Talk to Your Slides: Language-Driven Agents for Efficient Slide Editing

Anonymous ACL submission

Abstract

Editing presentation slides remains one of the most common and time-consuming tasks faced by millions of users daily, despite significant advances in automated slide generation. Existing approaches have successfully demonstrated slide editing via graphic user interface (GUI)-based agents, offering intuitive visual control. However, such methods often suffer from high computational cost and latency. In this paper, we propose TALK-TO-YOUR-SLIDES, an LLM-powered agent designed to edit slides by leveraging structured information about slide objects rather than relying on image modality. The key insight of our work is designing the editing process with distinct high-level and low-level layers to facilitate interaction between user commands and slide objects. By providing direct access to application objects rather than screen pixels, our system enables 34.02% faster processing, 34.76% better instruction fidelity, and 87.42% cheaper operation than baselines. To evaluate slide editing capabilities, we introduce *TS-Bench*, a human-annotated dataset comprising 379 diverse editing instructions paired with corresponding slide variations in four categories. Our code, benchmark and demos are available at anonymous.4open.science/r/talk-to-your-slides.

1 Introduction

Recent advancements in large language models (LLMs) have revolutionized how we interact with software applications through natural language instructions, demonstrating remarkable success in tasks such as code generation, GUI navigation, and slide generation (Yang et al., 2024; Hou et al., 2024; Zhang et al., 2024a; Xu et al., 2025). While these models have enabled significant progress in automated slide creation, a critical yet underexplored challenge remains in editing existing presentation slides. Presentation slides serve as a fundamental medium for communication across education, business, and research. However, modifying them to reflect updated content, adjust the layout, or enhance clarity often demands tedious, time-consuming manual effort. For example, as shown in Figure 1, a professor prepares an international lecture that contains 50 slides across 10 different sessions and has to be translated from Korean to English while preserving technical terminology and formatting. Similarly, a marketing team needs to update product pricing on 120 slides spanning several presentations before a major launch.

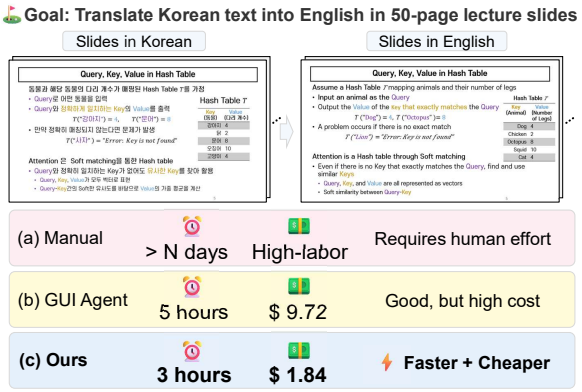

Figure 1: Comparison of slide editing methods on translating 50-page lecture slides from Korean to English. (a) Manual translation requires day(s) and consumes graduate-student labor. (b) A GUI-based agent reduces human effort but incurs high cost and occupies the host machine during execution. However, (c) our approach runs in the background at a low cost and in a relatively short time.

Several candidate approaches can be a solution to address these challenges. One straightforward approach is converting natural language instructions into direct scripting code that can be applied to PowerPoint presentations. But this baseline struggles with complex tasks requiring sequential operations. For instance, instructions like

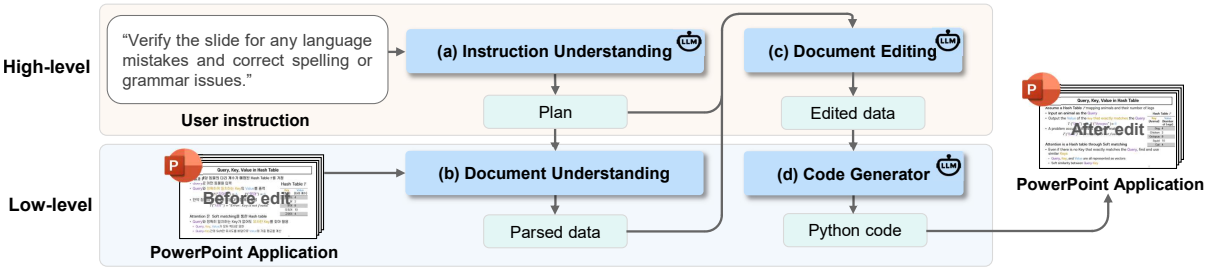

Figure 2: Overview of the TALK-TO-YOUR-SLIDES framework. The system consists of four modules: instruction understanding, document understanding, document editing, and code generator.

"Summarize the text on all slides and highlight key points in red" demand understanding both of user intent and contextual interpretation of slide content. Another approach leverages vision-based GUI agents that operate on screen-captured images (Figure 1b) through repeated mouse click and keyboard interactions. However, due to the high computational cost of processing image inputs with vision-language models (VLMs), and the latency between interactions, this method tends to be resource-intensive and expensive to deploy. This limitation extends beyond presentation software such as PowerPoint to any application where vision-only interfaces become bottlenecks, such as Windows or iOS navigation agents.

To alleviate these limitations, we propose TALK-TO-YOUR-SLIDES, an LLM-powered agent designed to edit presentation slides by directly handling objects through text-based low-level structured information. This approach enables more accurate and cost-effective editing operations compared to multi-modal agents. We design the editing process with distinct high-level and low-level layers to facilitate interaction between high-level user commands and low-level objects within slides. This insight draws from previous research on planning and reasoning processes (Guo et al., 2024b; Chen et al., 2024a,b; Zhao et al., 2023; Caldiran et al., 2009). At the high-level, an LLM agent interprets user instructions and formulates structured editing plans. At the low-level, our system directly accesses slide components and executes precise edits through generated code. By providing direct access to application objects, we enable 34.02% faster processing, 34.76% better instruction fidelity, and 87.42% cheaper operation than GUI-based methods.

Furthermore, to complement existing benchmarks that primarily focus on the visual aesthetics of slide generation (Ge et al., 2025; Zheng et al., 2025; Guo et al., 2023; Zhang et al., 2024b), we present *TSBench*, a human-annotated dataset specifically designed to evaluate slide editing capabilities. TSBench consists of 379 diverse editing instructions, each paired with corresponding slides created using official Microsoft slide templates. Our benchmark enables systematic assessment of models' proficiency in applying fine-grained modifications to existing presentation content. We categorize the editing commands into four distinct types: text editing, visual formatting, layout adjustment, and slide structure manipulation. These categories cover practical tasks such as modifying existing text, adjusting visual elements, aligning components, and managing slide transitions, respectively.

Our contributions are summarized as follows:

- We introduce a system TALK-TO-YOUR-SLIDES, an LLM agent-based system specifically designed for slide editing tasks that approaches editing through a division of high-level and low-level operations.
- We construct *TSBench*, a human-annotated benchmark dataset that enables systematic evaluation of slide editing agents in terms of their ability to accurately follow complex user instructions.
- Through comprehensive experiments, we demonstrate that TALK-TO-YOUR-SLIDES substantially outperforms baseline methods across execution success rate, instruction fidelity, and editing efficiency, reducing execution time by up to 87%.

2 Related Work

This section covers related research on slide generation, GUI-based agents, and code generation with LLMs.

2.1 Slide Generation

Prior work has primarily focused on generating presentation slides from natural language descriptions (Sefid et al., 2021). AutoPresent (Ge et al., 2025) fine-tuned an LLaMA-based model on the SlidesBench training set, a dataset comprising 7,000 slide-generation examples, to generate Python code that invokes the SlidesLib API. However, this approach remains prone to execution errors and does not support fine-grained slide editing. PPTAgent (Zheng et al., 2025) presents a simple process that mimics how people author slides. It first creates an outline and then edits slides using a fixed template. It also includes PPTEval, a tool to check slide content, design, and structure. PPTAgent works well for generating new slides. Our research extends these approaches by introducing precise editing capabilities that significantly reduce the manual effort required from users.

2.2 LLM Agents for GUI Control

Our work is also related to research on LLM-based agents that control graphical user interfaces (GUIs) (Gao et al., 2024; Koh et al., 2024). UFO and UFO2 by Microsoft (Zhang et al., 2024a) introduces a dual-agent framework composed of an application-selection agent and an action agent that can operate across Windows applications such as Word and PowerPoint. By observing application screenshots, the agent executes actions like menu clicks and text input. While powerful, UFO relies on image-based state representations and pixel-level interactions, which can introduce high computational costs and imprecise behavior, particularly for complex editing tasks. We compare our system against this model as a baseline.

2.3 Code Generation from language instructions

The task of translating natural language instructions into executable code has attracted considerable attention with the emergence of LLMs. While early work (Zan et al., 2023; Jiang et al., 2024; Yin et al., 2023) relied on rule-based systems or domain-specific languages-approaches that often lacked scalability and adaptability, LLMs have enabled more flexible, generalizable solutions. Building on this progress, recent studies (Wang et al., 2025a; Sun et al., 2024; Puerto et al., 2024; Yang et al., 2025) have introduced intermediate reasoning steps to further enhance code generation, such as guiding code generation through explicit natural language planning, using intermediate plans to decompose and solve complex, multi-step coding tasks-thereby bridging the gap between high-level user intent and low-level executable code. We utilize this code generation idea in our system to translate user instructions into slide editing operations.

```
{ "understanding": "Emphasize the important p
arts on slide 2 and slide 3.",
  "tasks": [
    { "page number": 2,
      "description": "Highlight the key phrases in t
he title and body text of slide 2 using bold and re
d font.",
      "target": "Title and Body text on slide 2" },
    { "page number": 3,
      "description": "Emphasize the most importa
nt content on slide 3 by underlining and changing
text color to blue.",
      "target": "All text elements on slide 3 (Title,
Body, Footers, Headers, Captions, Chart/Table la
bels, etc.)"    } ] }
```

Figure 3: Example output generated by the instruction understanding module.

3 Method

We categorize the capabilities required to edit slides given a user's instruction into four key components. First, the system must accurately understand the user's instruction. Second, to implement this instruction, it needs to comprehend the current state of the presentation slides. Third, based on the instruction and current state, it should generate slide data that reflects the instruction. Finally, it must implement these generated changes in the presentation environment such as PowerPoint. These sequential requirements naturally divide slide editing tasks into two levels. High level operations involve instruction interpretation and content editing. Low level operations require direct access to and manipulation of the presentation software (Caldiran et al., 2009).

With these careful consideration, we propose TALK-TO-YOUR-SLIDES, a system that separates these concerns into high-level and low-level components, as illustrated in Figure 2.

In the remainder of this section, we describe each components in detail.

3.1 Instruction understanding

The instruction understanding module operates at the high-level of our system architecture. It takes user instructions as input and interprets them into structured, actionable plans (Ruan et al., 2023) specifying which slides to modify, elements to target, and actions to perform as shown in Figure 2(a). The module outputs a structured list, where each entry explicitly details the target slide number, targeted element, and corresponding action, allowing for precise and versatile slide editing tasks as shown in Figure 3.

To handle diverse instructions that may target specific slides, subsets, or entire presentations, we utilize an LLM with carefully crafted prompts to support effective in-context learning (Dong et al., 2024; Guo et al., 2024a). The specific prompt used for instruction understanding is included in Appendix E.1. An example of this module's output and the corresponding original slide can be found in Figure 3 and Figure 14, respectively.

3.2 Document understanding

document understanding is a low-level component that accesses slides in our system. It plays a crucial role in slide editing tasks, as the quality of the parsed content essentially defines the set of editable elements and thereby determines an upper bound on the final editing accuracy.

To enable fine-grained document understanding, we develop a custom rule-based parser that extracts comprehensive information from each slide. This includes metadata such as the layout name, background fill type, and transition effects, as well as fine-level attributes of individual objects such as shapes, images, and text boxes. As shown in Figure 4, the parser identifies both the semantic type and positional information of each element on the slide. The parsed original slide that produced the results in Figure 4 is shown in Figure 14.

Recognizing that text formatting can vary within a single text box, we parse text at the run level, where each run denotes a contiguous segment of text with consistent formatting. This level of detail allows the system to more faithfully reflect how humans perceive and manipulate slides, thereby enabling precise, style-preserving edits.

All parsed outputs are converted into a structured JSON format. This representation facilitates downstream reasoning and editing, while ensuring compatibility with large language models. Prior work has shown that LLMs can better interpret and operate over structured data formats (He et al., 2024; Tan et al., 2025). Our parsing logic is fully implemented and publicly released as part of our codebase to support future research in structured document understanding. Additional details on document understanding are provided in Appendix D.


```
{ "contents": {
  "Presentation_Name": "Example.pptx",
  "Total_Slide_Number": "10",
  "Objects_Overview": "Found 1 object in slide number 1.",
  "Objects_Detail": [
   { "Object_number": 1,
     "Name": "Content Placeholder",
     "Type": "Placeholder",
     "Position_Left": 60.0,
     "Position_Top": 150.0,
     "Size_Width": 800.0,
     "Size_Height": 300.0,
     "Align": "Center",
     "More_detail": {
       "TextFrame": [
         { "RunIndex": 1,
           "Text": "Multi-agent systems can be improved by ",
           "Font": {
             "Name": "Arial",
             "Size": 24.0,
             "Bold": false,
             "Color": {"R": 0, "G": 0, "B": 0} } },
         { "RunIndex": 2,
           "Text": "prompt engineering",
           "Font": {
             "Name": "Arial",
             "Size": 24.0,
             "Bold": true,
             "Color": {"R": 255, "G": 0, "B": 0}
           } } ] } } ] } }
```


Figure 4: Example output of document understanding. The yellow sections contain information about the parsed object's name, type, location, size, and other details. The runs highlighted in green demonstrate that different text formatting styles can exist within a single text box.

3.3 Document editing

The document editing component performs high-level editing in our system. It takes as input the parsed output from the document understanding module and the editing description from the instruction understanding module. Its role is to modify the parsed content using an LLM in accordance with the editing description. For example, if the description specifies changing only the im-

TextEditing

- "Translate all visible text elements on ppt slide number {slide_num} into Japanese"
- "Review all text elements on ppt slide number {slide_num} for spelling, grammar, and typographical errors, and correct them."

VisualFormatting

- "Change the font of all text elements to Arial on ppt slide number {slide_num}."
- "Color code the keywords in the text while following the color theme in the slide number {slide_num}."

LayoutAdjustment

- "Center all titles at the top of each slide."
- "Resize all images on ppt slide number {slide_num} to have the same width while maintaining aspect ratio."

SlideStructure

- "Add slide numbers to every slide on the bottom right corner."
- "Divide the content of ppt slide number {slide_num} into two clear sections titled 'Overview' and 'Details' for improved structure."

Figure 5: Examples of instructions across four categories.

portant content in a text box to red, the document editing module identifies such content from the parsed data and generates output in the same format as that produced by the document understanding module. The prompt used for this module is provided in Appendix E.3.

3.4 Code generator

The primary function of code generator is to generate python code that applies the necessary modifications to the low-level instance. It receives the raw parsed data before edited, the data after edited from document editing module, and the plan. The generated codes are dependent on each presentation environment[1]. Consequently, the LLM is tasked with generating python code based on the semantics of presentation environment[2].

Separating the part that modifies actual slides at a low-level has a significant advantage. This modular design enhances platform adaptability. If the presentation application changes from PowerPoint to another platform, only the low-level component needs modification. The high-level reasoning remains intact. This minimizes accuracy degradation when transitioning between different presentation software.

In addition, we implement a self-reflection mechanism (Shinn et al., 2023) to handle execution failures. If an error occurs during execution, the error message and the generated code are appended to the original input, and inference is repeated until the modification succeeds or a predefined maximum number of iterations is reached. The prompt used for the code generator is described in Appendix E.3.

[1]In the case of PowerPoint, we adopt COM (Component Object Model, https://learn.microsoft.com/en-us/windows/win32/com/the-component-object-model)

[2]E.g, VBA API for Powerpoint https://learn.microsoft.com/en-us/office/vba/api/overview/powerpoint

Category	Text Editing	Visual Formatting	Layout Adjustment	Slide Structure	**Total**
Inst. #	16	19	15	6	56
Aug. #	160	190	150	60	560
Filtered	116	123	95	45	**379**

Table 1: Instruction count statistics by instruction category. The 'Aug. #' row indicates the GPT-4o-augmented dataset, while the 'Filtered' row represents the human-annotated, post-filtered dataset.

4 TSBench: Benchmark Dataset

Alongside proposing a novel system for editing presentation slides, we also introduce a benchmark dataset, TSBench designed to evaluate the slide editing capabilities of models or frameworks. In this section, we describe the construction process of the benchmark dataset in detail and present its key statistics.

4.1 Building the Benchmark

The proposed benchmark targets the task of editing PowerPoint slides. For this purpose, we first created a dataset of user instructions, and then developed slide data to which these instructions could be applied. In this section, we describe the construction methodology for each dataset component.

4.1.1 Instructions

To collect feasible and practical instructions, we first manually create 56 seed instructions that reflect plausible user commands. For each seed, we used GPT-4o (OpenAI, 2024) to generate 10 variations or paraphrases. These variations include, for example, replacing the target language in "Translate the slide into Chinese" with alternatives such as Japanese, French, or English. Some variations also involve paraphrasing while preserving the original intent. From the generated pool, we manually filtered out instructions lacking clear goals or evaluation criteria. Given that the precise interpretation of instructions significantly impacts evaluation outcomes, we implemented a manual review process where human evaluators examined all instructions to ensure clarity and validity. Only those with unambiguous objectives were retained.

System	Instruction	Performance metric							Efficiency metric		
		SR (%)	Instruction Following	Text	Image	Layout	Color	Exec. Time (s)	Avg. Input tokens	Avg. Output tokens	Avg. Cost ×0.001$
Direct code generation	TextEditing	62.07	0.00	0.10	1.70	1.70	1.70	23.08	1.21 k	0.93 k	0.7
	VisualFormatting	53.66	0.44	0.84	1.04	1.04	0.89	37.72	1.38 k	2.75 k	1.9
	LayoutAdjustment	58.95	0.66	1.55	0.79	1.42	1.40	23.25	1.26 k	0.94 k	0.8
	SlideStructure	73.33	0.50	1.61	1.25	1.58	1.72	28.17	1.00 k	1.05 k	0.8
	Overall	59.90	0.36	0.88	1.22	1.41	1.37	28.35	1.24 k	1.48 k	1.0
UI Agent	TextEditing	66.38	0.54	0.50	1.63	1.49	1.79	117.85	102.04 k	2.26 k	16.6
	VisualFormatting	86.18	**2.61**	2.01	1.72	2.25	1.80	122.25	93.27 k	2.15 k	15.2
	LayoutAdjustment	65.26	**2.07**	**2.82**	**2.38**	**2.51**	**2.87**	128.23	108.48 k	2.50 k	17.7
	SlideStructure	82.22	1.67	**2.31**	1.55	**2.38**	2.17	99.18	72.46 k	1.82 k	12.0
	Overall	74.41	1.64	1.81	1.83	2.11	2.10	119.66	97.29 k	2.23 k	15.9
Ours	TextEditing	99.14	**2.95**	**3.01**	**2.65**	**3.11**	**3.07**	55.98	3.81 k	1.96 k	1.6
	VisualFormatting	94.30	1.98	**2.22**	**1.86**	**2.38**	**2.16**	94.78	5.09 k	3.58 k	2.8
	LayoutAdjustment	100.00	1.80	2.39	2.15	2.37	2.56	86.14	4.46 k	2.29 k	2.0
	SlideStructure	91.10	**1.71**	1.95	**1.73**	2.17	**2.35**	96.54	2.71 k	1.69 k	1.2
	Overall	96.83	**2.21**	**2.48**	**2.17**	**2.58**	**2.57**	78.95	4.26 k	2.53 k	2.0

Table 2: System-wise scores by instruction category. "SR' 'denotes execution success rate. All three systems use the gemini-2.5-flash model. For UI Agent, we follows (Zhang et al., 2024a), imposing a cut-off for tasks that did not finish within 180 seconds and considering realistic feasibility constraints. Cost is in USD ($).

In total, we collected 379 instructions, which are categorized into four types: *TextEditing*, *VisualFormatting*, *LayoutAdjustment*, and *SlideStructure*. These categories are also used as evaluation dimensions in our subsequent analysis. Examples of instructions for each category are illustrated in Figure 5.

4.1.2 Slides

Constructing slide data tailored to a specific instruction is a non-trivial task. For instance, the instruction "Fix all typos in the slide" requires the slide to actually contain typographical errors, while "Translate the slide into Chinese" assumes the slide is written in another language which is not a Chinese.

To generate appropriate slides for each of the 379 instructions, we observed that each seed instruction and its GPT-4o-generated variants are associated with the same base PowerPoint file (OpenAI, 2024). Based on this insight, we manually created a PowerPoint presentation for each of the 56 seed instructions. To ensure visual quality and realism, we utilized publicly available templates[3] and we listed 10 template which we used in benchmark dataset in Table 5.

We release the full benchmark, including metadata that maps each instruction to its corresponding PowerPoint file and instruction group. We refered this map on Table 4 for detail.

[3]https://create.microsoft.com/en-us/search?filters=powerpoint

4.2 Statistics

Table 1 reports the number of instructions in each of the four categories. In the *Total* column, we observe that the original set of 56 human-authored seed instructions was expanded to 560 through GPT-4o-based augmentation. After excluding 181 examples with unclear objectives or those deemed unsuitable for benchmarking, a final set of 379 instruction–slide pairs remained. Consequently, TSBench comprises 379 instructions and 56 corresponding `.pptx` files. Detailed information, including the mapping between instructions and `.pptx` files, is provided in Appendix A, and slide content topics are listed in Table 3.

5 Experiment

In this section, we evaluate our proposed system, TALK-TO-YOUR-SLIDES, along with baseline methods using the benchmark dataset we introduced.

5.1 System Configuration

Direct code generation. We leverage the document understanding module (Section 3.2) to extract a structured representation of each slide, then prompt an LLM with this representation and the user instruction to generate executable editing code.

GUI agent. We include UFO2(Zhang et al., 2024a) as a representative GUI-based agent baseline, capable of operating a wide range of Windows applications, including PowerPoint. We used

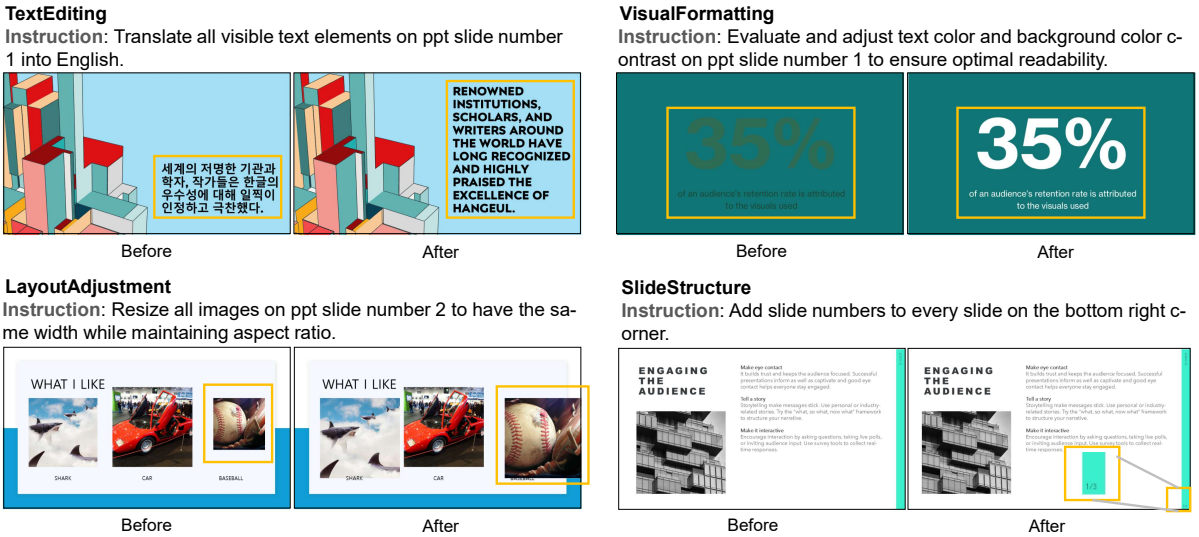

Figure 6: Results of TALK-TO-YOUR-SLIDES across four instruction categories. Modifications are highlighted with yellow boxes. In the *TextEditing* example, Korean text has been translated into English according to the instruction. In the *VisualFormatting* case, the original background and text colors were too similar, reducing readability; the revised version uses white text for improved contrast and clarity. In *LayoutAdjustment*, the widths of the three images have been unified while preserving their aspect ratios, as instructed. Lastly, in the *SlideStructure* example, page numbers have been added in response to the instruction.

Gemini-2.5-flash model in GUI agent. Additional configuration details are provided in Appendix C.

Talk-to-Your-Slides. In our proposed framework, the instruction understanding module is instantiated with Gemini-1.5-flash, while the document editing and code generation modules are evaluated with Gemini-2.5-flash. The code generator likewise performs up to three retries.

Additionally, we conducted experiments using GPT-4.1-mini in place of Gemini-2.5-flash across all system components. While the main results presented in this paper are based on the Gemini-2.5-flash configuration, results with GPT-4.1-mini are detailed in Appendix F. Full model descriptions and hyperparameter settings are provided in Appendix B.

5.2 Metrics

We adopt two primary categories of evaluation metrics.

Editing success metrics assess the system's effectiveness and include the *Execution Success Rate* (SR), *LLM judge scores* (Wang et al., 2025b), and *Execution Time*. First, SR indicates whether the finally generated code is successfully executed. Second, the LLM-based evaluation is crucial for our benchmark, as many instructions (summarization, emphasis, translation) lack definitive answers. It scores encompass instruction following, text, image, layout, and color aspects. Instruction following measures how effectively the edited presentation reflects the instruction. The remaining four metrics are established as reference-free metrics following (Ge et al., 2025). Scores range from 0 (worst) to 5 (best), with detailed criteria ensuring consistent interpretation. We employ the multimodal `gpt-4o` model for this evaluation. The model assesses edit quality by comparing original and edited slide images along with slide notes and instructions. The full scoring prompt appears in Figure 12 and 13. Finally, Execution time refers to the latency in seconds required to execute a single instruction.

Efficiency metrics capture the system's resource usage and include *Average Input Tokens*, *Average Output Tokens*, and *Average Cost*. All averages refer to those for a single instruction. Token counts are computed by aggregating all tokens passed to the LLM within each module, and the cost is then estimated accordingly. Table 2 reports these metrics using Gemini-2.5-flash in its non-thinking output mode, with cost computed based on the pricing as of May 16, 2025.[4] While editing

[4] As of 2025-05-16: $0.15 per million input tokens (text, image, video), $1.00 per million input tokens (audio), $0.60 per million output tokens (non-thinking), $3.50 per million output tokens (thinking).

accuracy is critical, efficiency is equally important for real-world applicability and user adoption of agent-based editing systems.

5.3 Results

This section analyzes the main experimental results. Then based on these findings, we discuss how agents should approach tasks that assist humans beyond PowerPoint editing when interacting with computing systems in general.

5.3.1 Main Results

Table 2 presents the results of PowerPoint editing systems evaluated on our proposed TSBench. Except for the execution success rate, all metrics are computed on the subset of examples for which the generated editing code executed successfully. Our proposed system, TALK-TO-YOUR-SLIDES, achieves the highest overall execution success rate and the highest average judge score. However, in the *LayoutAdjustment* categories, the GUI agent attains higher judge scores. In particular, for layout-related instructions, the GUI agent consistently received higher scores, suggesting that GUI-based operations offer advantages for layout manipulation. To justify the reliability of the LLM judge scores, we conducted an experiment on 30 instances, which yielded a Pearson correlation coefficient above 0.8 and similarly high Spearman correlation across all judge metrics as shown in Table 9. Four illustrative examples of slides edited by TALK-TO-YOUR-SLIDES is shown in Figure 6.

As for efficiency, our system requires just over one minute on average to edit slides. For the GUI agent, any run exceeding three minutes was deemed a failure, among successful runs, the mean execution time approaches two minutes. This difference is explained by the efficiency metrics: the GUI agent's average input token count is nearly twenty times larger than that of the other approaches, resulting in an average cost per instruction that is roughly ten times higher. These trends are stem from the use of image inputs, which is inherent to vision–language models.

5.3.2 Should Software Agents Ultimately Use Only GUI Images?

To better understand the trade-offs between GUI-based and code-based approaches in slide editing, we present concrete examples using the same instructions applied through different methods. When comparing the efficiency of GUI-based and code-based slide editing approaches, errors in VLM-based optical character recognition (OCR) provide persuasive evidence. The GUI agent, which relies on Vision Language Models (VLMs) for text recognition, failed to accurately identify text due to OCR limitations, resulting in low scores on text editing metrics. In contrast, the code-based system Talk-to-Your-Slides directly accessed text data without requiring VLM-based OCR processing. These results demonstrate that direct access to structured data is more reliable than external perception mechanisms like VLMs, highlighting how system accuracy depends on data accessibility.

Nevertheless, visual information from the GUI remains useful in software automation. There are editing tasks where purely low-level textual information is insufficient. For instance, when translating Chinese text into English, the translated content often expands in length, causing overflow beyond the original text box. In such cases, visual layout information helps preserve the slide's aesthetic quality. Here, GUI images complement the limitations of low-level approaches and can raise the upper bound of what can be achieved.

In this work, we demonstrated that a low-level, code-based approach is both effective and efficient for many core editing tasks in presentation software. Moving forward, we believe future research should explore hybrid approaches that combine the semantic precision and efficiency of structured parsing with the contextual awareness of visual understanding. We encourage the community to pursue this direction to further advance automated presentation editing systems.

6 Conclusion

This paper introduced TALK-TO-YOUR-SLIDES, an LLM-powered agent for editing slides through natural language instructions. By decomposing editing tasks into high-level semantic operations and low-level object manipulations, our system interacts with PowerPoint application to execute complex edits. We developed *TSBench*, a benchmark with 379 diverse editing instructions, to evaluate slide editing agents. Experiments show our approach significantly outperforms baselines in execution success, instruction following, and editing efficiency, demonstrating the effectiveness of our agent framework for automating slide editing tasks.

7 Limitations

Our current implementation focuses primarily on Powerpoint-based workflows, leveraging COM (Component Object Model) communication. While this architectural choice enabled us to deliver robust functionality for PowerPoint users, it currently offers limited support for alternative options. Although our high- and low-level implementation details are applicable to other presentation environments, we limited our experiments to PowerPoint in order to effectively compare with the other baseline that utilize a GUI.

References

Mohamed Aghzal, Erion Plaku, Gregory J. Stein, and Ziyu Yao. 2025. A survey on large language models for automated planning. *Preprint*, arXiv:2502.12435.

Ozan Caldiran, Kadir Haspalamutgil, Abdullah Ok, Can Palaz, Esra Erdem, and Volkan Patoglu. 2009. Bridging the gap between high-level reasoning and low-level control. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 342–354. Springer.

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2025. Unleashing the potential of prompt engineering for large language models. *Patterns*, page 101260.

Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024a. Towards tool use alignment of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1382–1400, Miami, Florida, USA. Association for Computational Linguistics.

Zhongwu Chen, Long Bai, Zixuan Li, Zhen Huang, Xiaolong Jin, and Yong Dou. 2024b. A new pipeline for knowledge graph reasoning enhanced by large language models without fine-tuning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1366–1381, Miami, Florida, USA. Association for Computational Linguistics.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

Difei Gao and 1 others. 2024. Assistgui: Task-oriented pc graphical user interface automation. In *CVPR*.

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

Tianyu Guo, Wei Hu, Song Mei, Huan Wang, Caiming Xiong, Silvio Savarese, and Yu Bai. 2024a. How do transformers learn in-context beyond simple functions? a case study on learning with representations. In *The Twelfth International Conference on Learning Representations*.

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

Yiju Guo, Ganqu Cui, Lifan Yuan, Ning Ding, Zexu Sun, Bowen Sun, Huimin Chen, Ruobing Xie, Jie Zhou, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024b. Controllable preference optimization: Toward controllable multi-objective alignment. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1437–1454, Miami, Florida, USA. Association for Computational Linguistics.

Yilun Hao, Yang Zhang, and Chuchu Fan. 2024. Planning anything with rigor: General-purpose zero-shot planning with llm-based formalized programming. *arXiv preprint arXiv:2410.12112*.

Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. 2024. Does prompt formatting have any impact on llm performance? *Preprint*, arXiv:2411.10541.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *Preprint*, arXiv:2308.10620.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *Preprint*, arXiv:2406.00515.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Thies Oelerich, Christian Hartl-Nesic, and Andreas Kugi. 2024. Language-guided manipulator motion planning with bounded task space. In *8th Annual Conference on Robot Learning*.

OpenAI. 2024. Gpt-4o api overview. https://platform.openai.com/docs/models/gpt-4o. Accessed: 2025-05-06.

Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in Text+Code LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11234–11258, Miami, Florida, USA. Association for Computational Linguistics.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, and 1 others. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Athar Sefid, Prasenjit Mitra, and Lee Giles. 2021. Slidegen: an abstractive section-based slide generator for scholarly documents. In *Proceedings of the 21st ACM Symposium on Document Engineering*, pages 1–4.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhihong Sun, Chen Lyu, Bolun Li, Yao Wan, Hongyu Zhang, Ge Li, and Zhi Jin. 2024. Enhancing code generation performance of smaller models by distilling the reasoning ability of LLMs. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 5878–5895, Torino, Italia. ELRA and ICCL.

Xiaoyu Tan, Haoyu Wang, Xihe Qiu, Leijun Cheng, Yuan Cheng, Wei Chu, Yinghui Xu, and Yuan Qi. 2025. Struct-x: Enhancing the reasoning capabilities of large language models in structured data scenarios. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, page 2584–2595, New York, NY, USA. Association for Computing Machinery.

Evan Z Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean M. Hendryx, Summer Yue, and Hugh Zhang. 2025a. Planning in natural language improves LLM search for code generation. In *The Thirteenth International Conference on Learning Representations*.

Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. 2025b. Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering. *arXiv preprint arXiv:2502.06193*.

Paiheng Xu, Gang Wu, Xiang Chen, Tong Yu, Chang Xiao, Franck Dernoncourt, Tianyi Zhou, Wei Ai, and Viswanathan Swaminathan. 2025. Skill discovery for software scripting automation via offline simulations with llms. *Preprint*, arXiv:2504.20406.

Dayu Yang, Tianyang Liu, Daoan Zhang, Antoine Simoulin, Xiaoyi Liu, Yuwei Cao, Zhaopu Teng, Xin Qian, Grey Yang, Jiebo Luo, and Julian McAuley. 2025. Code to think, think to code: A survey on code-enhanced reasoning and reasoning-driven code intelligence in llms. *Preprint*, arXiv:2502.19411.

Ke Yang, Jiateng Liu, John Wu, Chaoqi Yang, Yi Fung, Sha Li, Zixuan Huang, Xu Cao, Xingyao Wang, Heng Ji, and ChengXiang Zhai. 2024. If LLM is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Oleksandr Polozov, and Charles Sutton. 2023. Natural language to code generation in interactive data science notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 126–173, Toronto, Canada. Association for Computational Linguistics.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2023. Large language models meet nl2code: A survey. *Preprint*, arXiv:2212.09420.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, and 1 others. 2024a. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*.

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

Hongyu Zhao, Kangrui Wang, Mo Yu, and Hongyuan Mei. 2023. Explicit planning helps language models in logical reasoning. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

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. *Preprint*, arXiv:2501.03936.

A Detail of TSBench

In this section, we present detailed information about TSBench. First, we have listed the topics of slide content in Table 3.

Additionally, the mapping between instruction numbers, slide IDs, and the four categories is listed in Table 4. Each instruction is assigned an `instruction_key` of the form `n` or `n-m`, where `n` denotes one of the original 56 seeds, and `n-m` indicates the mth augmentation derived from seed `n`. The PowerPoint files are named `slide_<instruction_key>.pptx` (e.g., `slide_0.pptx`, `slide_3-1.pptx`), with the suffix matching the corresponding `instruction_key`.

The list of slide templates is enumerated in Table 5 along with their corresponding links.

Examples of TSBench slides and instructions are presented in Figure 7.

B Model

In this section, we provide details about the external LLM APIs used in the experiments. Gemini-2.5-flash and GPT-4.1-mini were embedded in the agent system, while GPT-4o was used as a judge when evaluating the system. The price of each models is illustrated in Table 6.

Gemini-1.5-flash (gemini-1.5-flash) Gemini 1.5 Flash offers cost-effective multimodal capabilities with tiered pricing based on context length. Released in early 2024, this model represents Google's focus on balancing accuracy with efficiency. It supports a context window of up to 1,048,576 tokens, max output token is 8,192, allowing for processing of extensive documents and conversations in a single request. The model handles multiple modalities including text, code, images, and limited audio processing. While not as powerful as its larger counterparts in complex reasoning tasks, it demonstrates strong capability in straightforward instruction following, summarization, and multimodal understanding tasks. We used this model for instruction understanding with maxtoken: 2048, temperature 0.05, and top_p 1.0.

Gemini-2.5-flash (gemini-2.5-flash-preview-04-17). Gemini 2.5 Flash is a high-throughput thinking model designed to strike an optimal balance between speed, cost, and reasoning capability. As Google's latest preview-tier model, it extends the popular 2.0 Flash foundation with major upgrades in reasoning capability, while still prioritizing low latency and economical usage for developers. It supports a wide range of modalities—including text, code, images, audio, and video—making it well suited for diverse AI tasks where both multimodal understanding and cost efficiency are critical. We used this model for document editing and code generation with the maximum supported token limit of 65536, a temperature of 0.05, and top_p of 1.0.

GPT-4.1-mini (gpt-4.1-mini-2025-04-14). GPT-4.1-mini is the 'mini' variant of the GPT-4.1 family, released April 14, 2025. It inherits the core strengths of the flagship GPT-4.1 series—state-of-the-art coding ability, robust instruction following, and support for very long (up to one million-token) contexts—while reducing model size to cut inference latency by roughly 50% and lower operational cost. This makes GPT-4.1-mini an ideal choice for applications that demand the latest model capabilities in a more resource-efficient footprint. We used this model for document editing and code generation with the maximum supported token limit of 32768, a temperature of 0.05, top_p 1.0.

GPT-4o (gpt-4o-2024-08-06). GPT-4o ("o" for "omni") is OpenAI's multimodal flagship, released August 6, 2024. It can ingest and generate text, images, and audio in real time, enabling unified reasoning across these modalities. Compared to its predecessor (GPT-4 Turbo), GPT-4o offers faster API throughput and lower per-token cost, making it especially powerful for tasks that require seamless cross-modal understanding and generation. We used this model as an LLM judge with a max token of 512, a temperature of 0.2, top_p 1.0.

C UFO: GUI Agent

We employ UFO2 (Zhang et al., 2024a), a state-of-the-art multi-agent GUI automation system for Windows desktops, as one of our baseline methods. UFO2 is designed to execute natural language instructions by integrating both visual and API-based control over various applications. It features a centralized *HostAgent* for task decomposition and orchestration, and multiple *AppAgents* tailored to specific applications such as PowerPoint.

In our experimental setup, both the HostAgent and AppAgent are powered by the vision-language model Gemini-2.5-flash. The system is further backed by a retry mechanism with up to three fallback attempts to ensure robust execution. This

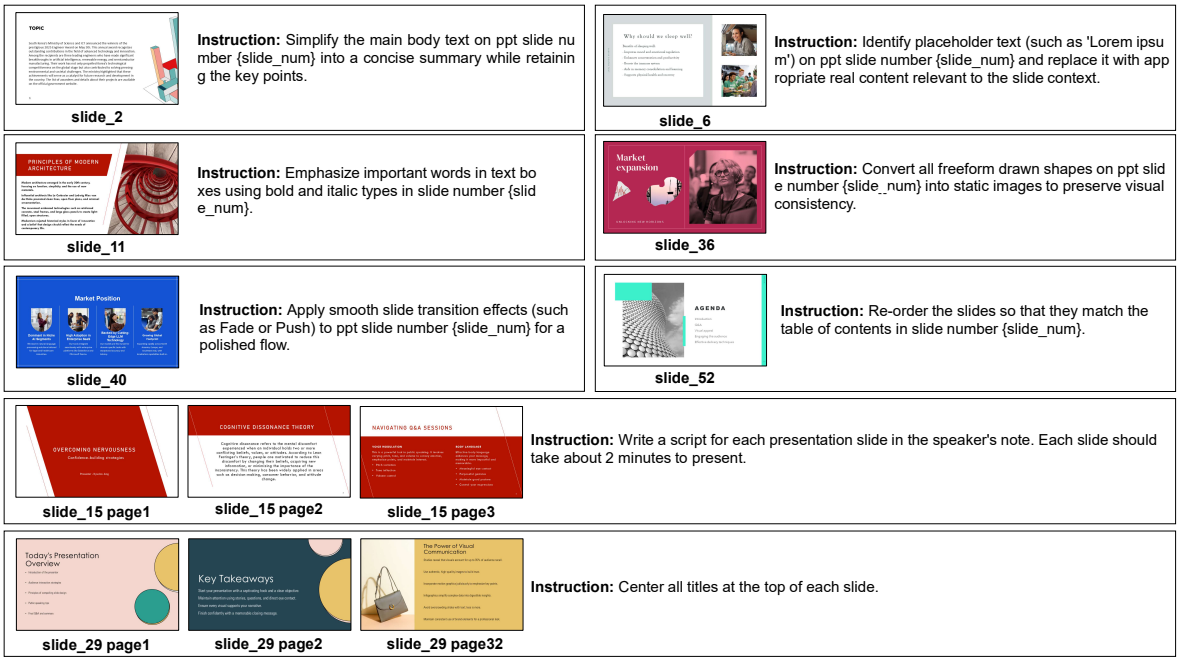

Figure 7: Example from the TSBench dataset. Some data points consist of a single slide, while others contain multiple slides.

Table 3: Content topics of slides in TSBench dataset.

Index	Topic	Index	Topic	Index	Topic
0	Linguistics	19	Economics	38	AI Strategy
1	Communication	20	Climate Change	39	Marketing
2	News Articles	21	Quotes	40	Marketing
3	Presentation	22	Some Numbers	41	Marketing
4	Remote Work	23	Aesthetics	42	Company
5	Data Collection	24	Presentation	43	Company
6	Sleeping	25	Presentation	44	Marketing
7	LLMs	26	Design	45	Financials
8	Hypertension	27	Q&A	46	Competitive Landscape
9	Impressionism	28	Presentation	47	Product Overview
10	Immanuel Kant	29	Visual Communication	48	AI Assistant Platform
11	Modern Architecture	30	Presentation Theme	49	Future Outlook
12	Aesthetics	31	What I Like	50	Design
13	Education	32	What I Like	51	Presentation
14	Cognitive Dissonance	33	Creative Vision	52	Visual Appeal
15	Nervousness	34	Sports	53	Design
16	Linear Algebra	35	Marketing Strategies	54	NLP
17	Artificial Intelligence	36	Marketing	55	Presentation
18	Economics	37	Marketing		

configuration enables UFO2 to leverage its hybrid GUI–API action interface, speculative multi-action execution, and a Picture-in-Picture (PiP) sandbox for non-intrusive user experience.

For additional technical details and evaluation results, we refer readers to the original UFO2 paper (Zhang et al., 2024a).

D Details of document understanding

Although a `.pptx` file is technically a collection of XML files, directly working with these raw XML structures is highly impractical for document-level understanding. The XML files are excessively verbose, and their content is fragmented across multiple interlinked components. This makes parsing both time-consuming and error-prone.

XML format of PowerPoint is also inherently index-based. For example, attributes such as text formatting (e.g., bold, italic) are not stored as human-readable strings but instead encoded as numeric indices that point to style definitions in separate lookup tables. As a result, even simple queries such as "is this text bolded?" require multi-step resolution across files.

To mitigate these challenges, Zheng et al. (2025) proposed converting slides into HTML format. While this method simplifies parsing to some extent, it introduces an additional rendering step and often fails to retain the full range of visual and structural information available in the original slide-particularly layout metadata, positional precision, and fine-grained formatting.

In contrast, our system avoids HTML-based conversion and instead directly parses the PowerPoint object model through COM (Component Object Model) interfaces. This design choice enables precise extraction of layout, style, and object-level data with full fidelity to the original file. The extracted information is then normalized into a structured JSON format to support downstream semantic reasoning and editing tasks.

E Prompt

In this section, we present the prompts used for the LLMs in our experiments. Following Chen et al. (2025) and Liu et al. (2024), we carefully designed detailed prompts. As experimental outcomes can vary significantly depending on subtle differences in prompts, we disclose our prompts in full to ensure specificity and reproducibility.

E.1 instruction understanding prompt

In the instruction understanding stage, the system interprets the user's intent and formulates a plan (Aghzal et al., 2025; Oelerich et al., 2024; Hao et al., 2024). The prompt of this is shown in Figure 8.

Table 4: Category mapping of slides in TSBench dataset.

Index	Category	Index	Category	Index	Category
0	TextEditing	19	TextEditing	38	LayoutAdjustment
1	TextEditing	20	VisualFormatting	39	LayoutAdjustment
2	TextEditing	21	VisualFormatting	40	VisualFormatting
3	TextEditing	22	VisualFormatting	41	VisualFormatting
4	TextEditing	23	VisualFormatting	42	VisualFormatting
5	TextEditing	24	VisualFormatting	43	LayoutAdjustment
6	TextEditing	25	VisualFormatting	44	LayoutAdjustment
7	TextEditing	26	VisualFormatting	45	VisualFormatting
8	TextEditing	27	VisualFormatting	46	SlideStructure
9	TextEditing	28	VisualFormatting	47	SlideStructure
10	TextEditing	29	LayoutAdjustment	48	SlideStructure
11	VisualFormatting	30	LayoutAdjustment	49	LayoutAdjustment
12	VisualFormatting	31	LayoutAdjustment	50	SlideStructure
13	TextEditing	32	LayoutAdjustment	51	SlideStructure
14	VisualFormatting	33	LayoutAdjustment	52	SlideStructure
15	TextEditing	34	LayoutAdjustment	53	VisualFormatting
16	VisualFormatting	35	LayoutAdjustment	54	TextEditing
17	LayoutAdjustment	36	LayoutAdjustment	55	TextEditing
18	VisualFormatting	37	LayoutAdjustment		

Instruction understanding prompt

You are a planning assistant for PowerPoint modifications.
Your job is to create a detailed, specific, step-by-step plan for modifying a PowerPoint presentation based on the user's request.
present ppt state: {get_simple_powerpoint_info()} Break down complex requests into highly specific actionable tasks that can be executed by a PowerPoint automation system.
Focus on identifying:
1. Specific slides to modify (by page number)
2. Specific sections within slides (title, body, notes, headers, footers, etc.)
3. Specific object elements to add, remove, or change (text boxes, images, shapes, charts, tables, etc.)
4. Precise formatting changes (font, size, color, alignment, etc.)
5. The logical sequence of operations with clear dependencies
Please write one task for one slide page.
Format your response as a JSON format with the following structure: {{ "understanding": "Detailed summary of what the user wants to achieve", "tasks": [{{ "page number": 1, "description": "Specific task description", "target": "Precise target location (e.g., 'Title section of slide 1', 'Notes section of slide 3', 'Second bullet point in body text', 'Chart in bottom right')", "action": "Specific action with all necessary details", "contents": {{ "additional details required for the action" }} }}, ...], }}
Below is the example question and example output.
input: Please translate the titles of slide 3 and slide 5 of the PPT into English.
output: {{ "understanding": "English translation of slide titles on slides 3 and 5", "tasks": [{{ "page number": 3, "description": "Translate the title text of slide 3", "target": "Title section of slide 3", "action": "Translate to English", "contents": {{ "source_language": "auto-detect", "preserve_formatting": true }} }}, {{ "page number": 5, "description": "Translate the title text of slide 5", "target": "Title section of slide 5", "action": "Translate to English", "contents": {{ "source_language": "auto-detect", "preserve_formatting": true }} }}], }}
Response in JSON format.

Response: JSON

Figure 8: A prompt used in instruction understanding.

E.2 document editing prompt

In the document editing stage, the system generates the post-editing data in JSON format based on the plan and the parsed data. This process is illustrated in Figure 9.

E.3 Code generator prompt

In the code generation stage, the system takes as input the original slide data, the document-edited slide data, and the plan, and outputs Python code that applies the corresponding changes in PowerPoint. The prompt used for this step is shown in Figure 10.

Document editing prompt

Information about slide {page_number}:
- Task description: {description}
- Action type: {action}
- Slide contents: {contents}

You are a specialized AI that analyzes PowerPoint slide content and performs specific tasks. You will receive the following JSON data, perform the designated tasks, and return the results in exactly the same JSON format.
Important rules: 1. You must maintain the exact input JSON structure
2. Only perform the work described in the 'action' within 'tasks'
3. Only modify the elements specified in 'target' within 'tasks'
4. Output must contain pure JSON only - no explanations or additional text
5. Preserve all formatting information (fonts, sizes, colors, etc.)
6. Verify that the JSON format is valid after completing the task
Before starting the task:
1. Check the 'understanding' field to grasp the overall task objective
2. Review 'page number', 'description', 'target', and 'action' within 'tasks'
3. Identify all text elements in 'Objects_Detail'
The output must maintain the identical structure as the original JSON, with only the necessary text modified according to the task.
Give only the JSON.

Response: JSON

Figure 9: A prompt used in Document Editing.

Code generator prompt

Generate Python code modify an active PowerPoint presentation based on the provided JSON task data. The code should:
0. Find activate powerpoint app with ppt_app
= win32com.client.GetActiveObject ("PowerPoint.Application")
active_presentation = ppt_app.ActivePresentation
1. Find the slide specified by page number: {slide_num}
2. Target to change: {before}
3. New content to apply: {after}
4. Generate ONLY executable code that will directly modify the PowerPoint.
CRITICAL REQUIREMENTS:
- DO NOT create a new PowerPoint application - use the existing one
- Please check if the slide number you want to work on exists and proceed with the work. The index starts with 1.
- The code should NOT be written as a complete program with imports - it will be executed in an environment where PowerPoint is already open
- Focus on finding and modifying the specified content
- For text changes, use both shape.Name and TextFrame.TextRange.Text to identify the correct element
- Make sure to explicitly apply any changes (e.g., shape.TextFrame.TextRange.Text = new_text)
- Do not write print function or comments.
- You can write at slide note with slide.NotesPage
"'python slide.NotesPage.Shapes.Placeholders(2). TextFrame.TextRange.Text = notes_text "' Note that the code will run in a context where these variables are already available:
- ppt_application: The PowerPoint application instance
- active_presentation: The currently open presentation
IMPORTANT: In PowerPoint, color codes use BGR format (not RGB). For example, RGB(255,0,0) for red should be specified as RGB(0,0,255) in the code. Always convert any color references accordingly.
If you want to modify the formatting, refer to the following code for modification:
if text_frame.HasText: text_range = text_frame.TextRange # Find text found_range = text_range.Find(text_to_highlight) while found_range: found_any = True found_range.Font.Bold = True # Bold found_range.Font.Color.RGB = 255 # Example color (RED in BGR format - 0,0,255) found_range.Font.Size = found_range.Font.Size * 1.2 # Increase font size by 20% start_pos = found_range.Start + len(text_to_highlight) found_range = text_range.Find(text_to_highlight, start_pos)
Do not use any "**" to make bold. It won't be applied on powerpoint. - You can add or split a page with 'presentation = ppt_app.Presentations.Add()'.
Make sure to close all curly braces properly and all variables used are properly defined. Omit Strikethrough, Subscript, Superscript as they caused issues.
The code must be direct, practical and focused solely on making the specific change requested. Ensure all color references use the BGR format for proper appearance in PowerPoint.

Response: Python code

Figure 10: A prompt used in Code Generator.

Template	Link	File Size (KB)
Architecture pitch deck	Link	3,820
Classic frame design	Link	3,054
Creative perspective presentation	Link	13,832
Helena design	Link	12,023
Light modernist design	Link	7,601
Modern geometry design	Link	9,701
PowerPoint party	Link	6,029
Rose suite presentation	Link	1,438
Simple company overview presentation	Link	7,254
Vivid circles presentation	Link	2,966

Table 5: Microsoft Create PowerPoint templates: direct search links and downloaded file sizes.

E.4 Direct code generation prompt

In the case of the Direct code generation, the system directly generates code using only the parsed slide data and the instruction, without intermediate planning or editing. The prompt used for this process is shown in Figure 11.

E.5 LLM judge prompt

To evaluate slide editing capabilities, we employed an LLM-based judge, following a similar approach to Ge et al. (2025). The prompt used to assess how well the instruction was followed is shown in Figure E.5, while the prompt used for evaluating text, image, layout, and color,based on the criteria from Ge et al. (2025), is presented in Figure 13.

F Auxiliary results

In this section, we report the results of our experiments conducted using gpt-4.1-mini instead of gemini-2.5-flash. The results are presented in Table F, and the correlation coefficients for the metrics assessed by the LLM judge are reported in Table 8.

We also report the human correlation results for the Gemini-2.5-flash experiments, where model outputs were evaluated using LLM-based judges. These correlation analyses are presented in Figure 9.

G Auxiliary image

The original slide is presented in Figure 14 from which the example parsed data Figure 4 was extracted.

Table 6: Pricing information for LLM models used in experiments (in USD per million tokens).

Model	Input	Cached Input	Output	Additional Features
Gemini-2.5-flash	$0.15[a]	$0.0375[a]	$0.60 / $3.50[b]	Google Search[c]
Gemini-1.5-flash	$0.075 / $0.15[d]	$0.01875 / $0.0375[d]	$0.30 / $0.60[d]	Storage: $1.00/hr
GPT-4.1-mini	$0.40	$0.10	$1.60	–
GPT-4o	$2.50	$1.25	$10.00	–

[a] $1.00 for audio input, $0.25 for cached audio input

[b] $0.60 without thinking mode, $3.50 with thinking mode

[c] Free up to 1,500 RPD, then $35 per 1,000 requests

[d] Lower price for contexts <128K tokens, higher price for contexts >128K tokens

Baseline prompt

The following is information parsed from a PPT slide.
{parsed data}
Create a Python code with win32com library that can edit PowerPoint presentations by executing the following command:
{instruction}
IMPORTANT: Your response must contain ONLY valid Python code wrapped in triple backticks with the 'python' language tag. Follow this exact format:
```python
# Your Python code here
# Include proper comments, imports, and function definitions
# No explanations or text outside this code block
```

Response: Python code

Figure 11: A prompt used in baseline system.

LLM Judge prompt

You are an expert slide-editing judge.
TASK
- Compare the ORIGINAL slide with the EDITED slide.
- Decide how well the EDITED slide follows the INSTRUCTION and how aesthetically pleasing it is.
SCORING
Return valid JSON with exactly these keys:
{instruction_adherence <int 0-5>,
visualquality <int 0-5>
}
GUIDELINES
Score each from 0 to 5, based on the following rubric:
5 = Perfect: Fully satisfies the instruction / visually excellent with no flaws.
4 = Mostly correct: Clearly reflects the instruction / visually strong but with minor flaws.
3 = Partially correct: Instruction was followed to a noticeable degree, but key aspects are missing or flawed / visual layout or formatting needs improvement.
2 = Slightly changed but inadequate: Some edits related to the instruction are present but insufficient or poorly done / visual design is lacking.
1 = Attempted but incorrect: Some change is visible, but it does not match the instruction / visual result is clearly poor.
0 = Completely fails: No meaningful attempt to follow the instruction / visually broken or irrelevant.
Judge only what you can see in the given image(s) and notes.
Return *only* the JSON object, nothing else.

Response: Python code

Figure 12: A prompt used in LLM judge.

LLM Judge Text, Image, Layout, Color evaluation prompt

You are an expert slide-editing judge.
TASK
- Compare the ORIGINAL slide with the EDITED slide.
- Evaluate how well the EDITED slide handles Text, Image, Layout, and Color aspects based on the INSTRUCTION.
SCORING
Return valid JSON with exactly these keys:
{ text_quality <int 0-5>,
image_quality <int 0-5>,
layout_quality <int 0-5>,
color_quality <int 0-5>
} GUIDELINES
Score each from 0 to 5, based on the following rubric:
TEXT QUALITY:
5 = Perfect: Text content, formatting, and typography are flawless and fully satisfy the instruction.
4 = Mostly correct: Text elements are clearly improved but have minor issues in content, formatting, or typography.
3 = Partially correct: Text improvements are noticeable but have significant issues in content, formatting, or typography.
2 = Slightly changed but inadequate: Some text edits are present but insufficient or poorly implemented.
1 = Attempted but incorrect: Text changes are visible but do not match the instruction or improve the slide.
0 = Completely fails: No meaningful text improvements or changes are severely detrimental.
IMAGE QUALITY:
5 = Perfect: Images are optimal in selection, placement, sizing, and enhancement, fully satisfying the instruction.
4 = Mostly correct: Images are well-selected and implemented with only minor issues in placement, sizing, or visual quality.
3 = Partially correct: Image improvements are noticeable but have significant issues in selection, placement, sizing, or quality.
2 = Slightly changed but inadequate: Some image edits are present but insufficient or poorly implemented.
1 = Attempted but incorrect: Image changes are visible but do not match the instruction or improve the slide.
0 = Completely fails: No meaningful image improvements or changes are severely detrimental.
LAYOUT QUALITY:
5 = Perfect: Slide organization, spacing, alignment, and element relationships are flawless and fully satisfy the instruction.
4 = Mostly correct: Layout is clearly improved but has minor issues in organization, spacing, or alignment.
3 = Partially correct: Layout improvements are noticeable but have significant issues in organization, spacing, or alignment.
2 = Slightly changed but inadequate: Some layout edits are present but insufficient or poorly implemented.
1 = Attempted but incorrect: Layout changes are visible but do not match the instruction or improve the slide.
0 = Completely fails: No meaningful layout improvements or changes are severely detrimental.
COLOR QUALITY:
5 = Perfect: Color scheme, contrast, balance, and emphasis are flawless and fully satisfy the instruction.
4 = Mostly correct: Color choices are clearly improved but have minor issues in scheme, contrast, or emphasis.
3 = Partially correct: Color improvements are noticeable but have significant issues in scheme, contrast, or emphasis.
2 = Slightly changed but inadequate: Some color edits are present but insufficient or poorly implemented.
1 = Attempted but incorrect: Color changes are visible but do not match the instruction or improve the slide.
0 = Completely fails: No meaningful color improvements or changes are severely detrimental.
Judge only what you can see in the given image(s) and notes.
Return *only* the JSON object, nothing else.

Response: text: integer, image: integer, layout: integer, color:integer

Figure 13: A prompt used in LLM judge which evaluate text, image, layout, color.

System	Instruction	Performance metric							Efficiency metric		
		SR (%)	Instruction Following	Text	Image	Layout	Color	Exec. Time (s)	Avg. Input tokens	Avg. Output tokens	Avg. Cost ×0.001$
Direct code generation	TextEditing	76.72	0.00	0.00	1.04	1.04	1.04	18.55	1.03 k	0.41 k	1.0
	VisualFormatting	78.05	0.53	0.73	1.06	1.02	0.86	18.08	1.16 k	0.45 k	1.1
	LayoutAdjustment	69.47	0.07	1.45	1.09	1.40	1.40	17.06	1.04 k	0.53 k	1.2
	SlideStructure	84.44	0.17	1.78	1.11	1.91	1.64	19.57	0.91 k	0.65 k	1.4
	Overall	76.25	0.53	0.81	1.07	1.23	1.15	18.19	1.06 k	0.48 k	1.2
UI Agent	TextEditing	65.48	0.48	0.67	1.58	1.48	1.56	110.84	101.48 k	2.10 k	15.9
	VisualFormatting	88.09	**2.44**	1.89	1.67	1.79	1.48	134.57	94.12 k	2.44 k	14.9
	LayoutAdjustment	62.64	**1.48**	**2.76**	**2.20**	2.23	2.45	127.13	121.04 k	2.47 k	18.8
	SlideStructure	82.30	**1.55**	2.27	1.30	2.08	1.99	95.07	75.45 k	1.77 k	11.9
	Overall	73.84	1.68	1.94	1.55	2.16	2.04	121.08	98.22 k	2.30 k	15.4
Ours	TextEditing	98.28	**2.6**	**3.01**	**3.09**	**3.54**	**3.6**	58.17	3.35 k	1.66 k	3.3
	VisualFormatting	95.93	2.00	**2.02**	**1.71**	**2.25**	**1.88**	89.99	4.24 k	2.15 k	4.5
	LayoutAdjustment	97.89	1.40	2.37	1.81	**2.26**	**2.48**	89.86	3.99 k	2.10 k	4.3
	SlideStructure	91.11	1.40	**2.44**	**2.59**	**2.78**	**2.9**	74.39	2.24 k	1.28 k	2.1
	Overall	**96.57**	**2.13**	**2.46**	**2.26**	**2.71**	**2.68**	78.37	3.60 k	1.89 k	3.8

Table 7: System-wise scores by instruction category. 'SR' denotes execution success rate. All three systems use the GPT-4.1-mini model. For UI Agent, we follows (Zhang et al., 2024a), imposing a cut-off for tasks that did not finish within 180 seconds and considering realistic feasibility constraints. Cost is in USD ($).

Correlation Coefficient	Instruction	Text	Image	Layout	Color
Pearson	0.93	0.94	0.89	0.78	0.88
Spearman	0.91	0.92	0.85	0.74	0.86

Table 8: Correlation coefficients between LLM judge–based metrics and human evaluation in the main experiment using GPT-4.1-mini. All p-values are below 10^{-3}.

Correlation Coefficient	Instruction	Text	Image	Layout	Color
Pearson	0.92	0.95	0.95	0.87	0.94
Spearman	0.91	0.93	0.95	0.84	0.88

Table 9: Correlation coefficients between LLM judge–based metrics and human evaluation in the main experiment using Gemini-2.5-flash. All p-values are below 10^{-3}.

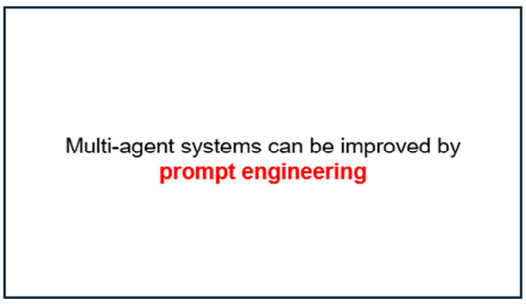

Figure 14: The original slide from which the example parsed data was extracted.