SlideTuner: PowerPoint Slide Design via XML Representation Learning and Preference Optimization

1st Lixiang Li
Department of Computer Science
Purdue University
West Lafayette, USA
li4256@purdue.edu

2nd Anjan Goswami
Microsoft Co.
Mountain View, USA
anjangoswami@microsoft.com

3rd Md Muksitul Haque
Microsoft Co.
Boston, USA
mdmhaque@microsoft.com

4th Bharat Bhargava
Department of Computer Science
Purdue University
West Lafayette, USA
bbshail@purdue.edu

***Abstract*—Generating high-quality PowerPoint slides from natural language instructions is a complex task that demands not only deep semantic understanding, but also aesthetic design. The essential component for building a functional and visually rich slide is the XML object. Therefore, it is intuitive that the most direct path to creating a high-quality slide is to generate it directly from the foundational XML structure. However, previous "human instruction to slide generation" models typically rely on generating Python code, which serves as an intermediary to produce the final slide output rather than direct production of the XML object. As a result, these models lack the ability to precisely construct and control the building blocks required for a detailed slide composition.**

We introduce SlideTuner, a custom finetuned GPT-4o model specifically engineered to generate high-quality PowerPoint slides by generating the required XML files. Through extensive empirical experiments, we demonstrate that the fine-tuned GPT-4o model successfully and consistently produces visually coherent and aesthetically pleasing slides. The SlideTuner employs a two-stage training approach: first we apply SFT to the language model, enabling it to generate slide-rendering XML code directly from user instruction, utilizing XML data extracted from native PowerPoint slides. Second, we apply Direct Preference Optimization (DPO) to align the model's outputs with preferred visual styles, such as specific font choices. The slides produced by our model exhibit superior layout scores and style adherence. While this work focuses on font-level aesthetic control, our work establishes a foundation for future research aimed at precisely guiding slide generation toward diverse visual or structural preferences.

I. Introduction

Designing PowerPoint slides from natural language instructions is a challenging task that requires semantic understanding of input natural language and visual design of a PowerPoint slide [1] [2].

While large language models have shown strong capabilities in tasks like code generation, document synthesis, and interface design, their ability to generate aesthetically refined, user-aligned visual content remains limited. Creating a high-quality slide requires not only correctly interpreting the user's intent, but also arranging text and media with clarity, balance, and design sensibility.

Recent systems like AutoPresent [3] have made progress by training models to translate user instructions into Python codes via supervised fine-tuning. While effective in producing syntactically valid code, these approaches relies on Python code as representation for PowerPoint generation and lack feedback-driven mechanisms for improving layout quality. As a result, the generated slides often exhibit crowded elements, poor alignment, or visually jarring designs, especially when user instructions are vague or underspecified.

In this work, we introduce SlideTuner, a finetuned model that generate XML code for PowerPoint slide generation. SlideTuner first trains a language model to generate executable slide-rendering XML code from user instructions, and then it is further optimized using DPO to output XML that matches preferred font style design. Our key contributions are as follows:

- We introduce SlideTuner, a custom finetuned instruction-to-XML generation model with GPT-4o [4] as backbone
- Contrary to previous work on slide generation that focuses on Python code generation, we empirically demonstrate that by training GPT-4o with XML dataset, better slide quality can be achieved despite much longer output tokens are required for slide XML generation.

We organize the paper as followed. Section II discuss related works and how our work differ from existing work. Section III describes the preliminaries of the PowerPoint slide, including the folder structure and XML files. Section IV describe the proposed SFT and DPO [5] training details for GPT-4o model and section V presents the experimental results. Section VI

compares the Python code vs XML represenation for PowerPoint slide generation. Section VII concludes the paper and discuss about limitation and future works.

II. Related Work

AutoPresent [3] explores the task of slide generation by finetuning large language models on instruction-to-code pairs. Given a natural language prompt, the model generates Python code that creates a corresponding slide. While AutoPresent demonstrates that SFT can produce functional and semantically aligned slides, the slides that it generates are often very simplistic and do not contain sophisticate visual elements (such as svg images that can compose visually appealing patterns). As a result, the model often struggles with aesthetic quality, relying on visual language model to generate large png image.

SlideCoder [6] advances layout-aware slide generation by introducing a retrieval-augmented pipeline that uses reference images and hierarchical visual segmentation to guide Python code generation. Its Slide2Code benchmark and Slide Complexity Metric provide structured ways to evaluate layout fidelity, particularly for slides with complex visual arrangements. However, the model requires explicit reference images to function effectively, and it does not support open-ended instruction inputs for layout aesthetics, limiting its applicability to only template style slide generation tasks.

[7] focuses on editing existing presentation slides using natural language commands. It decomposes user requests into high-level plans and executes them via structured code generation, allowing efficient and interpretable slide manipulation without relying on GUI actions. While this approach significantly improves usability and editability, it is designed for modifying existing slides rather than creating them from scratch. Moreover, it does not address the problem of aesthetic layout generation or include mechanisms to evaluate and refine visual quality through learning.

PPTAgent [8] formulates slide generation as an edit-based process over reference decks. It first clusters and extracts presentation schemas, then uses a model to generate Python code to adapt those templates to new content. PPTAgent also introduces PPTEval, a metric for measuring design, coherence, and content coverage. Despite its success in leveraging structural patterns from existing slides, PPTAgent again uses model to output Python code to create slide.

All the previous work above have Python code in their model or agent output in order to create or modify a slide. In this work, we want to use another approach: generating XML to create slide directly.

III. Preliminaries

With the release of Microsoft PowerPoint 2007, Microsoft adopted the Office Open XML (OOXML) standard for presentation slide. Each PowerPoint slide can be constructed with files and folders including the key internal components and directory paths that constitute a standard .pptx file, specifying their respective functions as defined by the Open Office XML specification. These components are logically organized to manage distinct aspects of the presentation: for example, the ppt/slides/ path contains the individual XML files that define the actual slide contents; the ppt/theme/ and ppt/slideLayouts/ directories govern the aesthetic properties, setting the default fonts, colors, and the positioning of placeholders.

IV. Proposed SlideTuner

Our proposed SlideTuner model addresses the challenge of precise, aesthetic-aware slide generation through a two-stage design as shown in Figure 1). Specifically, the process begins with SFT, where the language model is taught to translate natural language instructions directly into the fundamental slide-rendering XML code derived from native PowerPoint files. This is followed by a crucial aesthetic refinement phase utilizing DPO, which aligns the model's output with high-level design preferences.

A. Supervised Finetuning

We initiate our training pipeline with the SFT stage, namely stage I, focusing on generating the semantically correct XML structure. The dataset is constructed from rigorously curated pairs of (instruction, program code), where the instruction describes the slide content and intent, and the program code is the direct technical blueprint for the slide, which uses human-readable tags to define the structure, content, and precise visual properties of every element on the slide. This target program code is obtained by systematically reverse-engineering thousands of native PowerPoint files: a .pptx file is treated as a ZIP archive and it is unzipped to expose its underlying XML components (e.g., slideX.xml, theme files, and relationship definitions). Then we apply a flattening to all the components, which integrates them into one text file by concatenation (Figure 1). This approach deliberately bypasses intermediary libraries, allowing for complete control over visual parameters.

We utilize the powerful GPT-4o as our backbone model to generate XML code. The training set contains 1,000 examples that encompass a wide spectrum of functional slide layouts, including standard title slides, bullet points, image-and-caption, and comparisons. The model is optimized using cross-entropy loss between the generated program code and the reference program code in the dataset, ensuring a highly accurate and compliant translation from natural language to the final presentation structure.

An example shown in Figure 2 illustrates the core process of SlideTuner training to convert a natural language Instruction derived from the content and aesthetic rules of a validated PowerPoint slide directly into the Flattened XML Code. (the program code), which is the complete, concatenated XML component extracted directly from the .pptx file. This approach forces the model to learn the precise, low-level mapping between a user's intent and the final presentation file structure.

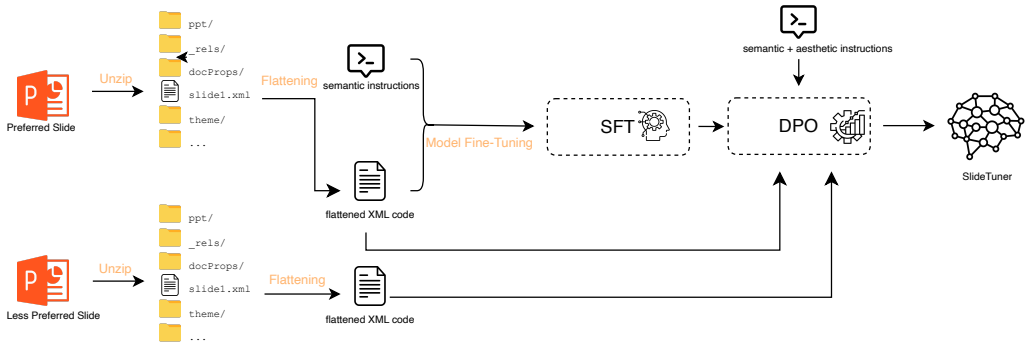

Fig. 1. Illustration of the two-stage training for model SlideTuner. PowerPoint slide is first unzipped and flatterned to construct training dataset for SFT; And then preference training dataset is constructed for DPO training; "semantic instruction" is system prompt plus extracted content from PowerPoint slide in the SFT training stage; "semantic + aesthetic instruction" is system prompt plus extracted content and font color & size information

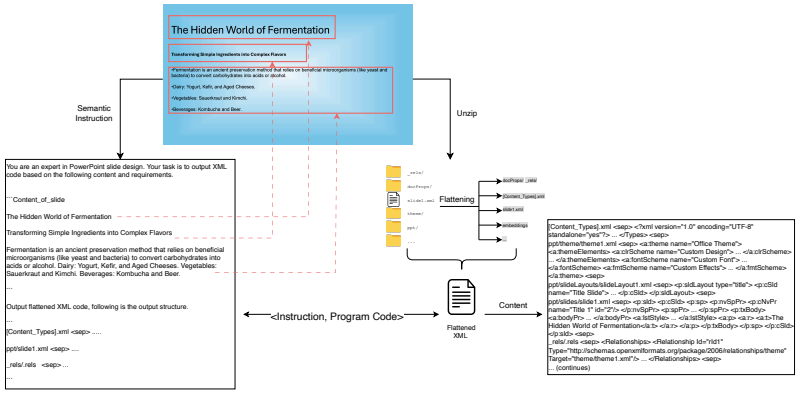

Fig. 2. This diagram illustrates the core process of training the SlideTuner model to translate natural language into the slide's structural definition. The Instruction is synthesized from the content and layout requirements of an internal, aesthetically validated PowerPoint slide. The model's task is to output the Flattened XML Code (the program code), which is the complete, concatenated XML component extracted directly from the .pptx file. This approach forces the model to learn the precise, low-level mapping between a user's intent and the final presentation file structure.

B. DPO

In the stage II, to move beyond the basic technical correctness achieved by SFT and achieve true aesthetic quality, we implemented DPO to further refine the visual aesthetics of the generated slides through SFT. Thus, for every input instruction, we sample two candidate XML outputs including a "preferred" slide, and a "less preferred" slide. The DPO loss function then optimizes the policy to significantly increase the probability of producing the "preferred" slide's XML structure over the "less preferred" one. This preference learning is formalized as follows in Equation 1 , which effectively guides the model towards superior aesthetic choices.

$$\mathcal{J}_\pi(\phi) = -\log\sigma\left(\beta'\frac{\log\pi_\phi(a^+\mid o_t)}{\log\pi_{\text{ref}}(a^+\mid o_t)} - \beta'\frac{\log\pi_\phi(a^-\mid o_t)}{\log\pi_{\text{ref}}(a^-\mid o_t)}\right) \quad (1)$$

The objective function $J_\pi(\phi)$ is designed to optimize the policy parameters ϕ by directly leveraging preference data. It employs a negative log-sigmoid loss $(-\log\sigma)$ to maximize the difference between the scores of a preferred response (a^+) and a dispreferred response (a^-) given an observation o_t. The score for a response is defined as the ratio of its log-probability under the current policy π_ϕ to its log-probability under a fixed reference policy π_{ref}, i.e., $\frac{\log\pi_\phi(a|o_t)}{\log\pi_{\text{ref}}(a|o_t)}$. The β' coefficient is a hyperparameter that scales the strength of this preference signal.

For implementation, we constructed a dataset of 1,000 preference pairs. The main challenge in creating such a dataset was ensuring that stylistic differences were learned apart from the semantic content of the slide. In order to overcome the issue, we leveraged our internal library of neatly organized and visually appealing slides, which act as the "preferred" version for every pair. Then, while preserving the exact same semantics, we created the "less preferred" version by intentionally introducing random degradations to the stylistic elements, such as randomizing the font font size and font color. Thus, paired examples with identical content but explicitly preferred or less preferred aesthetic styles were produced, which effectively isolates the model's learning to purely visual and design preferences. The hyper-parameters used in the DPO phase are shown in Table I.

Figure 3 illustrates the DPO training data construction using an (X, Y^+, Y^-) triplet. A Python script is used to generate a less preferred slide (Y^-) by altering the font style and size

TABLE I
DPO FINETUNING HYPERPARAMETERS

Setting	Value / Description
Initialization	DPO policy initialized from SFT checkpoint
Optimizer	AdamW
Learning Rate	1×10^{-5}
Batch Size	16
Epochs	3
Preference Pairs	1,000
Training Platform	Azure AI Foundry

of an existing preferred slide (Y^+). The input instruction (X) captures both the slide's semantic content and its aesthetic requirements, allowing the model to learn the difference between structurally precise (preferred) and aesthetically flawed (less preferred) XML code.

V. EXPERIMENT RESULTS

All experiments are run on Azure cloud computing platform. The evaluation criteria for the model are divided into two categories: Reference-based metrics, which include Code BLEU and exact match accuracy between the generated and reference code, and Reference-free metrics, which comprise a layout quality score, which is adopted from [3].

A. Study on DPO

To understand the impact of DPO on our model's performance, we conducted an ablation study. The baseline is the XML SFT GPT-4o model, where we apply SFT to train GPT-4o to generate XML code directly from input instructions. The full SlideTuner model, XML SFT+DPO GPT-4o, includes the additional DPO stage designed to further refine the model's output, especially in terms of aligning with aesthetic preferences as we discussed in section IV. We evaluate three metrics in this study. The first is the Render Success rate, which measures whether the model was able to successfully render a slide from the generated XML. For successfully rendered slides, we then assess two quality-based metrics.

The Layout Score reflects structural quality such as whether all slide elements are contained within the canvas, properly aligned, and not overlapping. A higher Layout Score indicates more balanced and visually coherent layouts.

Lastly, Style Adherence measures how closely the generated slides match the reference in terms of font and color choices. This metric captures whether the model respects stylistic instructions embedded in the prompt or reference design.

The results presented in Table II demonstrate that while the DPO stage provides a marginal increase in the Layout Score (from 0.72 to 0.73). However, style adherence changes from 0.51 to 0.62 and render success rate changes from 14.2% to 18.2%. This suggests that the DPO objective, by refining the model's generation policy based on preference pairs, considerably reduces stylistic and structural errors in the XML output that would otherwise prevent the file from rendering correctly.

B. Font Style Preference

We evaluated the capacity of SlideTuner to adhere to specific aesthetic instructions, namely font and color requirements (Table III). This metric measures the percentage of XML generations, over 100 repetitions per test sample, where the font and color of the generated slide are consistent with the internal reference slide.

The baseline GPT-4o model, when generating XML directly from the instructions, demonstrated limited control over specific aesthetic attributes, achieving an adherence rate of 58% for font and 55% for color. The rate dropped significantly to 42% when requiring simultaneous compliance with both font and color instructions.

The initial training stage provided substantial uplift compared with the baseline GPT-4o model. The SFT GPT-4o model, trained on 1,000 XML structural definitions derived from thousands of internal PowerPoint slides, showed a marked increase in adherence compared to the GPT-4o baseline. By successfully learning to translate instructions into the complete structural XML, the SFT model established a stronger base for aesthetic consistency, achieving 68% for font, which approximately reaches a 10 percentage point gain, and 64% for color. Its combined adherence rate also improved substantially to 52%, confirming that the STF step is essential for acquiring fundamental aesthetic control over the generated slides.

Crucially, the two-stage training framework, incorporating the aesthetic refinement from DPO, yielded a decisive improvement across all metrics. SlideTuner achieved the highest adherence rates, successfully implementing the requested font style 81% of the time, and the requested color scheme 78% of the time. This performance culminates in a combined Font and Color adherence rate of 69%, confirming that the DPO phase successfully guides the model toward superior aesthetic control by explicitly learning and reproducing the design preferences isolated during the dataset construction process.

C. Underspecified Instructions

A key challenge in slide generation is handling vague or underspecified instructions. We evaluate how well SlideTuner performs when given minimal guidance compared to baseline approaches, quantifying the models' ability to infer design choices and organize content effectively when explicit details are missing. Table IV summarizes this performance using a $1-5$ rating scale (5 being the highest score) for two critical metrics: Visual Quality and Content Organization. We use Visual Large Language model(VLLM) as the judge and create screenshot of the slide page as the input prompt and ask VLLM to give it a score from 1 to 5. The results in Table IV clearly demonstrate the benefit of our full two-stage approach in tackling ambiguous prompts. The baseline GPT-4o model performed modestly, achieving scores of 3.2 for Visual Quality and 3.3 for Content Organization. The SFT-only variant, which benefits from exposure to the structural XML definitions, showed improvement, establishing a higher baseline of 3.5 and 3.6, respectively. This improvement suggests that supervised

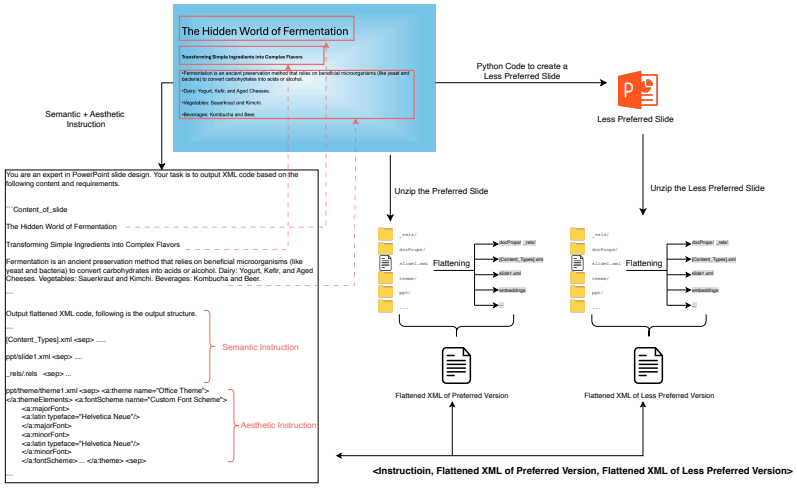

Fig. 3. This diagram illustrate the DPO training data construction; For every preferred PowerPoint slide, we use a python script to create a less preferred version of it by modifying its font style and size. We construct (X, Y+, Y-) triplet for DPO training, where X is instruction prompt with semantic and aesthetic instruction, Y+ is the flattened XML code of the preferred PowerPoint slide and Y- is the flatterned XML code of the less preferred PowerPoint slide

TABLE II
ABLATION RESULTS ON DPO COMPONENTS

Model Variant	Layout Score	Style (color and font) Adherence	Render Success Rate (%)
XML SFT GPT-4o	0.72	0.51	14.2
XML SFT+DPO GPT-4o	0.73	0.62	18.2

training on structured XML provides the model with fundamental knowledge of good layout and organization, allowing it to make better implicit decisions when instructions are vague. Crucially, SlideTuner (SFT+DPO) showed a decisive leap in generalization, achieving the highest scores for both metrics (4.1 for Visual Quality and 4.0 for Content Organization). The DPO stage, by learning explicit aesthetic preferences and stylistic alignments, enables the model to effectively "fill in the blanks" left by underspecified instructions. The DPO-trained model consistently defaults to superior organizational structure and visual balance, leading to a robust improvement in quality and layout when operating with minimal guidance.

D. Discussion

The development of SlideTuner demonstrates advance toward achieving aesthetic-aware slide generation via direct XML output. Our experiments confirm the necessity of our XML-first approach and the incremental value of each training stage. The initial SFT stage, trained on the 1000 flatterned XML codes derived from reverse-engineered PowerPoint slides, successfully provided the model with the fundamental domain-specific knowledge required for structural and aesthetic consistency. This is evidenced by the SFT GPT-4o model's marked improvement over the baseline GPT-4o, notably achieving a combined Font and Color adherence rate of 52% (compared to 42% for the baseline). The subsequent DPO stage proved essential for both aesthetic refinement and technical stability.

TABLE III
FONT AND COLOR STYLE PREFERENCE ADHERENCE RATE (%)

Models	Font	Color	Font and Color
GPT-4o	58	55	42
SFT GPT-4o	68	64	52
SlideTuner	81	78	69

The full SlideTuner (SFT+DPO) model achieved the highest stylistic compliance, with a 69% adherence rate for generating both the correct font and color scheme, demonstrating the DPO phase successfully guides the model toward superior, preferred aesthetic control. Furthermore, the ablation study (Table I) revealed that the DPO component significantly improved the Render Success Rate by 4 percentage points (from 14.2% to 18.2%), indicating that optimization for aesthetic quality simultaneously refines the model's ability to produce robust, functional XML output.

Crucially, SlideTuner demonstrated superior generalization capabilities when faced with ambiguous prompts. In the evaluation involving underspecified instructions (Table III), the full framework achieved the highest scores for both Visual Quality (4.1) and Content Organization (4.0). This result confirms that by optimizing the model for explicit aesthetic preferences via DPO, we enable it to effectively infer and apply high-quality design principles, thereby filling in the missing stylistic blanks left by vague user commands.

TABLE IV
PERFORMANCE ON UNDERSPECIFIED INSTRUCTIONS

Model	Visual Quality	Content Organization
GPT-4o	3.2	3.3
SFT (Ours)	3.5	3.6
SlideTuner (Ours)	**4.1**	**4.0**

VI. EVALUATION OF RICHNESS IN SLIDE REPRESENTATIONS

To robustly evaluate the expressive richness and structural fidelity of generated slides, we perform a detailed XML tag analysis comparing the XML produced directly by our XML-based model with the XML extracted from PPTX files generated by the Python-code-based model. By normalizing both outputs to the Open XML format, we enable a fair, feature-level comparison of slide content, formatting, and structure.

A. Compare With Model that Generate Python Code

To rigorously compare the fine-grained control offered by our direct XML-generating model against a standard Python-generating approach, we generated two slides for each test instruction: one using the XML-genearting model and one using a Python-generating model. The Python-generated PPTX files were converted back to XML using standard extraction tools for a direct comparison. We then analyzed the resulting XML structures across five critical dimensions: Tag Diversity and Frequency, Feature Tag Coverage, Tag Depth and Nesting, Attribute Richness, and Structural Completeness.

Our analysis reveals several important findings that validate the superiority of direct XML generation. Regarding Tag Diversity and Frequency, the XML-generating model consistently produced slides with a significantly higher number of unique tags, reflecting a broader support for advanced features and formatting options compared to the Python-generating model, which tended to omit tags related to complex effects or custom styling. This greater diversity translated directly into Feature Tag Coverage: slides generated by the XML-generating model showed comprehensive support for specific features, including tags for text styling (`<a:font>`, `<a:rPr>`), shapes (`<p:sp>`, `<p:grpSp>`), visual effects (`<a:effectLst>`, `<a:gradFill>`), media (`<p:pic>`), and layout/theme integration (`<p:sldLayout>`, `<a:theme>`).

In contrast, the Python-generated slides frequently lacked tags for advanced effects and often defaulted to basic formatting, limiting their aesthetic potential. Furthermore, we observed a crucial difference in structural complexity. The XML-generating slides exhibit both deeper nesting and a higher average number of attributes per tag. This is especially evident in slides featuring grouped shapes or nested content, where the greater nesting depth reflects more complex hierarchical structures and the increased attribute count indicates richer, more detailed formatting.

B. XML Tag Analysis Table

As shown and summarized in Table V, the XML Model utilized 177 unique tags, significantly more than the Python Model's 130, and generated a total of 5,534 tags, more than double the Python Model's $2,458$. This difference in complexity is further underscored by the XML hierarchy: the Python Model's maximum tag depth was only 10, compared to the **80** demonstrated by our XML Model, reflecting a far richer potential for complex, nested structures. Functionally, the Python Model showed only Partial or Rare support for key aesthetic elements such as Text Styling, Visual Effects, and Theme Integration, and failed entirely to implement Shape Grouping Tags. In contrast, the XML Model achieved full support across all these advanced feature categories, validating the necessity of bypassing abstraction libraries to gain granular control over the final presentation structure.

Therefore, our above detailed analysis confirms the superiority of our direct XML-generating generation approach over the intermediary Python approach for producing structurally rich slides. But it should also be noted that Python code generation has higher render rate as reported in [3].

TABLE V
XML TAG ANALYSIS FOR SLIDE RICHNESS

Tag/Feature	XML Model	Python Model
Unique Tags Used	177	130
Number of Tags Used	5534	2458
Max Tag Depth	80	10
Text Styling Tags	Yes	Partial
Shape Grouping Tags	Yes	No
Visual Effect Tags	Yes	Rare
Theme Integration Tags	Yes	Partial
Media Tags	Yes	Yes

VII. CONCLUSION

We finetuned GPT-4o model for generating presentation slides from natural language instructions using supervised instruction-to-XML finetuning and DPO. In our experiments, we targeted font size and color and improved visual quality and style adherence. Our experiment show considerable gains in font style adherence and render success rate as we after SFT and DPO. XML tag analysis indicates that direct XML generation offers richer formatting expressiveness than Python-code pipelines with the trade-off render success rate.

VIII. AI-GENERATED CONTENT ACKNOWLEDGEMENT

The authors used ChatGPT[1] (powered by GPT-5) to refine the wording in Section I and Section IV. The AI did not contribute to any original research ideas.

REFERENCES

[1] S. Al Masum, M. Ishizuka, and M. Islam, "'auto-presentation': a multi-agent system for building automatic multi-modal presentation of a topic from world wide web information," in *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, 2005, pp. 246–249.

[1] https://chatgpt.com/

[2] A. Sefid, J. Wu, P. Mitra, and C. L. Giles, "Automatic slide generation for scientific papers," in *Proceedings of the Third International Workshop on Capturing Scientific Knowledge co-located with the 10th International Conference on Knowledge Capture (K-CAP 2019)*, ser. CEUR Workshop Proceedings, vol. 2526, 2019, pp. 11–16. [Online]. Available: https://par.nsf.gov/servlets/purl/10173903

[3] J. Ge, Z. Z. Wang, X. Zhou, Y.-H. Peng, S. Subramanian, Q. Tan, and et al., "AutoPresent: Designing structured visuals from scratch," 2025.

[4] J. Achiam and et al., "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[5] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2024. [Online]. Available: https://arxiv.org/abs/2305.18290

[6] W. Tang, J. Xiao, W. Jiang, X. Xiao, Y. Wang, X. Tang, and et al., "Slidecoder: Layout-aware rag-enhanced hierarchical slide generation from design," 2025. [Online]. Available: https://arxiv.org/abs/2506.07964

[7] K. Jung, H. Cho, J. Yun, S. Yang, J. Jang, and J. Choo, "Talk to your slides: Language-driven agents for efficient slide editing," 2025. [Online]. Available: https://arxiv.org/abs/2505.11604

[8] H. Zheng, X. Guan, H. Kong, J. Zheng, W. Zhou, H. Lin, and et al., "Pptagent: Generating and evaluating presentations beyond text-to-slides," 2025. [Online]. Available: https://arxiv.org/abs/2501.03936