TALK TO YOUR SLIDES: High-Efficiency Slide Editing via Language-Driven Structured Data Manipulation

Kyudan Jung[1], Hojun Cho[2], Jooyeol Yun[2], Soyoung Yang[2], Jaehyeok Jang[1], Jaegul Choo[2],

{kyudan, hojun.cho, blizzard072, sy_yang}@kaist.ac.kr, achilloaaa@cau.ac.kr
jchoo@kaist.ac.kr

Abstract

Editing presentation slides is a frequent yet tedious task, ranging from creative layout design to repetitive text maintenance. While recent GUI-based agents powered by Multimodal LLMs (MLLMs) excel at tasks requiring visual perception, such as spatial layout adjustments, they often incur high computational costs and latency when handling structured, text-centric, or batch processing tasks. In this paper, we propose TALK-TO-YOUR-SLIDES, a high-efficiency slide editing agent that operates via language-driven structured data manipulation rather than relying on the image modality. By leveraging the underlying object model instead of screen pixels, our approach ensures precise content modification while preserving style fidelity, addressing the limitations of OCR-based visual agents. Our system features a hierarchical architecture that effectively bridges high-level user instructions with low-level execution codes. Experiments demonstrate that for text-centric and formatting tasks, our method enables 34% faster processing, achieves 34% better instruction fidelity, and operates at an 87% lower cost compared to GUI-based baselines. Furthermore, we introduce TSBENCH, a human-verified benchmark dataset comprising 379 instructions, including a HARD subset designed to evaluate robustness against complex and visually dependent queries. Our code and benchmark are available at **here**.

1 Introduction

Recent advancements in large language models (LLMs) have revolutionized software automation, demonstrating remarkable success in code generation (Yang et al., 2024; Hou et al., 2024; Xu et al., 2025), GUI navigation (Zhang et al., 2024a; Hong et al., 2024), and slide generation (Sefid et al., 2021; Zheng et al., 2025). However, slide tasks are often bifurcated into *creative design* and *content maintenance*. While automated generation has garnered attention, there is a growing demand for the latter, efficiently editing existing slides to reflect updated information or formatting. Slides serve as a fundamental medium for communication, yet modifying them often demands tedious, time-consuming manual effort, particularly for text-centric batch processing.

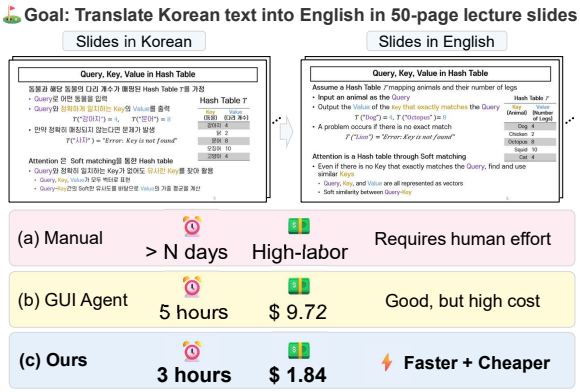

Figure 1: Comparison of slide editing methods on translating 50-page lecture slides from Korean to English. (a) Manual translation requires day(s) and consumes graduate-student labor. (b) A GUI-based agent incurs high cost. (c) Our approach runs in a low cost and in a relatively short time.

For example, as shown in Figure 1, a professor prepares an international lecture containing 50 slides and must translate them from Korean to English while strictly preserving technical terminology and formatting. Similarly, a marketing team needs to update product pricing on 120 slides spanning several presentations before a major launch. In these scenarios, manual effort is prone to errors and inefficiency.

Several candidate approaches can address these challenges. One straightforward approach is leveraging vision-based GUI agents that operate on screen-captured images through mouse and keyboard interactions (Zhang et al., 2024a; Microsoft,

Findings of the Association for Computational Linguistics: ACL 2026, pages 3370–3399
July 2-7, 2026

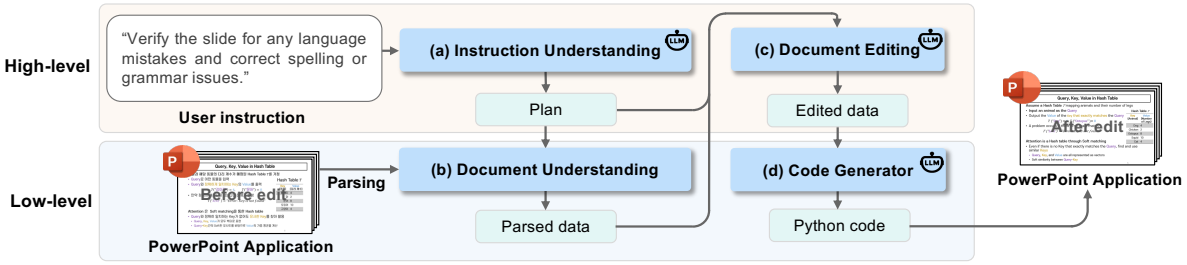

Figure 2: Overview of the TALK-TO-YOUR-SLIDES framework. The system consists of four modules: instruction understanding, document understanding, document editing, and code generator.

2024). While these agents excel at spatial tasks, applying them to text-heavy or batch-editing tasks reveals significant limitations due to the high computational cost of processing image inputs and the potential loss of fidelity in text recognition. This leads to our first research question **(RQ1)**: ***Can a non-visual, structure-aware agent achieve superior efficiency and fidelity for text-centric slide editing compared to visual approaches?*** We acknowledge that PowerPoint is an inherently visual medium; a purely non-visual agent cannot fully address layout stability (e.g., text overflow after translation) or subjective aesthetic judgments. Our scope is therefore deliberately focused on *structured, text-centric, and batch processing tasks*, where language-driven manipulation offers clear efficiency gains.

Another approach involves converting natural language instructions into direct scripting code. However, simple baselines struggle with complex tasks requiring contextual interpretation. For instance, instructions such as "Summarize the text on all slides and highlight key points in red" demand understanding both user intent and the sequential context of slide content. We therefore pose our second research question **(RQ2)**: ***What system architecture enables the effective decomposition of complex instructions into precise executable steps?***

In this paper, we answer our two research questions and propose TALK-TO-YOUR-SLIDES, an LLM-powered agent designed for efficient, high-fidelity slide editing. Unlike generalist visual agents, our method operates by leveraging the underlying structured data (e.g., XML, VBA object) of slide objects. This approach enables more accurate and cost-effective editing by avoiding the overhead of image processing. Experiments demonstrate that our method is significantly more cost-effective than UFO (Zhang et al., 2024a), the current state-of-the-art open-source UI agent.

We design the agent architecture with distinct high-level and low-level layers to facilitate interaction between user commands and slide objects. This hierarchical design ensures that complex logical instructions are accurately translated into executable code, drawing from insights on planning and reasoning processes (Guo et al., 2024b; Chen et al., 2024). By providing direct access to application objects, we achieve faster processing and better instruction fidelity compared to GUI-based methods.

Moreover, to complement benchmarks focused on visual aesthetics (Ge et al., 2025; Zheng et al., 2025), we present *TSBench*, a human-verified benchmark designed to evaluate slide editing capabilities. TSBench consists of 379 diverse editing instructions, covering text editing, visual formatting, layout adjustment, and structure manipulation. Crucially, to address task complexity, we include a robust subset, TSBench-Hard, which challenges the agent with visually dependent or ambiguous tasks, enabling a systematic assessment of the agent's reasoning capabilities.

Our contributions are summarized as follows:

- We demonstrate that for **text-centric and batch editing tasks**, a language-driven structural agent significantly outperforms GUI-based agents in speed, cost-efficiency, and instruction fidelity.
- We introduce TALK-TO-YOUR-SLIDES, a hierarchical framework specifically designed to bridge natural language instructions with low-level slide objects.
- We construct *TSBench* (including the challenging *Hard* subset), a benchmark that en-

ables systematic evaluation of slide editing agents beyond simple generation tasks.

2 Related Work

LLM agents are garnering significant interest in the modern era for their potential to perform complex tasks on behalf of humans. This section specifically reviews related research in three areas: slide generation, GUI-based LLM agents, and code generation.

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 7k slide-generation examples, to generate python code that invokes the SlidesLib API. 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 editing existing slides capabilities that significantly reduce the manual effort required from users.

2.2 LLM Agents for GUI Control

LLM-based agents that control graphical user interfaces (GUIs) (Gao et al., 2024) and (Koh et al., 2024) are also an active area of research. 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) relied on rule-based systems or domain-specific languages-approaches that often lacked scalability and adaptability, LLMs have enabled more flexible, generalizable solutions (Jiang et al., 2024; Yin et al., 2023).

Building on this progress, recent studies have introduced intermediate reasoning steps to further enhance code generation, such as guiding code generation through explicit natural language planning (Wang et al., 2025a), using intermediate plans to decompose and solve complex (Sun et al., 2024), multi-step coding tasks-thereby bridging the gap between high-level user intent and low-level executable code (Puerto et al., 2024; Yang et al., 2025). We utilize this code generation idea in our system to translate user instructions into slide editing operations.

3 Method

For clarity, we first categorize the capabilities required to edit slides based on 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, using language modality as illustrated in Figure 2. In the remainder of this section, we describe each components in detail.

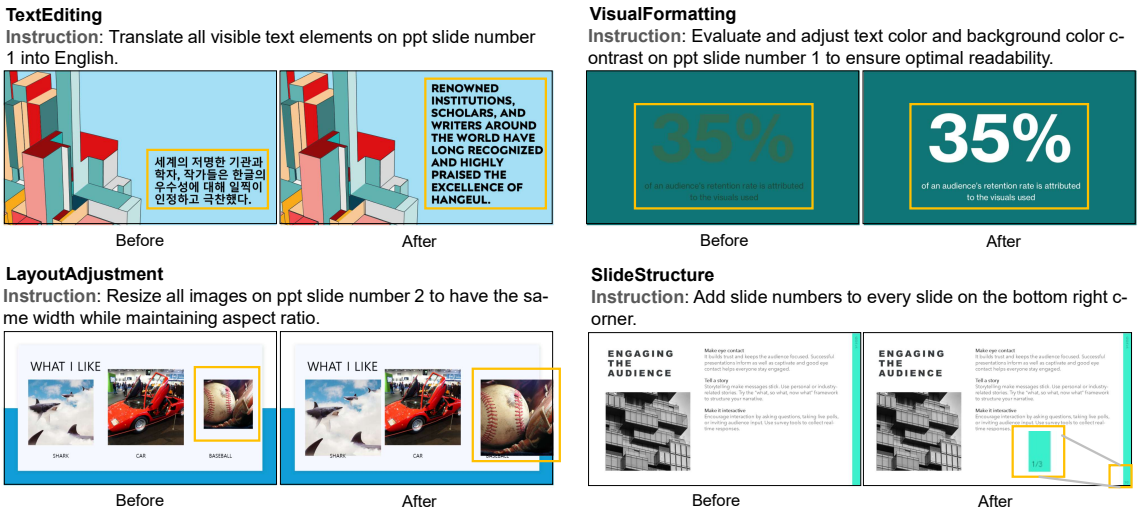

Figure 3: Results of TALK-TO-YOUR-SLIDES across four instruction categories. Modifications are highlighted with yellow boxes. *TextEditing*: Korean text has been translated into English according to the instruction. *VisualFormatting*: the original background and text colors were too similar, reducing readability; the revised version uses white text for improved contrast and clarity. *LayoutAdjustment*: the widths of the three images have been unified while preserving their aspect ratios, as instructed. *SlideStructure*: the page numbers have been added in response to the instruction.

3.1 High-level: Instruction understanding

For an LLM agent to edit slides, the first and most critical step is to understand the user's instructions. To this end, we follow the planning methodology described in Ruan et al. (2023). Our instruction understanding module operates at a high level within our system architecture, interpreting user instructions into structured, actionable plans that specify 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, the targeted element, and the corresponding action, allowing for precise and versatile slide editing tasks as shown in Figure 5.

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 as illustrated in Dong et al. (2024) and Guo et al. (2024a). The concrete prompts used for instruction understanding is included in Appendix N.1. Also an example of this module's output and the corresponding original slide can be found in Figure 5 and Figure 17, respectively.

3.2 Low-level: Document understanding

After understanding the user's command, a comprehensive understanding of the slide document is essential. Document understanding is a low-level component that accesses slides in our system (see Figure 2). It plays a crucial role in slide editing tasks, as the quality of the parsed content essentially defines the initial set of editable elements and thereby determines an upper bound on the final editing accuracy.

To enable fine-grained document understanding, we developed 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 6, 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 6 is shown in Figure 17.

In slide editing programs, a single text placeholder can contain text with diverse font formatting. Therefore, we found that we must parse the text at the *run-level* rather than at the placeholder level, where each *run* denotes a contiguous segment of text with consistent formatting. This level of details allow the system to more faithfully reflect how humans perceive and manipulate slides, thereby enabling precise, style-preserving edits.

We followed the finding that structured data formats enable LLMs to produce better outputs (He et al., 2024; Tan et al., 2025). Accordingly, parsed

System	Instruction	Performance metric (↑)							Efficiency metric (↓)		
		SR (%)	Inst. Foll.	Text	Image	Layout	Color	Exec. Time (s)	Avg. In (k)	Avg. Out (k)	Cost ($\times 10^{-3}$)
Model A: DeepSeek V3 (0324)											
Direct code gen.	TextEditing	77.05	0.00	0.00	1.01	1.02	1.02	16.30	1.00	0.39	0.4
	VisFmt	79.02	0.54	0.75	1.04	1.03	0.87	15.92	1.12	0.43	0.5
	LayoutAdj	70.10	0.08	1.47	1.08	1.41	1.41	15.20	1.02	0.51	0.5
	SlideStruct	85.02	0.18	1.80	1.12	1.93	1.66	17.10	0.89	0.62	0.6
	Overall	77.30	0.55	0.83	1.06	1.25	1.16	16.13	1.03	0.46	0.5
UI Agent	TextEditing	66.10	0.49	0.69	1.60	1.50	1.57	101.30	97.80	2.05	5.3
	VisFmt	88.75	2.46	1.91	1.68	1.81	1.50	122.40	90.10	2.38	5.8
	LayoutAdj	63.02	1.49	2.78	**2.21**	2.25	2.47	114.10	116.00	2.40	6.7
	SlideStruct	82.74	**1.56**	2.28	1.32	2.10	2.01	88.30	73.10	1.72	4.2
	Overall	75.65	1.70	1.97	1.55	2.18	2.04	114.03	94.80	2.14	5.5
Ours	TextEditing	**98.55**	**2.62**	**3.03**	**3.11**	**3.57**	**3.62**	**52.70**	**3.30**	**1.63**	**1.2**
	VisFmt	**96.10**	2.02	**2.05**	**1.73**	**2.27**	**1.90**	**82.30**	**4.18**	**2.11**	**1.6**
	LayoutAdj	**98.02**	1.41	2.39	1.83	**2.28**	**2.49**	**82.10**	**3.92**	**2.06**	**1.5**
	SlideStruct	**91.70**	1.41	**2.46**	**2.61**	**2.81**	**2.92**	**69.00**	**2.20**	**1.26**	**0.8**
	Overall	**96.84**	**2.12**	**2.48**	**2.27**	**2.73**	**2.69**	**71.55**	**3.55**	**1.87**	**1.4**
Model B: Gemini-2.5-flash											
Direct code gen.	TextEditing	62.07	0.00	0.10	1.70	1.70	1.70	23.08	1.21	0.93	0.7
	VisFmt	53.66	0.44	0.84	1.04	1.04	0.89	37.72	1.38	2.75	1.9
	LayoutAdj	58.95	0.66	1.55	0.79	1.42	1.40	23.25	1.26	0.94	0.8
	SlideStruct	73.33	0.50	1.61	1.25	1.58	1.72	28.17	1.00	1.05	0.8
	Overall	59.90	0.36	0.88	1.22	1.41	1.37	28.35	1.24	1.48	1.0
UI Agent	TextEditing	66.38	0.54	0.50	1.63	1.49	1.79	117.85	102.04	2.26	16.6
	VisFmt	86.18	**2.61**	2.01	1.72	2.25	1.80	122.25	93.27	2.15	15.2
	LayoutAdj	65.26	**2.07**	**2.82**	**2.38**	**2.51**	**2.87**	128.23	108.48	2.50	17.7
	SlideStruct	82.22	1.67	**2.31**	1.55	**2.38**	2.17	99.18	72.46	1.82	12.0
	Overall	74.41	1.64	1.81	1.83	2.11	2.10	119.66	97.29	2.23	15.9
Ours	TextEditing	**99.14**	**2.95**	**3.01**	**2.65**	**3.11**	**3.07**	**55.98**	**3.81**	**1.96**	**1.6**
	VisFmt	**94.30**	1.98	**2.22**	**1.86**	**2.38**	**2.16**	**94.78**	**5.09**	**3.58**	**2.8**
	LayoutAdj	**100.0**	1.80	2.39	2.15	2.37	2.56	**86.14**	**4.46**	**2.29**	**2.0**
	SlideStruct	**91.10**	**1.71**	1.95	**1.73**	2.17	**2.35**	**96.54**	**2.71**	**1.69**	**1.2**
	Overall	**96.83**	**2.21**	**2.48**	**2.17**	**2.58**	**2.57**	**78.95**	**4.26**	**2.53**	**2.0**

Table 1: Cross-model comparison results. 'SR' denotes execution success rate. **Ours** (highlighted in blue) achieves the best balance of high performance and efficiency. While 'Direct Code Gen' shows lower latency (underlined), its significantly lower SR makes it impractical. Cost is in USD ($\times 10^{-3}$).

outputs are converted into a structured JSON format. This representation facilitates downstream reasoning and editing, while ensuring compatibility with large language models. Our parsing logic is fully reproducible to support future research in structured document understanding. Additional details on document understanding are provided in Appendix A.

3.3 High-level: Document editing

With an understanding of both the user's instruction and the document's content, it is time to edit the slide document. The role of the document editing module is to modify the parsed content using an LLM, in accordance with the editing plan generated by the two preceding modules. For example, if the requirement specifies changing only the important 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 N.3.

3.4 Low-level: Code generator

The primary function of code generator is to generate python code that applies the necessary modifications to the slide editing program instance. It receives the raw parsed data before edited, the data after edited from document editing module, and the plan as illustrated in Figure 2. Once this module generates the code, it is then applied to the program. For the Windows version of PowerPoint, we adopt the COM protocol (Microsoft,

2021) with details provided in Appendix H.1. For macOS, AppleScript (Apple Inc.) can be used, as described in Appendix H.2. Additionally, an MCP server (Anthropic, 2025) is another alternative, which we discuss in Appendix H.3.

In addition, we follow the self-reflection mechanism described in (Shinn et al., 2023), to handle execution failures. We show the real world example on Appendix B. 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 N.3.

4 TSBench: Benchmark Dataset

Alongside proposing a novel system for editing presentation slides, we identify a notable scarcity of evaluation benchmarks for realistic, user-centric slide editing commands (Guo et al., 2023; Zhang et al., 2024b; Zheng et al., 2025). To address this gap and facilitate modular testing across a wide range of practical commands, we introduce TSBench, a new benchmark dataset designed to evaluate the slide editing capabilities of models or frameworks. In this section, we describe the construction process of our benchmark dataset in detail and present its key statistics.

4.1 Building the Benchmark

Users often perform edits on large presentations, with a single command sometimes impacting over 50 slides. However, rather than preparing such large-scale documents for evaluation, our benchmark is designed to deconstruct these complex scenarios into fundamental, modular tasks. This approach allows for a granular assessment of an agent's core capabilities without the overhead of managing massive slide decks.

To build this benchmark, we first created a dataset of user instructions and then developed the corresponding slide data to which these instructions could be applied. In this section, we describe the construction methodology for each component in detail.

Instructions data 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	Human Evaluation			
		Text	**Image**	**Layout**	**Color**
Direct code generation	TextEditing	1.55	2.20	2.25	2.10
	VisualFormatting	1.88	1.95	1.90	1.82
	LayoutAdjustment	2.11	1.76	2.05	2.01
	SlideStructure	2.05	1.91	2.15	2.24
	Overall	1.90	1.96	2.09	2.04
	PCC	0.84	0.69	0.73	0.88
UI Agent	TextEditing	2.45	2.88	2.79	2.91
	VisualFormatting	3.01	2.72	3.25	2.80
	LayoutAdjustment	3.85	3.41	3.55	3.76
	SlideStructure	3.31	2.55	3.38	3.17
	Overall	3.16	2.89	3.24	3.16
	PCC	0.87	0.69	0.71	0.8
Ours	TextEditing	4.15	3.95	4.22	4.18
	VisualFormatting	3.22	2.86	3.38	3.16
	LayoutAdjustment	3.39	3.15	3.37	3.56
	SlideStructure	3.95	2.73	3.17	3.35
	Overall	3.68	3.17	3.54	3.56
	PCC	0.89	0.75	0.72	0.94

Table 2: Human evaluation of slide editing performance using Gemini-2.5-flash with Pearson Correlation Coefficient (PCC) against an LLM judge. All p-values are below 10^{-3}.

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 analysis. Examples of instructions for each category are illustrated in Figure 8.

Slide deck data 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 slide decks for each of the 56 seed instructions. To ensure visual quality and realism, we adopted publicly available templates[1] and we listed 10 template which we used in benchmark dataset in Table 10.

We release the full benchmark, including metadata that maps each instruction to its corresponding PowerPoint file and instruction group. A summary of this mapping is presented in Table 8, and detailed statistics are provided in Appendix E.1.

TSBench-Hard While the core TSBench focuses on explicit and modular editing tasks, real-world user interaction is often underspecified or requires contextual reasoning. Since PowerPoint is an inherently visual medium, purely language-driven agents may struggle with tasks requiring spatial awareness or aesthetic inference. To systematically evaluate these operational boundaries, we introduce a challenging subset named *TSBench-Hard*. This subset comprises instructions in three high-difficulty categories: (1) *Visual-Dependent Tasks* that require spatial reasoning impossible to resolve with text matching alone (e.g., "Align the text box to the left edge of the image"); (2) *Ambiguous Instructions* where the agent must infer the user's aesthetic intent from vague directives (e.g., "Make the title slide look more professional"); and (3) *Impossible or Cross-Modal Tasks* that require perception inaccessible to a non-visual agent, evaluating its ability to correctly refuse execution rather than hallucinate (e.g., "Identify the speaker in the embedded video"). By incorporating these edge cases, TSBench-Hard serves as a stress test to determine the operational boundaries of slide editing agents. Illustrative examples are provided in Appendix E.2.

5 Experiment

In this section, we evaluate our proposed system, TALK-TO-YOUR-SLIDES, against two baseline methods using the benchmark dataset we introduced. Our evaluation measures both traditional performance and efficiency metrics, such as total cost. Each baseline is designed to address a specific research question.

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

Model	SR (%)	Instr. Follow	Overall Perf.	Exec. Time (s)	Avg. Cost ($)
Gemini-2.5-flash	96.83	**2.21**	**2.48**	78.95	0.0020
GPT-4.1-mini	96.57	2.13	2.46	78.37	0.0038
Claude 3.5 Haiku	95.89	2.08	2.41	75.33	0.0033
DeepSeek V3 (0324)	**96.84**	2.12	**2.48**	**71.55**	**0.0014**

Table 3: Cross-model comparison of Ours across Gemini-2.5-flash, GPT-4.1-mini, Claude 3.5 Haiku, and DeepSeek V3. Efficiency metrics are averaged across tasks. Cost in USD.

For RQ1, we evaluate a GUI-based agent baseline by including UFO2(Zhang et al., 2024a). This agent is capable of operating a wide range of Windows applications, including PowerPoint, and utilizes the Gemini-2.5-flash model. Additional configuration details are provided in Appendix G.

For RQ2, we establish a direct code generation baseline. This method leverages our document understanding module (Section 3.2) to extract a structured representation of each slide. It then prompts an LLM with this representation and the user instruction to generate executable code.

Configurations Our proposed framework utilizes two core large language models: the instruction understanding module is powered by Gemini-1.5-flash, while the document editing and code generation modules use Gemini-2.5-flash. The code generator is configured with up to three retry attempts. For comparison, we also conducted experiments by replacing this default Gemini setup with other models, specifically GPT-4.1-mini (OpenAI et al., 2024), Claude-haiku (Anthropic, 2024), and DeepSeek V3 (DeepSeek-AI et al., 2025). While the main results presented in this paper are based on our primary Gemini configuration, a detailed analysis of the results from these other LLMs is available in Appendix J. Full model descriptions and hyperparameter settings are provided in Appendix F.

Performance metrics To evaluate the performance of the slide editing LLM agent, we introduce three main metrics. First, we adopt the *Execution Success Rate* (SR) from prior work (Zhang et al., 2024b). SR indicates whether the finally generated code is successfully executed. Second, we use *LLM judge scores*, also employed in (Zheng et al., 2025; Wang et al., 2025b). These scores encompass text, image, layout, and color aspects. To ensure the reliability of the LLM judge on these four aspects, we also conducted a human

evaluation (Appendix I.1). In addition to these, we added *instruction following* metric, which indicates how well the edited presentation reflects the user's command. Our third metric is *Execution Time*, which refers to the latency in seconds required to execute a single instruction. More specific configurations are provided in Appendix I.

Efficiency metrics To assess system efficiency, particularly in the context of ***RQ1***, we include three metrics: *Average Input Tokens*, *Average Output Tokens*, and *Average Cost*. All averages are calculated for a single instruction. Token counts are computed by aggregating all tokens passed to the LLM within each module (including all four modules), and the cost is then estimated accordingly. Detailed information, including the API pricing for each LLM, is described in Appendix I.

6 Results

In this section, we will answer the research questions mentioned in the Section 1 based on the experimental results above. Then based on these findings, we discuss how agents should approach tasks that assist humans beyond slide editing when interacting with computing systems in general.

6.1 Main Results

This section answers two main research questions. First (RQ1), we investigate whether it is necessary to use an image modality from screen captures, as in conventional methods, to control a program. Second (RQ2), we explore if a level-wise architecture allows an LLM agent to operate an application more effectively. In response, we present the following research answers:

(RA1) Language can be used to control applications as an alternative to visual input. This means that an LLM agent does not require GUI screen captures as input; instead, it can operate using the internal language that functions within the program. As shown in Table 1, our system demonstrates superior performance on average when compared to the UI Agent system, both in Deepseek and Gemini. Although the UI Agent was more proficient on LayoutAdjustment instructions, which require significant visual confirmation, our system reduced the execution time to 66% of the UI Agent's. Moreover from an efficiency perspective, while our system generated more output tokens, it achieved an overwhelming advantage by using only 4% of the input tokens. Consequently, the average cost was lowered to just 13% of the UI Agent's.

(RA2) A level-wise architecture LLM agent can successfully operate an application. Our experiments show that an LLM agent that divides its workflow into high-level and low-level processes achieves a distinct performance improvement over direct code generation as shown in Table 1. The level-wise architecture, designed based on the nature of user-computer interaction explained in Section 3, scaled at test time. This led to a 160% increase in the success rate and a marked improvement in other performance metrics. This signifies that we successfully achieved test-time scaling (Muennighoff et al., 2025) by reconfiguring the LLM agent's architecture.

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 2. Four illustrative examples of slides edited by TALK-TO-YOUR-SLIDES are shown in Figure 3, and additional results conducted with other LLMs are in Appendix J.

6.2 Comparing the LLMs

We find that the choice of the underlying language model involves distinct trade-offs. For example, comparing Table 1 DeepSeek V3 (0324) and Gemini-2.5-flash, we see that in the Direct code generation baseline, Gemini-2.5-flash achieved a success rate of 59.90%, which was lower than DeepSeek's 77.30%. However, when leveraged within our system, Gemini-2.5-flash's performance improved to 96.83%, effectively matching GPT-4.1-mini's 96.84%. Although the detailed evaluations for Text, Image, Layout, and Color were similar between the two, our system achieved state-of-the-art results more frequently when using DeepSeek V3 (0324). From an efficiency perspective, although DeepSeek generated fewer tokens (1.87k) compared to Gemini-2.5-flash (2.53k), its higher unit pricing meant that it was relatively more expensive per token, despite the lower total cost ($1.4 vs $2.0).

Furthermore, as summarized in Table 3 (which shows results for our system exclusively), Gemini-2.5-flash used more input and output tokens than the other models. The higher input token count is likely due to token amplification as data passes

System	Visual-Dependent SR (%)	Ambiguous SR (%)	Multi-step SR (%)	Impossible RA (%)	Overall SR (%)
Direct code generation	4.1	11.6	8.9	22.4	9.2
UI Agent	**12.8**	18.2	10.4	35.0	14.8
Ours	12.5	**29.5**	**24.1**	**64.7**	**31.3**

Table 4: **TSBench-Hard results.** SR: execution success rate. RA: refusal accuracy on infeasible requests (higher is better). Best results are highlighted in bold.

through the four modules of our system. Notably, this resulted in a significantly higher instruction following score compared to the other models. Additionally, to demonstrate the practical robustness of our method in large-scale scenarios, we provide a full batch inference example processing an 89-page slide deck in Appendix L.

Meanwhile, could we create an even more efficient and capable LLM agent by supplementing the language modality with a vision modality? We discuss this topic in Appendix K.

6.3 TSBench-Hard Results

Table 4 reports execution success rates (SR) and refusal accuracy (RA) on TSBench-Hard. Given the high complexity of these tasks, existing baselines struggle significantly; the *Direct code generation* baseline achieves a near-zero success rate on visually dependent tasks, highlighting its inability to ground code in the visual state of slides.

Our system achieves the highest performance across most categories, with an overall SR of 31.3%, outperforming the strongest baseline (UI Agent) by +16.5 points. Notably, our system demonstrates superior capability in detecting infeasible requests, achieving a Refusal Accuracy of 64.7%, suggesting stronger semantic understanding of task boundaries.

However, the results also reveal the **operational boundaries** of our non-visual approach: on Visual-Dependent tasks, the UI Agent achieves a comparable SR (12.8% vs. our 12.5%), confirming that tasks requiring spatial reasoning remain challenging without visual perception. In contrast, our system substantially outperforms on Ambiguous (29.5% vs. 18.2%) and Multi-step (24.1% vs. 10.4%) tasks, where structured data access and hierarchical reasoning provide clear advantages. An ablation on self-reflection and detailed retry statistics are provided in Appendix B.

6.4 State Representation vs. Code Generation

A key distinction of our approach is that it operates on the document's *state representation*—the underlying object model (DOM)—rather than generating code from scratch to render visual output. Unlike code-driven image generation methods (e.g., TikZ-based approaches (Belouadi et al., 2023)), our system *edits* existing objects by modifying their properties (position, text, color) through API calls, ensuring that all elements remain fully editable native PowerPoint objects.

This design provides an advantage for batch processing: by operating on structured data, unmentioned elements are guaranteed to remain untouched, achieving the structural fidelity required for professional workflows. Re-generating slides via code rendering often leads to unintended layout shifts or hallucinated content. A more detailed comparison is provided in Appendix C.

6.5 Discussion: Toward Hybrid Agents

Our results paint a clear picture of complementary strengths. Language-driven structural agents excel at text-centric, batch, and multi-step editing with high efficiency and low cost. GUI-based agents, while slower and more expensive, remain competitive on spatially dependent tasks. We therefore position our system not as a total replacement for GUI agents, but as a *high-efficiency module for structured tasks* that could sit within a hybrid architecture. In such a system, structured manipulation handles the majority of editing operations, while visual perception is selectively invoked for layout verification and aesthetic quality assurance.

7 Conclusion

We introduced TALK-TO-YOUR-SLIDES, an LLM-powered agent for editing slides. By decomposing editing tasks into level-wise object manipulations, our system interacts with application in language modality to execute complex edits. We developed *TSBench*, a benchmark with 379 diverse editing instructions, including a *Hard* subset that systematically probes the operational boundaries of both visual and non-visual agents. Experiments show our approach significantly outperforms baselines on text-centric and batch editing tasks, demonstrating the effectiveness of a language-driven structural approach.

Limitations

By prioritizing code-based manipulation over visual processing, our agent achieves significantly higher speed, lower cost, and precise structural fidelity. However, this design choice introduces several concrete trade-offs.

Layout stability. When text length changes substantially—for instance, translating Chinese to English often doubles character count—the resulting text may overflow its bounding box. Detecting such overflow from structured data alone requires computing precise glyph extents and comparing them against container geometry, which is non-trivial without visual feedback (see the concrete example in Appendix M).

Visual aesthetics. Subjective instructions such as "Make this slide look balanced" or "Shrink the image to a proper size" require spatial perception that is inaccessible to a non-visual agent. As shown in TSBench-Hard (Table 4), our system's success rate on Visual-Dependent tasks (12.5%) is comparable to the UI Agent (12.8%), confirming that both paradigms struggle—but for different reasons.

These limitations reinforce our positioning of the system as a *high-efficiency module for structured tasks* rather than a complete replacement for GUI agents. We believe the most promising direction is a *Hybrid Agent* architecture (Section 6.5), where our efficient structural manipulation handles the majority of batch tasks, while VLM capabilities are selectively invoked for layout verification and visual quality assurance.

Ethics Statement

Our work introduces TALK-TO-YOUR-SLIDES, a system designed to enhance productivity and accessibility in slide editing, and TSBench, a benchmark for evaluating such systems. We hope our contributions will foster further research into language-driven software agents that reduce tedious manual labor.

We acknowledge potential sources of bias in our work. The *TSBench* dataset, while curated for quality, is built upon 56 seed instructions that were augmented using GPT-4o. Although the topics are diverse, covering areas from marketing to linguistics, the initial instructions and content may reflect the perspectives of their creators and may not represent all possible user domains. To mitigate this, all 379 instructions in the final benchmark underwent a manual human review process to filter for clarity, validity, and unambiguous objectives.

Furthermore, our agent relies on large language models (LLMs) such as Gemini and GPT, which may inherit societal biases from their training data. This could potentially influence how the agent interprets instructions or modifies content. In addition, our human evaluation was conducted with 21 participants noted as Appendix I.1. While this provided valuable qualitative feedback, the limited sample size may not fully capture the diverse range of user perspectives, and the findings should be interpreted with this in mind. We recommend that users, especially when working with sensitive topics, review the agent's output.

Since the system processes user slide data, which could be confidential, users should be mindful of the data privacy policies of the underlying LLM APIs. We encourage the responsible use of this technology as a tool to assist, not replace, human oversight.

Acknowledgements

This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (RS-2019-II190075, Artificial Intelligence Graduate School Program(KAIST)).

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2025-00555621)

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (RS-2025-02304967, AI Star Fellowship(KAIST))

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. 26

Anthropic. 2024. Introducing the next generation of claude. 7, 18, 22

Anthropic. 2025. Model Context Protocol (MCP). 6, 21

Apple Inc. AppleScript Language Guide. `https://developer.apple.com/library/archive/`

documentation/AppleScript/Conceptual/AppleScriptLangGuide/introduction/ASLR_intro.html. Accessed: 2025-09-30. 6

Jonas Belouadi, Anne Lauscher, and Steffen Eger. 2023. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. *arXiv preprint arXiv:2310.00367*. 9, 15

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. 3

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2025. Unleashing the potential of prompt engineering for large language models. *Patterns*, page 101260. 25

Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024. Towards tool use alignment of large language models. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1382–1400, Miami, Florida, USA. Association for Computational Linguistics. 2

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3290 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261. 17

William R Cook. 2007. Applescript. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 1–1. 20

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437. 7, 18

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 *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics. 4

Joshua P Ellis. 2017. Tikz-feynman: Feynman diagrams with tikz. *Computer Physics Communications*, 210:103–123. 15

Difei Gao and 1 others. 2024. Assistgui: Task-oriented pc graphical user interface automation. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*. 3

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. 2, 3, 21, 26

David N Gray, John Hotchkiss, Seth LaForge, Andrew Shalit, and Toby Weinberg. 1998. Modern languages and microsoft's component object model. *Communications of the ACM*, 41(5):55–65. 19

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 *Proc. the International Conference on Learning Representations (ICLR)*. 4

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. 6

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 *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1437–1454, Miami, Florida, USA. Association for Computational Linguistics. 2

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*. 26

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. 4

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 14281–14290. 1

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. 1

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. 3

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. *Preprint*, arXiv:2401.13649. 3

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. 25

Microsoft. 2021. The Component Object Model. `https://learn.microsoft.com/en-us/windows/win32/com/the-component-object-model`. Accessed: 2025-09-30. 5

Microsoft. 2024. Microsoft 365 copilot — microsoft 365. `https://www.microsoft.com/ko-kr/microsoft-365-copilot`. Accessed: 2024-05-17. 1

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *Preprint*, arXiv:2501.19393. 8

Matt Neuburg. 2006. *AppleScript: The Definitive Guide*. " O'Reilly Media, Inc.". 20

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*. 26

OpenAI. 2024. Gpt-4o api overview. `https://platform.openai.com/docs/models/gpt-4o`. Accessed: 2025-05-06. 6, 7

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774. 7, 17, 18, 22

Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in Text+Code LLMs. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 11234–11258, Miami, Florida, USA. Association for Computational Linguistics. 3

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Large language model-based ai agents for task planning and tool usage. *Preprint*, arXiv:2308.03427. 4

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*, DocEng '21, New York, NY, USA. Association for Computing Machinery. 1, 3

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*. 6

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. 3

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. 4

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 *Proc. the International Conference on Learning Representations (ICLR)*. 3

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. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):1955–1977. 7

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. 1

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. 3

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*. 1

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 *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 126–173, Toronto, Canada. Association for Computational Linguistics. 3

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. 3

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024a. Ufo: A ui-focused agent for windows os interaction. *Preprint*, arXiv:2402.07939. 1, 2, 3, 7, 19

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. 6, 7

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. 1, 2, 3, 6, 7, 14

Appendix

We provide detailed supplementary materials organized as follows:

Appendix A and **Appendix B** detail the system implementation, including document parsing and self-reflection mechanisms.

Appendix C compares our approach with code-driven image generation methods.

Appendix D provides example of output in understanding modules.

Appendix E provides comprehensive statistics and details of the TSBench dataset, including Hard subset examples and evaluation details.

Appendix F and **Appendix G** describe the experimental models and the UFO baseline.

Appendix H details our primary implementation on Windows via COM and discusses AppleScript as a macOS alternative.

Appendix I and **Appendix J** present evaluation details (including human eval) and auxiliary experimental results.

Appendix K and **Appendix L** discuss the role of GUI images and present qualitative results on batch slide inference.

Appendix M discusses future directions, specifically addressing the challenges of fine-grained layout adjustment and the necessity of visual feedback.

Appendix N lists the full prompts used for planning, editing, coding, and evaluation.

A 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.

B Detailed Analysis of Self-Reflection Mechanism

In this section, we provide a concrete example to elaborate on the self-reflection mechanism described in Section 3.4. The self-reflection module is triggered when the execution layer encounters a runtime error (e.g., `com_error` in Windows or API failures). Upon detecting an error, the agent analyzes the full error trace, reflects on the syntax or logical flaws, and regenerates the code. This iterative process continues until the code executes successfully or reaches a pre-defined maximum number of attempts (`max_attempt_num`).

We present a real-world scenario in Figure 4 where the agent resolves a data type mismatch during a PowerPoint automation task.

B.1 Impact of Self-Reflection

We further investigate the contribution of the self-reflection mechanism in Table 6. The ablation study reveals that self-reflection is critical for handling complex instructions.

Performance Gain: Enabling self-reflection yields a consistent improvement across all subcategories, boosting the overall SR from 23.5% to 31.3% (+7.8%). The gain is particularly pronounced in Multi-step tasks (+8.8%), where

iterative refinement helps correct intermediate logic errors.

Error Mitigation: More importantly, self-reflection drastically reduces the Catastrophic Failure (CF) rate from 24.8% to 8.7%. Without reflection, the agent often generates code that corrupts slide elements or throws unhandled exceptions. The reflection loop acts as a safety guard, catching these errors before they finalize.

Case Study: Self-Correction Loop

User Instruction: "Change the title color to red."

1. Attempt 1 (Generated Code):

```
# Agent incorrectly tries to assign a tuple
slide.Shapes(1)...Font.Color.RGB = (255, 0, 0)
```

Execution Error:
TypeError: Objects of type 'tuple' can not be converted to a COM VARIANT

2. Refinement Step (Self-Reflection):
"I attempted to assign a tuple (255, 0, 0) directly. However, the PowerPoint COM interface expects a single integer... I must convert it."

3. Attempt 2 (Regenerated Code):

```
# Agent corrects the format to an integer
slide.Shapes(1)...Font.Color.RGB = 255
```

Result: **Success** (Color updated to red)

Figure 4: A real-world example of the self-reflection mechanism handling a data type error during execution.

Variant	Visual-Dependent SR (%)	Ambiguous SR (%)	Multi-step SR (%)	Overall SR (%)	CF (%)
Ours (w/o self-reflection)	20.4	21.1	15.3	23.5	24.8
Ours (w/ self-reflection)	**26.8**	**29.5**	**24.1**	**31.3**	**8.7**

Table 5: **Ablation on self-reflection.** CF: catastrophic failure rate (e.g., unhandled exception, corrupted slide state, or repeated invalid actions). Define CF consistently with your pipeline logs.

System	Avg. #Attempts	P90 #Attempts	Refine Triggered (%)	Latency Overhead (%)
Ours (w/o self-reflection)	1.00	1	0.0	–
Ours (w/ self-reflection)	1.42	3	45.6	+38.2

Table 6: **Self-reflection efficiency.** Attempt counts and overhead on TSBench-Hard. "Refine Triggered" is the fraction of instances that entered the self-correction loop at least once.

TSBench-Hard. Table 4 reports performance on visually dependent, ambiguous, multi-step, and impossible instructions. Our agent improves overall SR by **+16.5** points over the strongest baseline, while achieving the best refusal accuracy (RA) on infeasible requests. As shown in Table 5, self-reflection yields consistent gains (overall **+7.8** SR) and reduces catastrophic failures, at a modest runtime overhead (Table 6).

C Comparison with Code-Driven Image Generation

The Area Chair and reviewers noted similarities between our code-generation approach and existing text-to-image generation works, such as TikZ-based methods (e.g., Automatikz(Belouadi et al., 2023)). While both paradigms utilize LLMs to generate code for visual outputs, there are fundamental differences in their objectives, operational mechanics, and practical applications.

Object Manipulation vs. Pixel Rendering. The primary goal of TikZ-based approaches is *generation*—rendering a static image or vector graphic from scratch based on a description (Belouadi et al., 2023; Ellis, 2017). In contrast, our work targets *editing* and *manipulation* of existing documents. We do not render pixels; instead, we modify the properties (e.g., position, text, color) of pre-existing objects within the slide's Document Object Model (DOM). This distinction is critical: generating a slide as a single image (or TikZ compilation) results in a non-editable, *dead* output where individual elements cannot be easily modified by the user afterwards. Our API-based approach ensures that all elements remain fully "alive" and editable native PowerPoint objects.

Structural Fidelity in Batch Processing. As emphasized in our positioning, our key contribution lies in *efficient text-centric batch processing*. Image generation models (visual or code-based) often struggle to maintain strict structural fidelity across multiple slides (e.g., "Change the font of all titles to Arial while keeping their exact positions"). Re-generating the entire slide via TikZ or VLM often leads to unintended layout shifts or hallucinations. By strictly operating on the underlying structured data (XML/COM), our method guarantees that unmentioned elements remain untouched, achieving the high fidelity and low latency required for professional, large-scale workflows that image generation methods cannot support.

Operational Efficiency. Rendering images or compiling complex TikZ code is computationally

```
{ "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 5: Example output generated by the instruction understanding module.


```
{ "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 6: 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.

expensive and slow. Our approach, which executes lightweight API calls to modify attributes, bypasses the rendering pipeline entirely. This efficiency is what enables the massive cost and time reductions (as shown in Table 1) compared to visual approaches, making it the only viable solution for real-time, bulk editing tasks.

D Example of Understanding Output

In this section, we show the example output generated by instruction understanding module (Figure 5 and document understanding module (Figure 6.

E Detail of TSBench

In this section, we present detailed information about TSBench. First, we have listed the topics of slide content in Table 7.

Additionally, the mapping between instruction numbers, slide IDs, and the four categories is listed in Table 8. 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 10 along with their corresponding links.

Examples of TSBench slides and instructions are presented in Figure 9.

E.1 Statistics

Table 9 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 Table 8, and slide content topics are listed in Table 7.

E.2 Detail of TSBench-Hard

The three categories of TSBench-Hard (Visual-Dependent, Ambiguous, and Impossible/Cross-Modal) are described in the main text (Section 4). Here we provide illustrative examples in Figure 7.

In addition, the original slide is presented in Figure 17 from which the example parsed data Figure 6 was extracted.

E.3 TSBench-Hard Evaluation Details

The main TSBench-Hard results are reported in Section 6.3. For this evaluation, the image of the edited slide was fed into `gemini-3-flash`. The model rated the extent to which the result satis-

Table 7: 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		

fied the dataset's `ideal_description` on a scale of 0 to 5, which was subsequently converted into a percentage.

F 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 11.

Gemini-1.5-flash (gemini-1.5-flash) Gemini 1.5 Flash (Comanici et al., 2025) 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 (Comanici et al., 2025) 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 (OpenAI et al., 2024) 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.

Table 8: 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		

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 9: 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.

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

GPT-4o (gpt-4o-2024-08-06). GPT-4o ("o" for "omni") (OpenAI et al., 2024) 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.

Claude 3 Haiku (claude-3-haiku-20240307). Claude 3 Haiku (Anthropic, 2024) is the fastest and most economical model in Anthropic's Claude 3 family. It is designed for near-instantaneous responsiveness, making it ideal for applications requiring real-time interaction, such as customer support chatbots and content moderation. Like the other models in the Claude 3 series (Opus and Sonnet), Haiku possesses robust vision capabilities and supports a 200K token context window. The model is positioned as a cost-effective solution for enterprise workloads where speed and accuracy are critical, offering strong performance relative to other models in its class.

DeepSeek-V3-0324 DeepSeek-V3 (DeepSeek-AI et al., 2025) is a powerful Mixture-of-Experts (MoE) language model comprising 671 billion total parameters, with 37 billion activated for each token. It leverages the Multi-head Latent Attention (MLA) and DeepSeekMoE architectures, previously validated in DeepSeek-V2, to ensure efficient inference and cost-effective training. Innovatively, DeepSeek-V3 introduces an auxiliary-loss-free strategy for load balancing and a multi-token prediction training objective to enhance per-

Type 1: Visual-Dependent Task

Instruction: "Move the title text box on slide 5 so that its bottom edge touches the top of the bar chart."

Why Hard: Requires computing relative coordinates (bounding boxes) not explicitly given in text. The agent must calculate $y_{title\_bottom} = y_{chart\_top}$.

Ideal Outcome: Zero vertical spacing between the title and chart. The title retains its horizontal alignment (centered) but anchors perfectly to the chart's top edge.

Type 2: Ambiguous & Complex Logic

Instruction: "For every slide with a flowchart, check if the arrows are connected; if not, connect them."

Why Hard: Requires identifying composite structures ("flowcharts") across multiple slides and conditionally modifying shape connectors only where gaps exist.

Ideal Outcome: Arrows snap to connection points (green dots) with zero gap. Simple lines are converted to dynamic connectors that move with the shapes.

Type 3: Impossible / Cross-Modal Task

Instruction: "Identify who is speaking in the background audio of the video on slide 4."

Why Hard: Requires processing non-textual data (audio streams inside a video container) which is inaccessible via standard Slide APIs.

Ideal Behavior (Refusal): Instead of hallucinating a speaker's name or producing a runtime error, the agent must correctly identify the cross-modal dependency and **refuse the instruction** (e.g., *"I cannot access audio content within video files."*).

Figure 7: Examples of challenging instructions included in **TSBench-Hard**. These cases evaluate the agent's capability in spatial alignment (Type 1), multi-step logical reasoning (Type 2), and handling cross-modal requests that require external perception (Type 3).

formance. The model was pre-trained on a high-quality dataset of 14.8 trillion tokens, followed by Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL). Evaluations show that DeepSeek-V3 surpasses other open-source models and rivals the performance of leading closed-source models, while requiring a notably efficient 2.788 million H800 GPU hours for its complete and stable training process.

G 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-

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 8: Examples of instructions across four categories.

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 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).

H Alternative Platforms and Implementation Details

In this section, we describe the technical details of our implementation. We use the Component Object Model (COM) for our Talk-to-your-slides system. While COM has the limitation that it cannot be applied to macOS, we also describe an alternative method that can be implemented on this platform: AppleScript. Finally, we introduce a concurrent work for creating slides using the Model Context Protocol (MCP).

H.1 Component Object Model (COM) for Windows

The Component Object Model (COM) is a platform-independent, distributed, object-oriented system developed by Microsoft that allows software components to interact (Gray et al., 1998). It

Table 11: 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	–
Claude Haiku 3[e]	$0.25	–	$1.25	–
Deepseek-V3[f]	$0.27	$0.07	$1.10	–

[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

[e] https://claude.com/pricing#api

[f] https://api-docs.deepseek.com/news/news1226#-api-pricing-update

enables objects to be implemented in a language-agnostic way and facilitates communication between objects in different processes.

Essentially, it can be seen as a protocol enabling inter-program communication within the Windows environment. For instance, the PowerPoint application can transfer a class object, which encapsulates all the information of a shape placeholder, to a Python script. The script can then accept this object, modify its properties, and subsequently return it to PowerPoint.

In the system described in this paper, COM plays a crucial role in interacting with PowerPoint in a Windows environment. The PowerPoint application acts as a COM server, exposing its functionalities and data as COM objects (e.g., presentations, slides, shapes, text boxes). The paper's Code generator module (Section 3.4) generates Python code that acts as a COM client, typically using a library like 'win32com'. By directly accessing the application's internal object model instead of going through the GUI, this method enables fast, accurate, and cost-effective edits.

H.2 AppleScript for macOS

AppleScript (Neuburg, 2006; Cook, 2007) is a scripting language developed by Apple for macOS, designed to help users automate repetitive tasks and create complex workflows by controlling multiple applications. Scriptable Mac applications expose an object model (e.g., 'slide', 'shape' in Keynote) and a command dictionary that AppleScript can understand.

If the system from the paper were to be implemented in a macOS environment, AppleScript could replace the role of COM. The Code generator module (Section 3.4 would be modified to produce AppleScript code instead of Python-COM code. For instance, to execute the instruction "Add page numbers to the bottom of every slide," the system would generate and execute an AppleScript that iterates through each slide and adds a page number text box. This approach, similar to the COM-based method on Windows, achieves automation by directly manipulating program objects without GUI interaction.

Model	COM	AppleScript
Gemini-2.5-flash	96.83	76.1
GPT-4.1-mini	96.57	69.3
Claude Haiku 3	95.89	78.2
DeepSeek V3 (0324)	96.84	78.1

Table 12: A comparison of the execution success rates between the COM-based Python code and the AppleScript-based code.

Nevertheless, AppleScript had a lower success ratio than COM, which varied depending on the capabilities of the commercial LLMs as shown in Table. As shown in Table 12, the optimized and recommended configuration for our system is COM for Windows. This is based on the assumption that the models were likely trained on more COM-related data than AppleScript data.

H.3 MCP (Office-PowerPoint-MCP-Server)

The Office-PowerPoint-MCP-Server [2] is a server that provides an API to control PowerPoint. This project operates by running a server on a local machine that listens for HTTP requests and executes the corresponding commands within PowerPoint. This provides a flexible, platform-agnostic method

[2] https://github.com/GongRzhe/Office-PowerPoint-MCP-Server

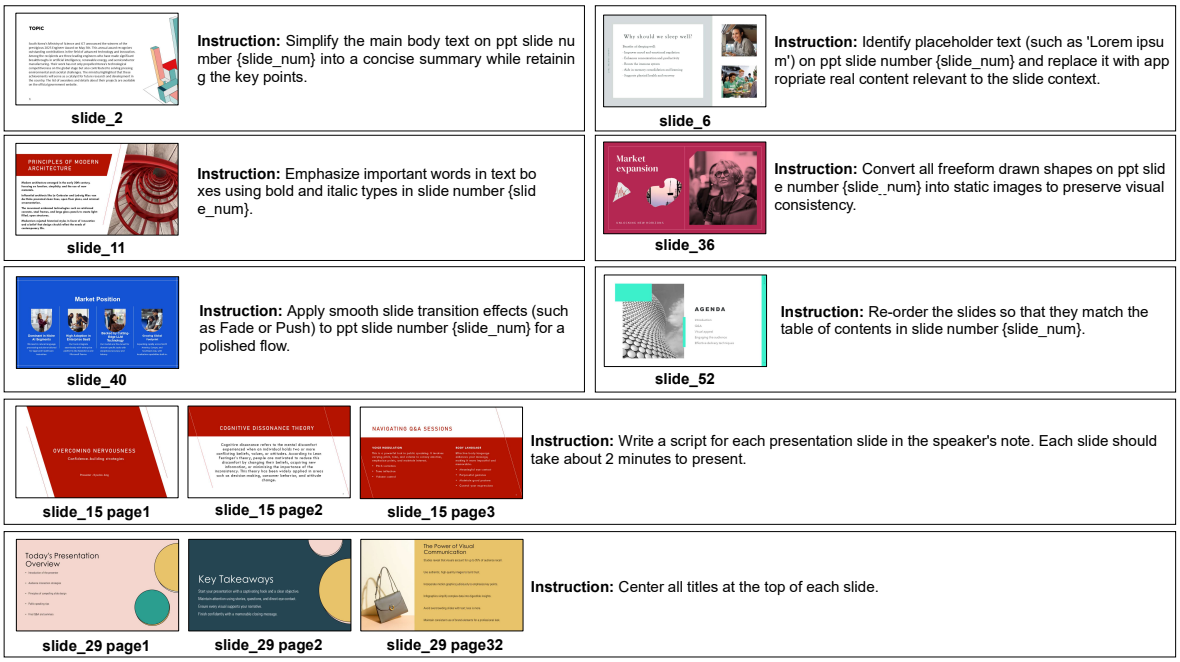

Figure 9: Example from the TSBench dataset. Some data points consist of a single slide, while others contain multiple slides.

for controlling PowerPoint from any programming environment that can make HTTP requests, without being tied to specific technologies like COM or AppleScript.

To apply Model Context Protocol (MCP) (Anthropic, 2025) to the paper's framework, the Code generator module (Section 3.4 would generate code that sends HTTP requests to specific API endpoints. For example, modifying the text of a particular slide could be implemented by sending a 'POST' or 'PUT' request to an endpoint like '/slide/slide_number/shape/shape_name' on the MCP server. The body of the request would contain the text content and formatting information in a JSON format. This approach offers an alternative for the low-level control component, refactoring it into a microservice architecture that enhances scalability and language independence.

I Details of evaluation

Instruction following measures how effectively the 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 18 and 19.

I.1 Human evaluation

We conducted a human evaluation with 21 volunteers (17 male, 4 female) from Chung-Ang University, all with extensive experience using PowerPoint slides in both academic and external settings. Participants evaluated a subset of 30 items from our benchmark. Similar to the LLM judge, the volunteers were asked to rate the results on a scale from 0 (worst) to 5 (best) based on four criteria: text, image, layout, and color. The correlation coefficient presented in our analysis is calculated based on these 30 data points. None of the participants were color-blind. The ethical considerations for this study are detailed in Section 7.

Table 1 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.[3]

J Auxiliary results

In this section, we report the results of our experiments conducted using gpt-4.1-mini (OpenAI et al., 2024), Claude-haiku (Anthropic, 2024). The results are presented in Table 13 and Table 15, the correlation coefficients for the metrics assessed by the LLM judge are reported in Table 14.

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 2.

K 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.

[3]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).

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.

L Examples of Batch Slide Inference Results

This section presents examples of actual lecture materials (originally in Korean) translated into English using Talk-to-your-slides. We have processed large-scale PowerPoint decks, specifically *07_RNN_LSTM_Seq2Seq* (89 pages) and *08_Transformer* (49 pages), and made the results publicly available at the following link: `https://drive.google.com/drive/folders/1-TmujhQkMg17QsdFx_LOCVRr19QNFyOd?usp=sharing`.

Figure 10 illustrates the original PPT slides, while Figure 11 shows the processed versions. The translation into English was executed successfully. Notably, as shown in the slide at row 2, column 1, terms with the same meaning were successfully edited at the 'run' level while maintaining consistent coloring.

However, some limitations remain for future work. First, mathematical formulas are occasionally altered by the LLM into illegible formats. Second, in cases involving complex illustrations or diagrams, the transition from Korean to English can lead to text box overflows due to differences in text length and structure.

M Future Work

Research in slide editing, such as the methods discussed in Section 2.1 and our proposed TALK-TO-YOUR-SLIDES, is still in its early stages. This raises a critical question from a practical user perspective: ***what is the most significant area for improvement?*** We propose that the primary challenge lies in handling fine-grained layout adjustments.

As illustrated in Figure 12, while the title, shape, sub-caption, and text are correctly inserted, the layout of the text box is improper. A human user editing this slide would naturally resize

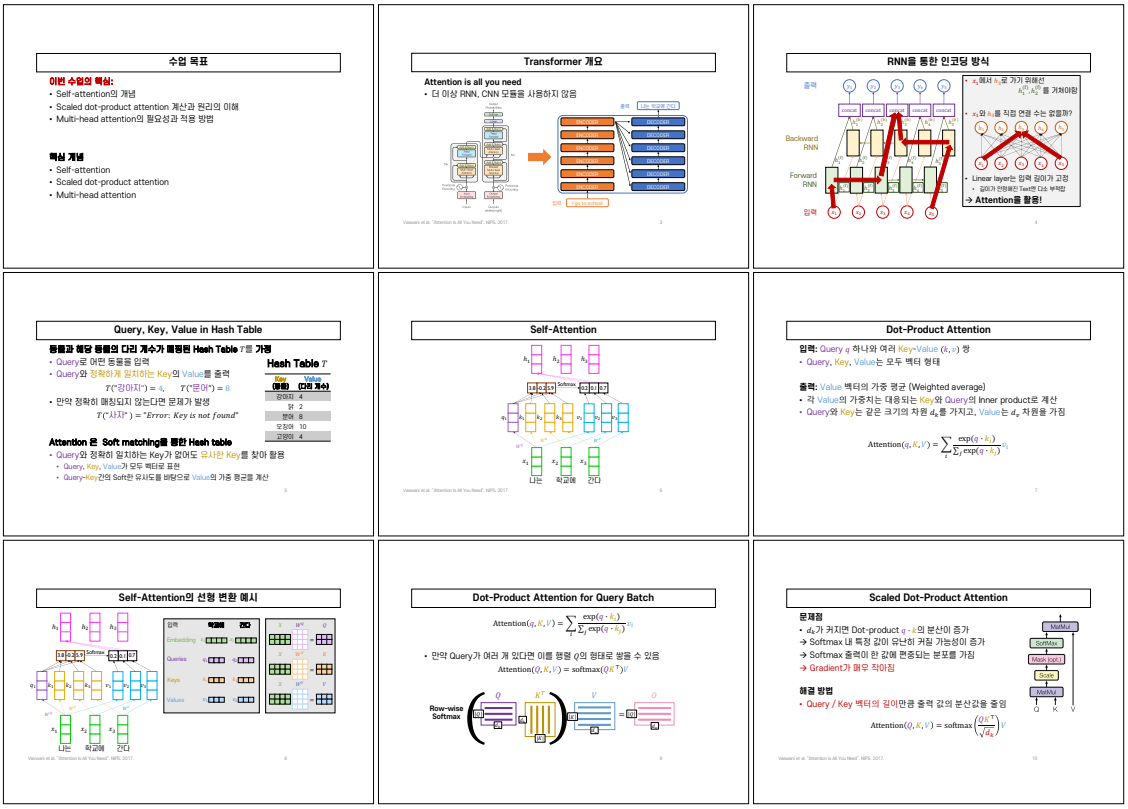

Figure 10: Example of the original lecture slide before processing. This is a part of the batch editing process (editing 89 slides). You can find the file in here .

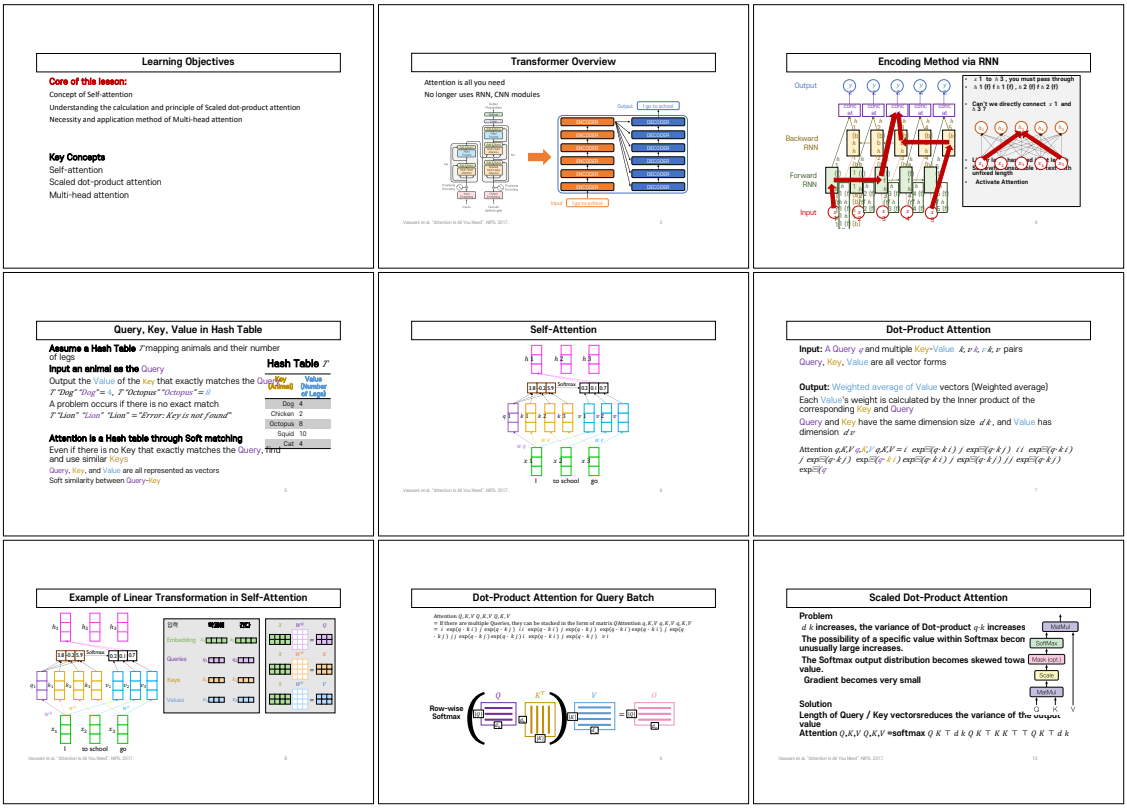

Figure 11: Example of the original lecture slide after processing. This is a part of the batch editing process (editing 89 slides). You can find the file in here .

System	Instruction	Performance metric (↑)							Efficiency metric (↓)		
		SR (%)	Inst. Foll.	Text	Image	Layout	Color	Exec. Time (s)	Avg. In (k)	Avg. Out (k)	Cost ($\times 10^{-3}$)
Direct code gen.	TextEditing	76.72	0.00	0.00	1.04	1.04	1.04	18.55	1.03	0.41	1.0
	VisFmt	78.05	0.53	0.73	1.06	1.02	0.86	18.08	1.16	0.45	1.1
	LayoutAdj	69.47	0.07	1.45	1.09	1.40	1.40	17.06	1.04	0.53	1.2
	SlideStruct	84.44	0.17	1.78	1.11	1.91	1.64	19.57	0.91	0.65	1.4
	Overall	76.25	0.53	0.81	1.07	1.23	1.15	18.19	1.06	0.48	1.2
UI Agent	TextEditing	65.48	0.48	0.67	1.58	1.48	1.56	110.84	101.48	2.10	15.9
	VisFmt	88.09	**2.44**	1.89	1.67	1.79	1.48	134.57	94.12	2.44	14.9
	LayoutAdj	62.64	**1.48**	**2.76**	**2.20**	2.23	2.45	127.13	121.04	2.47	18.8
	SlideStruct	82.30	**1.55**	2.27	1.30	2.08	1.99	95.07	75.45	1.77	11.9
	Overall	73.84	1.68	1.94	1.55	2.16	2.04	121.08	98.22	2.30	15.4
Ours	TextEditing	**98.28**	**2.60**	**3.01**	**3.09**	**3.54**	**3.60**	**58.17**	**3.35**	**1.66**	**3.3**
	VisFmt	**95.93**	2.00	**2.02**	**1.71**	**2.25**	**1.88**	**89.99**	**4.24**	**2.15**	**4.5**
	LayoutAdj	**97.89**	1.40	2.37	1.81	**2.26**	**2.48**	**89.86**	**3.99**	**2.10**	**4.3**
	SlideStruct	**91.11**	1.40	**2.44**	**2.59**	**2.78**	**2.90**	**74.39**	**2.24**	**1.28**	**2.1**
	Overall	**96.57**	**2.13**	**2.46**	**2.26**	**2.71**	**2.68**	**78.37**	**3.60**	**1.89**	**3.8**

Table 13: Experimental results using the **GPT-4.1-mini** model. **Ours** (highlighted in blue) demonstrates superior performance and efficiency compared to the UI Agent (highlighted in red). While 'Direct Code Gen' shows lower latency (underlined), its execution success rate is insufficient for practical use.

System	Instruction	Human Evaluation			
		Text	Image	Layout	Color
Direct code generation	TextEditing	1.44	2.17	2.18	2.16
	VisualFormatting	1.82	2.13	1.87	2.08
	LayoutAdjustment	2.10	1.54	2.55	1.83
	SlideStructure	1.98	1.92	2.33	2.23
	Overall	1.83	1.94	2.23	2.08
	PCC	0.82	0.68	0.69	0.87
UI Agent	TextEditing	2.43	2.77	2.84	2.90
	VisualFormatting	3.04	2.69	3.17	2.72
	LayoutAdjustment	3.56	3.37	3.62	3.44
	SlideStructure	3.25	2.62	3.17	3.47
	Overall	3.07	2.86	3.20	3.13
	PCC	0.86	0.77	0.69	0.82
Ours	TextEditing	3.86	3.85	4.16	4.16
	VisualFormatting	3.21	2.82	3.28	3.82
	LayoutAdjustment	3.28	3.06	3.17	3.48
	SlideStructure	3.88	2.66	3.01	3.45
	Overall	3.56	3.10	3.41	3.73
	PCC	0.78	0.75	0.69	0.88

Table 14: Human evaluation of slide editing performance using GPT-4.1-mini with Pearson Correlation Coefficient (PCC) against an LLM judge. All p-values are below 10^{-3}.

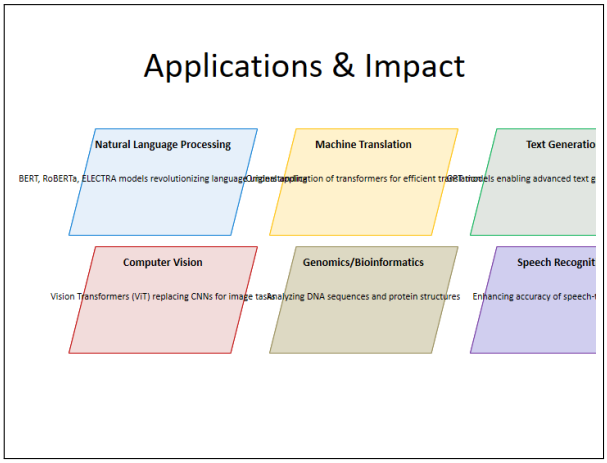

Figure 12: An example of a layout error in the slide editing results. (see Appendix H.3)

the text box to fit the text neatly within the parallelogram. However, determining this text overflow using only PowerPoint's XML data is a non-trivial task. It would require calculating the precise boundaries of the parallelogram and comparing them with the coordinates of the text box to judge whether an overflow has occurred. This process is computationally complex.

For such spatial reasoning tasks, it would be far more effective for the agent to perceive the slide visually, for instance, through a screenshot. As we will further discuss in Appendix K, we believe the ideal slide editing agent, the ultimate goal for our research community, should be a language-based model that leverages vision as an auxiliary input to determine its actions.

N 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

System	Instruction	Performance metric (↑)							Efficiency metric (↓)		
		SR (%)	Inst. Foll.	Text	Image	Layout	Color	Exec. Time (s)	Avg. In (k)	Avg. Out (k)	Cost ($\times 10^{-3}$)
Direct code gen.	TextEditing	75.91	0.00	0.00	1.02	1.00	1.03	17.92	1.02	0.40	0.9
	VisFmt	77.12	0.51	0.71	1.05	1.01	0.84	17.48	1.15	0.44	1.0
	LayoutAdj	68.80	0.06	1.41	1.07	1.37	1.36	16.58	1.03	0.52	1.1
	SlideStruct	83.72	0.16	1.73	1.09	1.86	1.60	18.94	0.90	0.63	1.2
	Overall	75.39	0.51	0.79	1.06	1.21	1.13	17.73	1.05	0.47	1.1
UI Agent	TextEditing	64.62	0.47	0.65	1.55	1.45	1.53	106.10	96.30	2.03	12.7
	VisFmt	87.21	**2.39**	1.85	1.64	1.75	1.45	129.42	89.40	2.35	12.0
	LayoutAdj	61.90	**1.43**	**2.69**	**2.14**	2.18	2.38	121.56	114.80	2.38	15.1
	SlideStruct	81.55	**1.50**	2.20	1.27	2.03	1.93	92.01	71.90	1.70	9.9
	Overall	73.07	1.64	1.90	1.52	2.10	1.99	117.27	93.10	2.12	12.4
Ours	TextEditing	**97.71**	**2.55**	**2.95**	**3.04**	**3.49**	**3.54**	**56.10**	**3.28**	**1.62**	**2.9**
	VisFmt	**95.21**	1.95	**1.98**	**1.68**	**2.21**	**1.85**	**86.40**	**4.16**	**2.09**	**3.9**
	LayoutAdj	**97.23**	1.36	2.32	1.78	**2.21**	**2.42**	**86.70**	**3.90**	**2.04**	**3.8**
	SlideStruct	**90.42**	1.37	**2.39**	**2.54**	**2.73**	**2.84**	**72.10**	**2.19**	**1.25**	**1.9**
	Overall	**95.89**	**2.08**	**2.41**	**2.23**	**2.66**	**2.64**	**75.33**	**3.52**	**1.85**	**3.3**

Table 15: System-wise scores using the **Claude 3.5 Haiku** model. **Ours** (highlighted in blue) demonstrates superior performance and efficiency compared to the UI Agent (highlighted in red). Cost is in USD ($\times 10^{-3}$).

in prompts, we disclose our prompts in full to ensure specificity and reproducibility.

N.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 13.

N.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 14.

N.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 15.

N.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 16.

N.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 N.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 19.

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 13: A prompt used in instruction understanding.

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 14: 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 15: A prompt used in Code Generator.

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 16: A prompt used in baseline system.

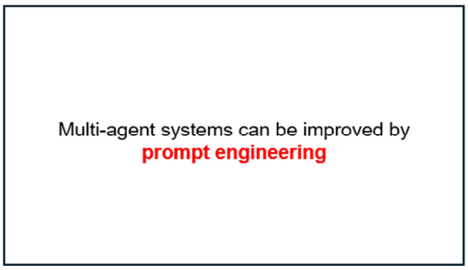

Figure 17: The original slide from which the example parsed data was extracted.

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 18: 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 19: A prompt used in LLM judge which evaluate text, image, layout, color.