subsequently, the reference image design **<Design>**, global body description **<Overall description.>**, partial codes **<Code Snippets>** from **Coder**, layout representation **<Position*>**, and syntactic patterns **<Grammar>** retrieved from the Hierarchical Retrieval-Augmented Generation(H-RAG) knowledge base are integrated into a predefined prompt template to construct the final layout-aware prompt (see Appendix E for details).

4.4 SlideMaster

Using the SLIDESBENCH training set, we construct a dataset of (RI, instruction, program) triplets. The reverse-engineering tool proposed by (Ge et al., 2025) produces labels (Python code) for only a limited set of slide styles, resulting in suboptimal training data quality. To mitigate this, we develop a new reverse-engineering tool capable of handling a broader spectrum of slide styles, thereby enhancing label quality. The effectiveness of this tool is analyzed in §5.3. We fine-tune our model, SlideMaster, based on Qwen2.5-VL-7B-Instruct (Bai et al., 2025), using LoRA (Hu et al., 2022). Full configuration details are provided in Appendix C.

5 Experiments and Results

5.1 Experimental Setup

Model. To evaluate the performance of the SlideCoder, we employ state-of-the-art (SOTA) models, including GPT-4o (Achiam et al., 2023), Gemini-2.0-flash (Google, 2025), and SlideMaster, which is a fine-tuned model based on the open-source Qwen2.5-VL-7B-Instruct (Bai et al., 2025). The SOTA models are accessed via their official APIs, with GPT-4o using version 20241120 and Gemini-2.0-flash accessed in May 2025. For both models, the maximum token limit and temperature are set to 4096 and 0, respectively. Same as (Ge et al., 2025), we allow both **Coder** and **Assembler** agents up to three self-refinement attempt. The first successful attempt is taken as the output. If **Coder** fails to generate executable code after the maximum number of attempts, the corresponding block is discarded. If **Assembler** fails, the corresponding sample is marked as execution failure.

Metric. To comprehensively assess generation quality, we adopt four metrics, using the notations defined in §3.1. (1) **Global Visual Metrics**, including CLIP (Hessel et al., 2021) and SSIM (Nilsson and Akenine-Möller, 2020) scores computed between the original image I_0 and the generated image I_g; (2) **Local Structural Metrics**, which compare the original and generated slide files F_0 and F_g in terms of content similarity and position similarity, following (Ge et al., 2025); (3) **Execution**, defined as the success rate of executing C_g without errors; and (4) **Overall Score**, calculated as the average of all metric values across all samples, with failed executions assigned a score of zero.

5.2 Quantitative Results and Analysis

The upper part of Table 1 presents the performance of different frameworks on our proposed benchmark, evaluated using the metrics introduced in Section 3.1. The results show that SlideCoder consistently achieves the best performance across all difficulty levels. Specifically, its overall score surpasses the best baseline by 40.5, 34.0, and 29.9 points on the simple, medium, and complex levels, respectively, demonstrating the overall superiority of our framework. For execution success rate, SlideCoder outperforms the best baseline by 38%, 32%, and 27% across the three difficulty levels, indicating that the proposed H-RAG and CGSeg mechanisms significantly enhance model performance and reduce task difficulty.

Moreover, SlideCoder outperforms all baselines in both Local Structural Metrics and Global Visual Metrics, confirming its strong fidelity in preserving both the structural layout and visual appearance of the original slides. The stepwise decline in SlideCoder's overall score across increasing difficulty levels further indicates its ability to leverage visual and structural cues from the input slides. In contrast, baseline models relying solely on natural language descriptions exhibit weak sensitivity to slide complexity, failing to reflect the difficulty hierarchy in their overall scores.

On the SLIDESBENCH dataset (as shown in the lower part of Table 1), SlideCoder also surpasses all baselines across all metrics, with an overall score of 78.8 when using GPT-4o as the backbone, representing a 11.9 improvement over the best-performing baseline. Notably, the open-source fine-tuned model SlideMaster also demonstrates competitive performance, outperforming the best GPT-4o-based baseline on both datasets.

5.3 Reverse Tool Analysis

Table 2 summarizes the supported object types and corresponding styles in our proposed reverse engineering tool. Our tool supports 10 commonly used object types and 44 distinct object styles, whereas Autopresent (Ge et al., 2025) only supports 5 object types and 16 styles. Detailed comparisons can be found in Appendix B. To quantitatively assess the reverse engineering capabilities of both tools, we adopt two evaluation metrics:

Reconstruction Ratio: This metric calculates the ratio between the number of shapes in the slide reconstructed from the reverse-engineered code

Table 1: Results on Slide2Code (top) and SLIDESBENCH (bottom) using SlideCoder and AutoPresent with different MLLMs. Green, yellow, and red indicate simple, medium, and complex levels in SlideCoder. **Bolded values** mark the best result per level.

Framework	Backbone	Execution%	Local Structural Metrics		Global Visual Metrics		Overall
			Content	Position	Clip	SSIM	
Slide2Code							
AutoPresent	AutoPresent	61.0	92.7	78.9	70.8	80.3	48.6
		53.0	89.6	77.3	69.2	79.1	41.4
		67.0	87.2	71.4	65.9	73.4	48.5
	Gemini2.0-flash	57.0	91.4	78.3	69.7	79.0	44.8
		68.0	88.7	79.9	66.3	71.6	51.5
		66.0	89.3	72.2	63.1	64.7	45.2
	GPT-4o	58.0	92.7	80.9	68.8	75.6	45.4
		50.0	92.3	74.6	67.6	72.6	36.8
		69.0	90.3	73.3	62.3	63.3	47.1
SlideCoder	SlideMaster	86.0	92.4	87.4	77.6	91.1	76.7
		75.0	84.7	79.8	75.4	**86.4**	61.7
		73.0	76.1	70.5	72.4	**82.8**	54.2
	Gemini2.0-flash	97.0	94.5	**88.6**	**81.3**	90.7	87.0
		90.0	90.9	84.6	**82.3**	85.5	76.6
		88.0	92.7	**80.9**	**81.7**	81.2	71.6
	GPT-4o	**99.0**	**96.3**	88.1	79.8	**91.8**	**89.1**
		100.0	**92.5**	**84.7**	81.5	86.2	**85.5**
		96.0	**94.3**	80.0	80.7	82.6	**78.4**
SLIDESBENCH							
AutoPresent	AutoPresent	84.1	92.2	67.2	81.6	73.7	65.3
	Gemini2.0-flash	56.4	91.7	62.9	77.1	66.0	40.4
	GPT-4o	86.7	92.5	76.3	78.0	70.8	66.9
SlideCoder	SlideMaster	87.2	91.5	76.9	73.4	80.0	68.4
	Gemini2.0-flash	89.7	90.0	**85.4**	81.8	80.0	75.0
	GPT-4o	**94.9**	**94.8**	83.9	**82.1**	**80.9**	**78.8**

Table 2: Object Types and Corresponding Style count

Type Name	Ours	AutoPresent's
`title`	10	3
`textbox`	10	5
`bullet points`	8	5
`background color`	1	1
`image`	2	2
`placeholder`	4	–
`freeform`	2	–
`connector`	5	–
`table`	4	–
`triangle`	5	–

and the original slide. Our tool achieves a reconstruction ratio of 90.38%, significantly outperforming (Ge et al., 2025), which only reaches 65.67%. This demonstrates the broader object type coverage enabled by our tool.

CLIP Score: Our method achieves a CLIP score (Hessel et al., 2021) of 88.66%, whereas Autopresent (Ge et al., 2025) only achieves 69.87%. The higher score indicates that our reverse-engineered slides more accurately preserve the visual and stylistic details of the original, owing to the broader support for object types and styles.

5.4 Slide Complexity Metric Analysis

To evaluate the effectiveness of the proposed Slide Complexity Metric (SCM), we conducted a human subject study. A total of 100 samples were randomly selected from the Slide2Code benchmark for evaluation. Four doctoral students were recruited as annotators, each assigned 50 slides to assess. The annotators were instructed to score each slide from the perspective of three dimensions: the number of shapes, the diversity of shape types, and the level of element coverage. The scoring range was 0–100, following the protocol in Appendix D. Each slide was rated independently by two annotators, and the final score was their average.

To assess the alignment between SCM and human perception, we first compute the Pearson correlation coefficient (Cohen et al., 2009) between the TSC complexity scores and the averaged human scores. The result is $r = 0.873$ with a p-value of 2.776×10^{-32}, indicating a strong and statistically significant correlation. Additionally, we calculated the intraclass correlation coefficient (Koo and Li, 2016) between the SCM scores and each individual annotator's score to assess consistency. The ICC result is 0.726 with a p-value of 1.186×10^{-31},

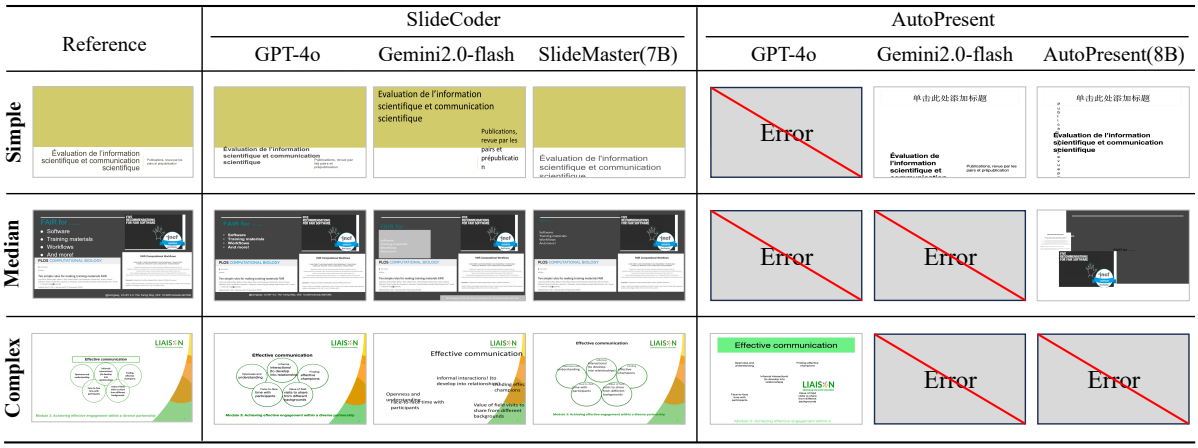

Figure 5: Examples of slides generated by different methods in three difficulty levels.

demonstrating substantial agreement between SCM and human evaluations. These results confirm that SCM is a reliable and objective metric aligned with human judgment of slide complexity.

5.5 Ablation Study

Table 3: Overall performance of ablation study.

Setting	Execution%	Overall
SlideCoder	100.0	89.9
	100.0	85.8
	100.0	82.2
w/o Layout	100.0	81.2
	93.9	73.6
	93.9	71.8
w/o CGSeg	75.8	55.4
	51.5	39.6
	69.7	48.4
w/o H-RAG	90.9	80.4
	81.8	69.3
	84.8	70.7
Native Setting	75.8	53.9
	48.5	37.4
	66.7	46.9

We design three ablation settings to validate the effectiveness of different components in our framework: (1) w/o Layout, removes the layout-aware prompt; (2) w/o CGSeg, disables both the CGSeg mechanism and the layout-aware prompt; (3) w/o H-RAG, removes the **<Grammar>** content from all prompts.(4) Native setting, which removes H-RAG on top of the w/o CGSeg setting. Detailed descriptions are provided in Appendix A.1. We randomly sample 33 instances from each difficulty level, resulting in a total of 99 samples, and perform inference using GPT-4o. The overall results are reported in Table 3, with detailed metric result provided in Appendix A.2. After removing each component, both execution rate and overall score exhibit varying degrees of decline, which demonstrates the contribution of each component to the overall framework. Notably, the w/o CGSeg setting shows significant performance drops across all metrics. Although slightly better than the Native setting due to the presence of H-RAG.

5.6 Case Study

Figure 5 presents slides generated by different models under three levels of difficulty. It can be observed that models based on natural language often fail to satisfy the detailed and layout-specific requirements of reference images. These models frequently produce slides with overlapping elements or content that extends beyond canvas boundaries. In medium and complex samples, the generated code often fails to compile. In contrast, SlideCoder's CGSeg mechanism enables the MLLM to focus more effectively on fine-grained details. Moreover, the layout-aware prompt helps ensure that the spatial arrangement of elements aligns more closely with reference image.

6 Conclusion

We introduce a new Reference Image to Slide Generation task and a novel Slide Complexity Metric for evaluating slide complexity. Based on this metric, we build the Slide2Code benchmark with different levels of difficulty. We also propose SlideCoder enhanced by a Color Gradients-based Segmentation algorithm, a Layout-aware Prompt and a Hierarchical Retrieval-Augmented Code Generation for accurate slide generation. A high-quality training set is curated to fine-tune a 7B open-source model. Experimental results show that SlideCoder outperforms the strongest baselines.

Limitations

In this work, we take the first step toward vision-based slide generation. While our method achieves substantial improvements across multiple evaluation metrics, several limitations remain unaddressed. First, the current framework focuses on generating a single slide from one reference image and does not explore the multi-slide generation scenario. Second, we assume that user input contains separate design and image components, and do not handle the case where a complete slide with embedded pictures is provided as input. Third, due to budget and time constraints, our segmentation algorithm adopts a fixed-rule paradigm. Future work may investigate more flexible model-based detection approaches to enable adaptive and accurate block partitioning.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

SV Burtsev and Ye P Kuzmin. 1993. An efficient flood-filling algorithm. *Computers & graphics*, 17(5):549–561.

Steve Canny. 2023. Python-ptx documentation.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2318–2335.

Israel Cohen, Yiteng Huang, Jingdong Chen, Jacob Benesty, Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. *Noise reduction in speech processing*, pages 1–4.

Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–729.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501.

Tsu-Jui Fu, William Yang Wang, Daniel McDuff, and Yale Song. 2022. Doc2ppt: Automatic presentation slides generation from scientific documents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 634–642.

Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, and 1 others. 2025. Autopresent: Designing structured visuals from scratch. *arXiv preprint arXiv:2501.00912*.

Yingqiang Ge, Wenyue Hua, Kai Mei, Juntao Tan, Shuyuan Xu, Zelong Li, Yongfeng Zhang, and 1 others. 2023. Openagi: When llm meets domain experts. *Advances in Neural Information Processing Systems*, 36:5539–5568.

Google. 2025. Gemini API. https://ai.google.dev/gemini-api. Accessed: 2025-05-19.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks*, pages 195–201. Springer.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. 2021. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

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

Min-Yen Kan. 2007. 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.

Terry K Koo and Mae Y Li. 2016. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of chiropractic medicine*, 15(2):155–163.

Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, Zhiyong Huang, and Jing Ma. 2024. Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems. *arXiv preprint arXiv:2404.09486*.

Jenny GuangZhen Ma, Karthik Sreedhar, Vivian Liu, Pedro A Perez, Sitong Wang, Riya Sahni, and Lydia B Chilton. 2025. Dynex: Dynamic code synthesis with structured design exploration for accelerated exploratory programming. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–27.

Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Jim Nilsson and Tomas Akenine-Möller. 2020. Understanding ssim. *arXiv preprint arXiv:2006.13846*.

Kunato Nishina and Yusuke Matsui. 2024. Svgeditbench: A benchmark dataset for quantitative assessment of llm's svg editing capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8142–8147.

Juan A Rodriguez, Abhay Puri, Shubham Agarwal, Issam H Laradji, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2025. Starvector: Generating scalable vector graphics code from images and text. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 29691–29693.

Athar Sefid and Jian Wu. 2019. 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*.

Yuxuan Wan, Yi Dong, Jingyu Xiao, Yintong Huo, Wenxuan Wang, and Michael R Lyu. 2024a. Mrweb: An exploration of generating multi-page resource-aware web code from ui designs. *arXiv preprint arXiv:2412.15310*.

Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R Lyu. 2024b. Automatically generating ui code from screenshot: A divide-and-conquer-based approach. *arXiv preprint arXiv:2406.16386*.

Ronghuan Wu, Wanchao Su, and Jing Liao. 2024. Chat2svg: Vector graphics generation with large language models and image diffusion models. *arXiv preprint arXiv:2411.16602*.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zhiyao Xu, and Michael R Lyu. 2024. Interaction2code: How far are we from automatic interactive webpage generation? *arXiv preprint arXiv:2411.03292*.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. 2024. Empowering llms to understand and generate complex vector graphics. *arXiv preprint arXiv:2412.11102*.

John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, and 1 others. 2024. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*.

Mingyue Yuan, Jieshan Chen, Zhenchang Xing, Aaron Quigley, Yuyu Luo, Tianqi Luo, Gelareh Mohammadi, Qinghua Lu, and Liming Zhu. 2024. Design-repair: Dual-stream design guideline-aware frontend repair with large language models. *arXiv preprint arXiv:2411.01606*.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, and 1 others. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *arXiv preprint arXiv:2406.20098*.

Fengji Zhang, Linquan Wu, Huiyu Bai, Guancheng Lin, Xiao Li, Xiao Yu, Yue Wang, Bei Chen, and Jacky Keung. 2024a. Humaneval-v: Evaluating visual understanding and reasoning abilities of large multimodal models through coding tasks. *arXiv preprint arXiv:2410.12381*.

Linhao Zhang, Daoguang Zan, Quanshun Yang, Zhirong Huang, Dong Chen, Bo Shen, Tianyu Liu, Yongshun Gong, Pengjie Huang, Xudong Lu, and 1 others. 2024b. Codev: Issue resolving with visual data. *arXiv preprint arXiv:2412.17315*.

Hao Zheng, Xinyan Guan, Hao Kong, Jia Zheng, Weixiang Zhou, Hongyu Lin, Yaojie Lu, Ben He, Xianpei Han, and Le Sun. 2025. Pptagent: Generating and evaluating presentations beyond text-to-slides. *arXiv preprint arXiv:2501.03936*.

A Detail ablation analysis

A.1 Details of Ablation Settings

- w/o Layout: Removes only the layout-aware prompt, meaning that the input to **Assembler** does not contain the positional coordinates of each block.

- w/o CGSeg: Disables the CGSeg mechanism. Since the goal of **Coder** is to generate partial code and **Assembler** is responsible for code assembly, the removal of CGSeg renders

Assembler unnecessary. Consequently, both **Assembler** and its layout-aware prompt are removed in this setting, and the output code generated by **Coder** is directly treated as the final output of the framework.

- w/o H-RAG: Disables the retrieval of knowledge base content for all agents.

- Native setting: Disables both H-RAG and CSeg components. Specifically, we input ordinary prompts that do not incorporate H-RAG, allowing the MLLMs to generate complete slide code directly from the reference image. This setup is used to evaluate the baseline capability of native MLLMs in handling the reference image to slide code generation task.

A.2 Detailed Analysis of Ablation Results

Table 4 provides a detailed evaluation metrics under different ablation settings.

In the w/o Layout setting, the Position score under the complex level drops significantly from 81.35 to 72.16. This is primarily because, in complex cases, the CGSeg algorithm typically divides the Reference Image(RI) into more blocks, and without layout information, the Agent struggles to model spatial relationships among multiple elements. This often leads to overlapping or out-of-bound content, causing a sharp decline in the Position metric and slightly affecting other metrics as well.

In the w/o CGSeg setting, both the CGSeg mechanism and the layout-aware prompt are removed. As a result, a single **Describer** Agent is required to handle the entire complex slide, which exceeds its processing capacity, often leading to code generation failures and a sharp drop in execution success rate. Its performance is slightly better than the Native setting due to the additional knowledge provided by H-RAG.

In the w/o H-RAG setting, the **<Grammar>** component is removed from each Agent. Excluding this component from **Describer** reduces its ability to accurately identify the corresponding python-pptx object. Similarly, removing it from **Coder** and **Assembler** deprives the Agents of essential syntactic guidance, often resulting in version-related errors caused by inconsistencies between the model's training data and the current version of the python-pptx library. These combined factors lead to overall performance degradation.

In the Native setting, both the CGSeg mechanism and H-RAG are removed, leaving a single **Coder** Agent to handle the entire slide without any auxiliary support. This reduces the framework to a plain MLLM-based inference process, severely limiting its ability to generate structured and executable code, and resulting in the lowest execution rate and overall performance.

B Detailed comparisons of Reverse Tool

Table 5 lists the object types and their styles supported by our reverse engineering tool.

Table 6 lists the object types and their styles supported by AutoPresent's reverse engineering tool.

C LoRA fine-tuning parameters

The LoRA fine-tuning parameters are listed in Table 7.

D Evaluation Dimensions and Scoring Criteria

The evaluation guidelines for the four doctoral student annotators are provided in Figure 6.

E Prompt Templates

The prompt templates for the Describer and Coder are shown in Figure 7 and Figure 8, respectively. Layout-aware prompt is shown in Figure 9.

F Details of the Knowledge Base Construction

Figure 10 presents several examples from the Shape Type Knowledge Base, which consists of object types defined in the python-pptx library along with their corresponding descriptions. Figure 11 shows an example from the Operation Function Knowledge Base, which includes the function name, parameters, return value, usage example, and a textual explanation of the function.

Table 4: Detailed performance analysis under several ablation settings. Green , yellow , and red indicate simple, medium, and complex levels in SlideCoder. **Bolded values** mark the best result per level.

Setting	Execution%	Global Visual Metrics		Local Structural Metrics		Overall
		Content	Position	Clip	SSIM	
SlideCoder	**100.0**	**97.1**	**89.9**	80.8	**92.9**	**89.9**
	100.0	92.7	**86.5**	**82.7**	85.8	**85.8**
	100.0	**95.0**	81.3	**82.2**	82.3	**82.2**
w/o Layout	100.0	88.8	86.4	**81.2**	79.2	81.2
	93.9	90.4	75.2	80.9	78.4	73.6
	93.9	93.6	72.2	80.3	76.4	71.8
w/o CGSeg	75.8	90.4	86.5	69.4	73.1	55.4
	51.5	91.7	81.4	68.5	71.4	39.6
	69.7	93.0	83.2	68.1	69.0	48.4
w/o H-RAG	90.9	98.6	88.4	79.7	91.8	80.4
	81.8	91.6	84.7	81.7	**87.8**	69.3
	84.8	94.0	**87.9**	81.3	**83.4**	70.7
Native Setting	75.8	90.0	87.9	71.1	71.2	53.9
	48.5	**92.9**	83.3	66.7	69.5	37.4
	66.7	92.6	85.7	66.5	70.4	46.9

Table 5: The object types and their styles supported by our reverse engineering tool.

Object Type	Styles
textbox	Position, Text frame margin, Alignment, Paragraph spacing, Font style, Fill color, Font size, Bold, Italic, Underline
rectangle	Position, Line color, Line width, Fill color
object_placeholder	Position, Fill color, Object position
freeform	Position, Fill color
bullet_points	Position, Item content, Font size, Font color, Fill color, Bold, Italic, Underline
image	Position, Image path
background_color	Color
connector	Start position, End position, Arrow color, Arrow width, Arrow style
table	Position, Cell height, Cell fill color, Text inside cell
triangle	Position, Type, Line color, Line width, Fill color

Table 6: The object types and their styles supported by AutoPresent's reverse engineering tool.

Object Type	Styles
title	Font size, Font color, Fill color
textbox	Position, Font size, Bold, Font color, Fill color
bullet_points	Position, Item content, Font size, Font color, Fill color
image	Position, Image path
background color	Color

Slide Complexity Evaluation Guide

Purpose of Evaluation

This guideline is intended to assist you in subjectively evaluating the complexity of slide samples based on the following three dimensions:

1. **Number of Shapes**
2. **Diversity of Shape Types**
3. **Visual Complexity**

Each dimension should be scored on a scale from **0 to 100**. You are expected to assess each slide independently and provide a **final overall score** reflecting your holistic judgment of the slide's complexity.

Evaluation Procedure

For each slide, please follow these steps:

1. Review the slide thoroughly to understand its structure and element layout.
2. Evaluate each of the three dimensions separately (see detailed criteria below).
3. Based on your judgment, assign a comprehensive **overall score** (0–100).Record your scores (three dimensions + overall) clearly in the scoring table.

Scoring Dimensions and Criteria

1. Number of Shapes

Refers to the total count of visual elements on the slide, including but not limited to: text boxes, diagrams, arrows, lines, images, geometric shapes, etc.

- **0–20**: Very few elements (e.g., only a title and 1–3 text boxes).
- **21–50**: Moderate amount of shapes (e.g., 4–10 elements, such as text + one chart).
- **51–80**: High density of shapes (e.g., 11–20 elements, visually filled slide).
- **81–100**: Extremely dense, cluttered with over 20 elements.

2. Diversity of Shape Types

Measures how varied the types of visual components are. Common types include text boxes, images, tables, flowcharts, icons, arrows, geometric shapes (e.g., rectangles, circles, lines), and more.

- **0–20**: Only one type used (e.g., all text).
- **21–50**: Two or three different types, basic variety.
- **51–80**: Four to six types, indicating notable diversity.
- **81–100**: Rich variety with more than six distinct shape types.

3. Visual Complexity

Refers to how complex the slide appears in terms of visual density, layout structure, information layering, and cognitive load. It captures the subjective perception of how "complicated" the slide looks.

- **0–20**: Very clean and minimalist, with generous whitespace.
- **21–50**: Well-structured, moderately filled, visually comfortable.
- **51–80**: Noticeably dense, some clutter, yet still readable.
- **81–100**: Overwhelming amount of information, chaotic layout, hard to scan quickly.

Overall Score Guidelines

After rating the three dimensions above, you are asked to provide a **final overall score** (0–100) that reflects your subjective judgment of the slide's overall complexity.

⚠️ Note: This **does not need to be a simple average** of the three scores. Instead, consider how each factor influences the overall perception of complexity.

Figure 6: Evaluation guidelines provided to the four doctoral student annotators.

Prompt of Describer

Block Description

You are a python-pptx expert. Please describe this region in detail, including its textual content, graphical elements, and layout structure. Analyze both the content and its visual presentation.
The following is an introduction to layout shape types in PPT:
{<Grammar>}
{block_image}

Overall PPT Description

Please provide a detailed description of this PPT screenshot, including its overall content and layout. Describe how different sections are arranged and what main content each section contains.
The following is an introduction to layout shape types in PPT:
{<Grammar>}
{reference_image}

Figure 7: Prompt of Describer.

Prompt of Coder

Code generation process

Please write Python code to create a PowerPoint slide that matches the following description:
{block_description}

The following is an introduction in python-pptx API Documentation:
{<Grammar> }

Please generate Python code using the python-pptx library to create a PowerPoint slide based on the provided codes. The code should:
1. Create a new PowerPoint presentation.
2. Add a slide using the slide layout with index 6 (typically a Blank Layout) to ensure a clean slate for custom content placement.
3. Include all text elements and shapes as specified in the slide, with properties such as font, size, color, and alignment accurately applied.
4. Use inches (in) units for all size and position measurements, directly converting them using python-pptx's Inches() function for shapes and positions, and Pt for font sizes.
5. Save the presentation in the output/generated_ppts directory with a descriptive filename (e.g., generated_slide.pptx).
6. Ensure the code is well-commented and handles any necessary imports.
{block_image}

Fix code process

You are a python-pptx expert.
The previous code generated an error. Please fix the code.
Error message:
{error_message}
Previous code:
{code}

Introduction in python-pptx API Documentation:
{<Grammar> }
Please provide the complete corrected code that will create the PowerPoint slide successfully.

Figure 8: Prompt of Coder.

Layout-aware prompt

You are a python-pptx expert. Based on the information and code snippets I provide, please assemble a complete python-pptx script:
<Design> refers to the reference image for this slide.

Its global description is *<Overall Description>*.

The code snippets and their layout positions are given as
<Code Snippets1>, <Position1>.*
<Code Snippets1>, <Position1>.*
…

Here are some syntax rules that might be useful: *<Grammar>*.

The background and images path is ...

Background path:
{background_image_path}
Image1 Path:
{image_path_1}
Image1 Coordinates:
Left: *{x1}* inches
Top: *{y1}* inches
Width: *{w1}* inches
Height: *{h1}* inches
Please provide the complete corrected code that will create the PowerPoint slide successfully.
Please generate Python code using the python-pptx library to create a PowerPoint slide based on the provided codes. The code should:

1. Create a new PowerPoint presentation.
2. Add a slide using the slide layout with index 6 (typically a Blank Layout) to ensure a clean slate for custom content placement.
3. Include all text elements and shapes as specified in the slide, with properties such as font, size, color, and alignment accurately applied.
4. Use inches (in) units for all size and position measurements, directly converting them using python-pptx's Inches() function for shapes and positions, and Pt for font sizes.
5. Save the presentation in the output/generated_ppts directory with a descriptive filename (e.g., generated_slide.pptx).
6. Ensure the code is well-commented and handles any necessary imports.

Figure 9: Layout-aware prompt.

Auto Shape
An auto shape is a predefined, customizable shape in PowerPoint, such as a rectangle, ellipse, or block arrow, with approximately 180 variations. Auto shapes can have a fill, outline, and contain text. Some include adjustable features, indicated by yellow diamond handles (e.g., to modify the corner radius of a rounded rectangle). A text box is a specific type of auto shape, typically rectangular, with no default fill or outline.

########

Picture
A picture in PowerPoint refers to a raster image, such as a photograph or clip art, treated as a distinct shape type with unique behaviors compared to auto shapes. Note that an auto shape can have a picture fill, where an image serves as the shape's background instead of a color or gradient, but this is a separate feature.

########

Graphic Frame
A graphic frame is a container that automatically appears in a PowerPoint file when adding graphical objects like tables, charts, SmartArt diagrams, or media clips. It cannot be inserted independently and typically requires no direct interaction from the user.

########

Group Shape
A group shape is created when multiple shapes in PowerPoint are grouped, enabling them to be selected, moved, resized, or filled as a single unit. The group shape is only visible through its bounding box when selected, containing the individual member shapes.

########

Line/Connector
Lines are linear shapes distinct from auto shapes. Some lines, known as connectors, can attach to other shapes and remain connected when those shapes are moved. Connectors are not yet fully supported in some contexts, but they are valuable for creating dynamic diagrams.

########

Content Part
A content part involves embedding external XML data, such as SVG, within a PowerPoint presentation. PowerPoint itself does not actively utilize content parts, and they can generally be ignored without impacting functionality.

……

Figure 10: Examples from the Shape Type knowledge base.

Function: `pptx.Presentation`

Function Name

`pptx.Presentation`

Function Parameters

- **pptx** (`Union[str, IO[bytes], None]`, optional, default: `None`)
  - Description: Specifies the source of the presentation.
    - If a `str`, it represents the file path to a `.pptx` file.
    - If an `IO[bytes]`, it represents a file-like object containing the `.pptx` file data.
    - If `None`, loads the built-in default presentation template.
  - Constraints: The file or stream must be a valid `.pptx` file if provided.

Function Return Value

- **Type**: `presentation.Presentation`
- **Description**: A `Presentation` object representing the loaded or newly created PowerPoint presentation.

Function Python Example

```python
from pptx import Presentation

# Create a new presentation using the default template
prs = Presentation()

# Load an existing presentation from a file
prs = Presentation("existing_presentation.pptx")

# Load a presentation from a file-like object
from io import BytesIO
with open("existing_presentation.pptx", "rb") as f:
    prs = Presentation(BytesIO(f.read()))
```

Function Purpose

The `pptx.Presentation` function is the primary entry point for creating or loading a PowerPoint presentation. It initializes a `Presentation` object, which provides access to slides, slide masters, layouts, and other presentation components, enabling programmatic manipulation of presentation content.

Figure 11: An example from the Operation Function knowledge base.

Table 7: LoRA fine-tuning configuration used in our experiments.

Parameter	**Value**
Rank	8
Max Sequence Length	4096
Batch Size	4
Gradient Accumulation Steps	8
Learning rate	1e-4
Epochs	10
Warmup Ratio	0.1
Mixed Precision	bf16