SLIDEFORGE: Controllable Editing of Slides as Structured Artifacts

Haozhen Zheng[1*], **Fulin Wang**[1*], **Tianhu Xiong**[1], **Yingjie Yu**[1], **Shengyi Qian**[2], **Hanchao Yu**[2], **Alex Schwing**[1], **Klara Nahrstedt**[1], **Mingyuan Wu**[1]
[1]University of Illinois Urbana-Champaign, [2]Meta
{haozhen3, fulinw2, mw34}@illinois.edu

Abstract

Current AI agents compellingly describe slides. However, AI-assisted slide editing requires more than understanding: the output must retain layout, style, component structure, and native editability. Towards, AI-assisted slide editing, existing agents operate on screenshots or weak document representations and often fragment coherent visual units, rasterize editable content, or break layout.

In contrast, for controllable slide editing we introduce an agentic framework, SLIDEFORGE, which builds a Deck State Graph, an executable slide state that links visual decomposition, native PPTX object structure, and perceptual organization. By recovering human-referable components while retaining fine-grained editable structure, SLIDEFORGE supports theme-preserving reconstruction through slide-native operations and rendered-state verification. We further introduce an evaluation paradigm for controllable slide transformation that jointly measures component recovery, preservation, restyling consistency, visual quality, and native editability. Experiments show that SLIDEFORGE outperforms direct prompting, screenshot-based agents, and generic code-agent baselines across these dimensions. Code is available at https://github.com/UIUC-MONET/SLIDEFORGE.

1 Introduction

Presentation slides are a central medium for communicating knowledge in science, education, business, and design (Daffner, 2003; Baker et al., 2018; Kaplan, 2011). Yet despite recent advances of AI agents, reliable AI-assisted slide editing remains challenging. A central reason is a mismatch between how slides are typically perceived by multimodal agents and how they are authored and edited in practice. **A slide deck is not simply a sequence of rendered images in pixel spaces**: it is a structured artifact composed of editable text, figures, shapes, charts, tables, groups, themes, z-ordering, slide masters, and cross-slide visual conventions. Transforming such an artifact therefore requires more than recognizing its visual content; it requires recovering human-referable components, preserving their layout and relationships, and reconstructing the deck in a form that remains editable.

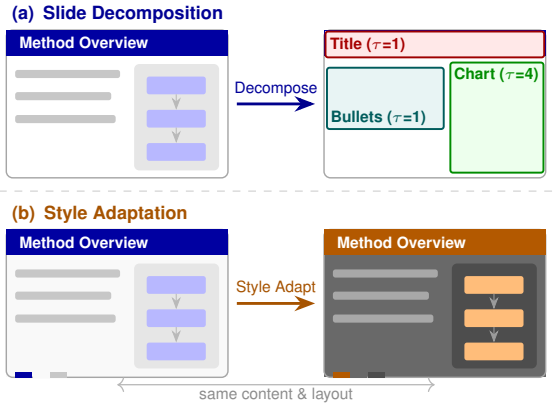


Figure 1: **SLIDEFORGE** treats slide editing as structured presentation editing. **(a) Perception-aligned decomposition** constructs a Deck State Graph (DSG) by recovering perceptually coherent, human-referable components and grounding them in editable PPTX structure. **(b) Theme-preserving reconstruction** uses the same DSG to restyle components under a target visual theme while preserving content, layout, and native editability.

Current AI agents often approach slide editing through direct prompting, screenshot-level reasoning, or generic code generation (Zheng et al., 2025; Jung et al., 2026). These approaches produce plausible edits, but they frequently break layout, alter non-target content, lose editability, or fail to maintain deck-level design consistency. The core challenge: slide editing is not only a visual generation problem. Instead, it requires structured-artifact manipulation, where visual appearance and document structure must be jointly understood and controlled.

[*]Equal contribution.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 38429–38471
October 24-29, 2026 ©2026 Association for Computational Linguistics

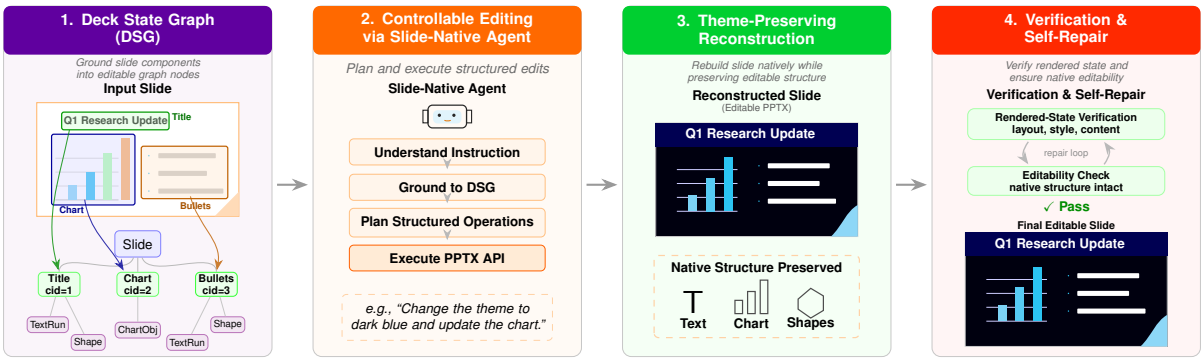


Figure 2: Overview of SLIDEFORGE: deck state graph construction from perception-aligned slide decomposition, slide-native editing, theme-preserving reconstruction, and verification with self-repair.

For this, we introduce SLIDEFORGE, an agentic framework for controllable editing of slide decks (see Figure 1). The key novelty of SLIDEFORGE: shift slide editing from direct multimodal generation to verified structured-artifact editing. Rather than asking a frontier model to rewrite or redraw a slide in one step within pixel space, SLIDEFORGE decomposes the deck, grounds instructions to editable state, executes constrained operations, and verifies the resulting artifact. This design supports diverse editing tasks, including local content updates, layout refinement, generated visual replacement, style adaptation, audience adaptation, and cross-slide consistency repair. For this, SLIDEFORGE represents a deck as an executable slide state that connects visual appearance, editable document structure, and perceptual organization, capturing human-referable components while retaining fine-grained structure for reconstruction.

Our contributions are:

- We formulate controllable slide editing as recovering and manipulating the structured state of a presentation, rather than solely regenerating its rendered appearance.
- We introduce the Deck State Graph, a representation that connects visual slide decomposition with native PPTX object structure, enabling grounded reasoning.
- We design a slide-native agentic pipeline that uses the Deck State Graph as an executable interface for theme-preserving reconstruction. The pipeline preserves native editability and uses rendered-state feedback to verify and repair visual or structural inconsistencies without fine-tuning frontier LLM/VLM backbones.
- We propose a comprehensive evaluation paradigm for controllable slide editing and show that SLIDEFORGE improves controllability, preservation, layout correctness, restyling quality, visual quality, and editability over direct prompting, screenshot-based agents, and generic code-agent baselines across diverse editing tasks.

2 Related Work

Slide generation and editing. Early document-to-slide systems such as PPSGen (Hu and Wan, 2013), D2S (Sun et al., 2021), and DOC2PPT (Fu et al., 2022) convert documents or scientific papers into slide decks through content selection, retrieval, summarization, and slide structure or layout prediction. Recent LLM-based and agentic systems further expand slide generation: PPTAgent (Zheng et al., 2025) uses reference presentations and editing actions to generate new slides; AutoPresent (Ge et al., 2025) generates editable slides from natural-language instructions through tool-augmented code generation; SlideGen (Liang et al., 2025) introduces a collaborative multimodal-agent pipeline with visual-in-the-loop refinement; SlideTailor (Zeng et al., 2026) conditions paper-to-slide generation on user preferences inferred from example slide pairs and templates; and DeepPresenter (Zheng et al., 2026) uses environment-grounded reflection to plan, render, and revise intermediate slide artifacts. In contrast, SLIDEFORGE focuses on controllably editing existing PPTX decks while preserving non-target content, layout, style, and editability.

Structured slide and visual artifact manipulation. Talk to Your Slides (Jung et al., 2026) shows that language-driven slide editing benefits from manipulating structured PowerPoint objects rather than relying solely on screenshots or screen pixels. Related poster generation and editing systems such as APEX (Shi et al., 2026), Paper2Poster (Pang et al., 2025), and PosterGen (Zhang et al., 2025b)

also emphasize structured design operations, layout planning, and visual evaluation or feedback. SLIDEFORGE builds on this object-centric direction, but introduces a Deck State Graph that links native PPTX objects with rendered visual components, semantic roles, spatial layout anchors, logical component groups, z-order, and deck-level style metadata. This representation enables controllable edits that preserve and restyle embedded figures and rasterized diagrams, support style migration, and maintain cross-slide visual consistency.

Graphic decomposition, layout, and evaluation. Graphic decomposition methods such as DeaM (Nie et al., 2025), LayerD (Suzuki et al., 2025), and Qwen-Image-Layered (Yin et al., 2025) recover layered RGBA representations and metadata from raster designs, enabling layer-level editing and reuse. Layout generation methods such as LayoutGPT (Feng et al., 2023) and LayoutDM (Inoue et al., 2023) study spatial planning for visual composition, while VASCAR (Zhang et al., 2025a) uses rendered-layout feedback and automatic metrics to iteratively refine content-aware layouts. Presentation-oriented benchmarks evaluate complementary aspects of slide systems. SlidesBench (Ge et al., 2025) focuses on slide generation from natural-language instructions, TSBench (Jung et al., 2026) evaluates language-driven slide editing, SlideAudit (Zhang et al., 2025c) targets automated detection of slide design flaws, and DECKBench (Jang et al., 2026) evaluates academic slide generation and multi-turn slide editing with instruction-following criteria. SLIDEFORGE differs by treating slide editing as a structured-artifact task, where success requires instruction fidelity, edit locality, content preservation, layout correctness, visual quality, and native PPTX editability.

3 Method

A presentation deck is both a visual artifact and an editable structured document, requiring the agent to reason about document structure before making edits. SLIDEFORGE addresses this challenge through a two-stage design in Figure 2. First, it constructs an executable deck state representation, the Deck State Graph, which links rendered visual components to native PPTX objects and structural relations in Section 3.2. Second, it uses this graph inside a slide-native agentic pipeline that performs localized planning, PPTX-level editing, rendered-state verification, and targeted repair in Section 3.3.

3.1 Problem Formulation

Let D denote an input presentation deck in PPTX XML and u a natural-language editing instruction. The goal is to produce an edited deck D' that satisfies u while preserving non-target content, layout, visual hierarchy, and native editability. For style adaptation, the instruction additionally specifies a target theme, and the output must restyle the deck without changing the underlying message.

3.2 Stage I: Deck State Graph Construction

The first stage constructs a *Deck State Graph* (DSG), an executable slide state that bridges three views of a presentation: native PPTX objects, rendered visual components, and human-referable editing units. This is the key novelty of SLIDEFORGE: instead of treating slides as screenshots or as raw document XML, the DSG connects what is visible, what is editable, and what must be preserved.

Native and visual state extraction. For each slide, SLIDEFORGE parses the PPTX XML to extract native editable objects, including text boxes, shapes, tables, charts, pictures, groups, and placeholders. For each object, we record its type, bounding box, text runs, style attributes, z-order, group membership, placeholder binding, and slide-master inheritance. This object layer provides direct handles for PPTX-level editing, but it is incomplete because many meaningful regions appear only as rasterized figures, screenshots, imported diagrams, or picture objects. To recover such regions, SLIDEFORGE also renders each slide and applies a slide-specific segmentation pipeline. We use a fine-tuned SAM-style vision-only model (Ravi et al., 2024), trained on a vocabulary of $|\Lambda| = 306$ slide component types. Each detected component e_i stores a mask, bounding box, opaque crop, and optional component type. These rendered components are later linked to native PPTX objects when possible, allowing the agent to distinguish directly editable elements from elements that require reconstruction.

Cross-modal grounding and refinement. Slide components are heterogeneous: some are easier to identify by semantic description, while others are easier to recognize visually. SLIDEFORGE therefore uses two complementary grounding paths. In the text-mediated path, a VLM describes visually distinct regions and an LLM maps these descriptions to labels in Λ. In the visual path, a VLM directly selects applicable labels from Λ by inspecting the rendered slide. Their union forms the slide-

conditioned vocabulary $\Lambda_{\mathbf{p}}$, which guides segmentation. Then the initial detections are refined using feedback from the rendered slide state. Validated regions are whitened to produce a cleaned image $\hat{\mathbf{p}}$, and a VLM reviewer checks whether substantive content remains uncovered, whether detections are invalid, and whether overlapping candidates represent independent components.

Perception-aligned component state. Since slide elements are naturally referenced and edited at different granularities SLIDEFORGE audits the detections into human-referable editing units. For example, a flowchart may contain many boxes and arrows but should often be edited as one diagram, while a large visual region may contain independently editable subcomponents. The final component representation is $\Omega = (\mathcal{E}_{\text{fine}}, \mathcal{M}, \mathcal{R})$ where $\mathcal{E}_{\text{fine}}$ contains fine-grained detections, $\mathcal{M}$ contains perceptual merge groups, and $\mathcal{R}$ contains substantive missed regions. This preserves both instruction-level grounding and fine-grained reconstruction ability.

Deck State Graph. The final DSG is a directed graph $G = (V, E)$. The node set V contains slide nodes, object nodes, and component nodes. Slide nodes store background, theme, and slide-master information; object nodes store native PPTX geometry, text, style, z-order, and grouping attributes; and component nodes store the recovered visual components in Ω. The edge set E encodes containment, reading order, part-of hierarchy, z-order, slide-master inheritance, recurring style patterns, and edit bindings between rendered components and native PPTX objects.

Thus, the DSG acts as the executable state for controllable slide editing. It exposes human-referable targets for instruction grounding, preserves native editable handles for PPTX-level operations, and provides structural constraints for protecting non-target content. See App. E.1 for detailed prompts.

3.3 Stage II: Slide-Native Editing Harness

Given an instruction u and the Deck State Graph $\mathcal{G}$, SLIDEFORGE treats $\mathcal{G}$ as an executable interface between user intent, rendered appearance, and native PPTX structure. We build a slide-native agent harness beyond for additional controllability.

Grounding instructions to editable state. The harness first grounds u to nodes and relations in $\mathcal{G}$, producing a *target scope* $\mathcal{S}_{\text{edit}}$ and a *preservation scope* $\mathcal{S}_{\text{keep}}$. This separation is central to controllability: before any modification, the agent knows which slide elements may change and which must remain fixed. Because $\mathcal{G}$ links semantic labels, text, geometry, perceptual groups, palette roles, and native PPTX objects, the same grounding step supports content edits, layout edits, style adaptation, and raster-component replacement over structured slide state rather than pixels alone.

Executing slide-native operations. An LLM planner converts the grounded instruction into a small set of PPTX-level operations, such as updating text, adjusting geometry, changing styles, editing tables, regrouping objects, or replacing non-editable raster components. Whenever possible, SLIDEFORGE edits native objects directly, so text remains editable as text, tables and shapes remain structured, and foreground elements stay anchored unless u requires movement. For rasterized or imported regions, $\mathcal{G}$ still provides semantic role, bounding box, and surrounding layout context, allowing the system to reconstruct only the affected component instead of disturbing the full slide.

Rendered-state verification and repair. After execution, SLIDEFORGE renders the modified deck and verifies the output against u and $\mathcal{S}_{\text{keep}}$, and refines (Madaan et al., 2023; Liao et al., 2025). The verifier checks instruction satisfaction, preservation of protected content, and presentation errors such as overflow, broken alignment, missing elements, or incorrect layering. When verification fails, the error is mapped back to the corresponding nodes in $\mathcal{G}$ and repaired locally.

3.4 Stage III: Style Adaptation via Theme-Preserving Reconstruction

Style adaptation restyles an existing deck under a target theme $\mathcal{T}$ while preserving content, layout, and native editability. SLIDEFORGE formulates this as *theme-preserving reconstruction* over the DSG: instead of regenerating slides as flat images, it reconstructs each foreground element with the document primitive that best preserves its function.

Structured aesthetic contract. Given $\mathcal{T}$, SLIDEFORGE builds a deck-level aesthetic contract $\mathcal{C}$, including role-specific backgrounds, dominant colors, role-labeled theme colors, font choices, and contrast-aware text colors. $\mathcal{C}$ provides a shared visual vocabulary for all slides, ensuring that style adaptation is globally consistent rather than independently applied to each element.

Modality-aware reconstruction. Each foreground component is assigned a reconstruction modality

$\tau_i \in \{0, \ldots, 5\}$: discardable ornament, editable text box, editable text panel, native table, shape composition, or styled raster asset. This modality typing preserves editability by matching each component to the most suitable PPTX primitive. Text remains editable text, tables remain native tables, diagrams are reconstructed as grouped shapes, and logos, screenshots, photos, or data-critical charts are adapted as bounded raster assets.

Palette-constrained assembly. For editable objects, SLIDEFORGE constructs a functionally consistent color mapping f_C from source colors to theme colors. Identical source colors map to identical target colors, preserving semantic differentiation such as headers versus body cells, while low-contrast text is recolored using the contrast-aware palette in $\mathcal{C}$. The final deck is assembled by placing the themed background and restoring each foreground component at its DSG anchor. Geometry and layout remain fixed unless bounded text fitting is required.

3.5 Stage IV: Verification and Self-repair

Reconstruction alone does not guarantee that the edited deck is visually correct. To make slide editing reliable, we introduce a rendered verification and self repair stage that closes the loop between native PPTX editing and rendered visual feedback.

Component verification. For reconstructed editable components $\tau_i \in \{1, 2, 3, 4\}$, SLIDEFORGE renders the single-element PPTX and compares it against the original component crop b_i. The verifier checks whether the reconstructed component preserves the intended content, remains inside its bounding box, and avoids local failures such as text overflow, table corruption, or shape drift. Failed components are repaired locally by revising only their parameterization, rather than regenerating the full slide.

Slide verification. After assembly, SLIDEFORGE renders the full deck and verifies it against the target theme $\mathcal{T}$, the aesthetic contract $\mathcal{C}$, and the DSG preservation constraints. The verifier checks style consistency, content preservation, readability, layout stability, layering, and native editability. When an error is detected, it is mapped back to the corresponding DSG node or relation, enabling localized repair of color, scale, z-order, or component reconstruction.

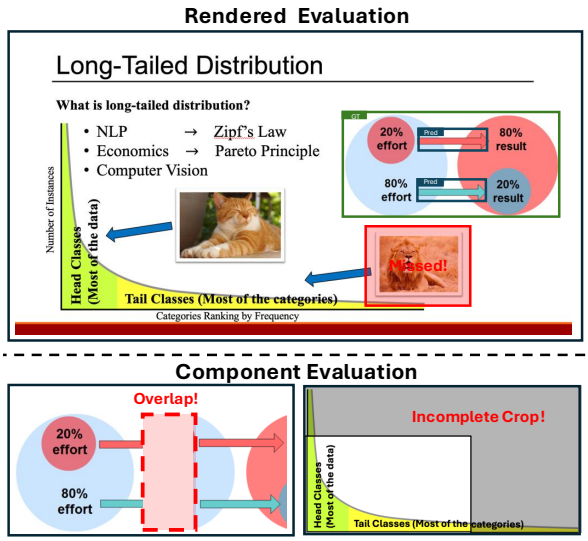


Figure 3: **Complementary Decomposition Evaluation.** Rendered evaluation compares the reassembled slide with the source when ground-truth and predicted boxes differ in granularity, while component-wise evaluation detects local failures such as overlapping components and incomplete crops.

4 Evaluation Paradigm

Controllable slide editing cannot be evaluated by visual plausibility alone. A system must recover a faithful structured state and transform it without losing content, layout, style, or editability. We evaluate SLIDEFORGE along two axes. State recovery measures whether the system reconstructs a structured representation that covers the original content, avoids fragmented components, and assigns meaningful labels. State transformation measures whether the recovered state supports theme-preserving reconstruction and editable restyling.

4.1 State Recovery: Decomposition

We evaluate decomposition as presentation-state recovery rather than object detection. Granularity mismatch makes box-level matching unreliable (Figure 3), so we combine rendered evaluation for reconstructability with component-wise evaluation for overlap, crop completeness, and label validity. A valid state satisfies three properties. *Reconstructability* means components recover the original artifact. *Coverage* means meaningful content is captured without redundancy. *Component validity* means each unit is semantically coherent and correctly typed. Our paradigm combines reference-based image similarity with reference-free VLM judgments (See App. E.3 for prompts), following recent practices for slide generation, decomposi-

Method	SSIM↑	LPIPS↓	CLIP↑	SCAN↑	Cohesion↑	Label Fit↑	Gran.↑	Non-Red.↑
SlideCoder	0.743	0.482	0.726	0.780	4.54	2.68	3.18	3.73
Claude Code (Opus 4.7)	0.813	0.341	0.837	0.858	4.54	2.82	3.90	3.89
SLIDEFORGE (Ours)	**0.950**	**0.054**	**0.984**	**0.960**	**4.70**	**4.50**	**4.68**	**4.87**

Table 1: Decomposition results on the out-of-domain system-conference test set. **Gran.** denotes Granularity and **Non-Red.** denotes Non-Redundancy. SLIDEFORGE achieves the best performance on all metrics.

Dimension	Industrial			Warm			Finance		
	AP	GI+C	SLIDEFORGE (Ours)	AP	GI+C	SLIDEFORGE (Ours)	AP	GI+C	SLIDEFORGE (Ours)
Deck-level coherence	**7.50**	7.03	7.39	**7.52**	6.83	7.14	**6.17**	6.03	6.10
Slide-level coherence	5.20	5.37	**6.00**	5.03	5.07	**5.90**	4.17	4.52	**5.14**
Information fidelity	6.03	4.37	**6.96**	5.38	4.00	**6.55**	5.38	4.45	**6.59**
Text readability	6.57	4.07	**6.79**	5.14	3.69	**6.38**	5.83	3.93	**6.66**
Content-decoration balance	5.17	5.53	**6.50**	4.21	5.00	**5.97**	4.00	4.90	**5.66**
Fitness for purpose	4.83	3.67	**5.71**	3.62	2.86	**4.90**	4.24	3.48	**5.34**
Overall	5.88	5.01	**6.56**	5.15	4.57	**6.14**	4.97	4.55	**5.91**

Table 2: Restyling evaluation across three target visual styles. Scores are reported on a 1–10 scale. AP denotes AutoPresent, and GI+C denotes GPT-Image-2 + Claude. SLIDEFORGE achieves the best overall score for all styles, while also improving information fidelity, readability, content-decoration balance, and fitness for purpose.

tion, and MLLM-as-judge assessment. (Tang et al., 2025; Ge et al., 2025; Muppidi et al., 2025; Zheng et al., 2025)

Rendered reconstruction. To measure reconstructability, we rebuild a slide from the extracted components using their predicted geometry and z-order, render the result, and compare it with the original slide. We report MSE, PSNR, SSIM (Nilsson and Akenine-Möller, 2020), LPIPS (Zhang et al., 2018), and CLIP (Radford et al., 2021) cosine similarity, capturing pixel-level, perceptual, and semantic agreement with the source rendering.

Content coverage and redundancy. To measure whether the recovered state captures slide content without unnecessary duplication, we compute foreground coverage and spatial non-overlap. Let F be the foreground mask, b_i be the predicted component boxes, $B = \cup_i b_i$, and $A = \sum_i |b_i|$. We compute $R_c = |F \cap B|/|F|$ and $R_o = 1 - (A - |B|)/A$, and linearly combine them as $\mathrm{SCAN} = \lambda R_c + (1 - \lambda) R_o$, where λ is set to 0.5

Component validity. We use a VLM judge to score each component crop on a 1–5 scale for *cohesion* and *label fit*, measuring whether it forms a complete editing unit and whether its assigned type matches the visual content.

Holistic state quality. Finally, we evaluate global decomposition quality by showing a VLM judge the original slide and the reconstructed rendering side by side. The judge scores coverage, granularity, and non-redundancy on a 1–5 scale, capturing failures that local metrics may miss, such as over-fragmented diagrams, duplicated regions, or semantically invalid groupings.

4.2 State Transformation: Restyling

We evaluate restyling as the quality of a transformed presentation state, asking whether the rendered slides inhabit the target style, whether original information is preserved, and whether the output remains a genuinely editable PPTX.

VLM aesthetic and functional judgments. We render each output PPTX through the same headless engine and pass all deck slides (as PNGs via LibreOffice) to a Claude-Sonnet-4.6 judge with the target style prompt. The judge scores six independent aspects on a 1 to 10 scale, each requiring a cited visual element. *Deck-level coherence* rates shared visual identity across slides in palette, typography, and motif. *Slide-level coherence* rates whether every element conforms to the target style. *Information fidelity* rates whether chart data, labels, and diagram structure stay intact. *Text readability* rates legibility of titles, body, and annotations at presentation scale. *Content and decoration balance* rates whether ornament supports rather than crowds the message. *Fitness for purpose* rates whether the deck would work in a realistic technical talk.

Editability preservation. To measure whether the restyled artifact remains a structurally editable PPTX, we extract all leaf shapes via python-pptx, treating any non-picture shape as editable. Let E_m denote the editable shape boxes of method m, $|b|$ the bounding-box area of b, and $S_w \cdot S_h$ the slide dimensions. For an output deck of n slides we

report

$$R_a = \frac{\sum_{b \in E_m} |b|}{n \cdot S_w \cdot S_h}, \qquad R_n = \frac{|E_m|}{|E_{\mathrm{GT}}|},$$

so $R_n = 1$ for ground truth, and methods that flatten content fall well below 1.

5 Experiments

We follow the evaluation paradigm described in Section 4 and focus on quality of *state recovery* and *state transformation*.

5.1 Dataset Collection and Pre-processing

Training data for SAM3. We collect **publicly available** presentation decks from the web, focusing on academic conference slides. This setting is challenging because academic slides contain dense text, equations, plots, tables, diagrams, and mixed raster/vector content while requiring high readability and precise layout preservation. For training the SAM3 backbone, we use 127 AI-conference decks. Bounding boxes are automatically derived from the native PPTX structure, and each component is mapped to the SAM3 input taxonomy via an LLM-based labeling pipeline. This avoids manual box annotation and adapts to new domains by changing the taxonomy and labeling prompts.

Evaluation data for controllable slide editing. To evaluate generalization, we use an out-of-domain test set of 89 system conference decks. For each deck, we include the first and last slides and uniformly sample six intermediate ones. The result remains academic in genre but differs from the AI-conference training distribution in topic, visual convention, and slide organization. Focusing on academic slides also enables a fair evaluation of style transfer. The source decks share similar communication goals, while the target styles (finance, warm, and industrial themes, with 30, 29, and 29 decks respectively) impose substantially different visual constraints.

5.2 Baselines and Metrics

We compare SLIDEFORGE against prior research systems and strong closed-source proprietary baselines, grouped by task. See App. E.2 for detailed baseline implementations.

- **SlideCoder** (Tang et al., 2025): a recent slide recovery pipeline used for decomposition.
- **Claude Opus 4.7** (Anthropic, 2026): a frontier proprietary agentic system, prompted for direct slide decomposition.
- **AutoPresent** (Ge et al., 2025): a prior slide-generation and editing pipeline that takes original slides image and target style prompt as input.
- **GPT-Image-2** (OpenAI, 2026) + **Claude Opus** 4.7 (Anthropic, 2026): a proprietary screenshot-first pipeline that renders a full slide with GPT-Image-2, then decomposes it back into editable PPTX with Claude Opus 4.7. It produces visually plausible slides but recovers editability only post-generation.

Evaluation Metrics. We follow the evaluation protocol in Section 4. For decomposition, we evaluate reconstruction with **SSIM**, **LPIPS**, and **CLIP**; coverage and redundancy with **SCAN**; component validity with **Cohesion** and **Label Fit**; and holistic state quality with **Granularity** and **Non-Red.** For restyling, we assess source preservation, target-style adherence, and native editability.

5.3 Decomposition Results

Table 1 shows that SLIDEFORGE achieves the best performance on all decomposition metrics under the out-of-domain system-conference test set. It improves rendered reconstruction to 0.950 SSIM, 0.984 CLIP, and 0.960 SCAN, while reducing LPIPS to 0.054, suggesting that the recovered components can be reassembled with high visual fidelity.

Beyond reconstruction, SLIDEFORGE also produces higher-quality editable state: Label Fit, Granularity, and Non-Red. rise to 4.50, 4.68, and 4.87, respectively. These results indicate that DSG grounding helps recover human-referable slide units that are more coherent, accurately typed, and less duplicative than those from generic code-agent baselines.

5.4 Restyling Results

Overall quality. As shown in table 2, SlideForge achieves the best Overall scores of 6.56, 6.14, and 5.91 across the three style targets. By decomposing and faithfully preserving text, shapes, tables, and charts and applying generation only to the background and selected foreground elements, it leads on Slide-level coherence, Information fidelity, Text readability, and Content-decoration balance while avoiding the baselines' redundant decoration. AutoPresent's whole-slide generation yields slightly stronger Deck-level coherence but severely harms fidelity and readability and unpredictably alters key elements such as data charts. Figure 4 confirms this qualitatively, where SLIDEFORGE repaints back-



Figure 4: **Case study.** AutoPresent yields distracting backgrounds and hallucinated charts; the GPT&Claude baseline fails to fully decompose foreground elements, corrupting content and background. SlideForge preserves foreground content faithfully while cleanly restyling the background. See App. F.2 for dark industrial machinery.

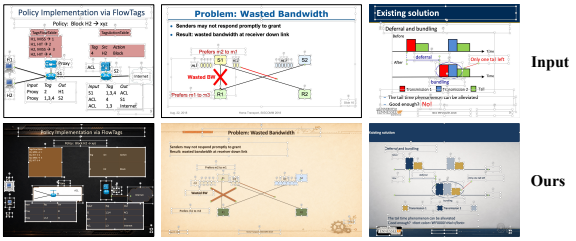


Figure 5: Case study showing that the outcome PPTX of SLIDEFORGE stays highly editable.

grounds and accents into the target mood while keeping titles, text, charts, and diagram structure faithful, whereas the baselines flatten elements or drift from the original content. See App. F.1 for more results.

Method	EC	EAR	EC/input
Input	71.2	62.36%	1.000
Autopresent	11.9	50.17%	0.266
Gpt_Claude	32.4	31.63%	0.696
ours	**72.8**	**49.74%**	**1.349**

Table 3: Comparison of editable count, area ratio, and count ratio against the input deck. EC is the average editable count and AR is the editable area ratio.

Editable portion. Table 3 reports the editability results on the restyled decks, where we count the editable leaf shapes per slide, measure the slide area they cover, and normalize the count against the input deck. SLIDEFORGE preserves on average 72.8 editable shapes per slide and even surpasses the input deck with a normalized editable count ratio of 1.349, whereas AutoPresent and the GPT&Claude baseline retain only 11.9 and 32.4 shapes with editable area ratios of 0.266 and 0.696. These numbers confirm that the baselines flatten much of the deck into static imagery while SLIDEFORGE keeps the artifact natively editable. Figure 5 makes this concrete by opening the restyled PPTX in an editor beside the input. Text stays selectable, tables and shapes remain structured objects, and charts keep their underlying data, so a user can revise wording, recolor a shape, or update a number directly. This matches Table 3 and shows that SLIDEFORGE returns an editable deck rather than a flattened picture.

Failure Analysis. For style adaptation, two failure modes remain (App. Figure 9). The $\tau = 4$ shape composition modality can break visual style on overly intricate shapes, as the backbone cannot express their geometry. The $\tau = 5$ raster path is limited by GPT-Image-2, which lacks transparent output, so images requiring an alpha channel fall back to a solid fill. Both failure stem from the base model's limited capacity.

5.5 Human Evaluation and Judge Validation

To validate whether the automatic and VLM-based evaluations in Sections 5.3 and 5.4 reflect human judgments, we conduct a human evaluation of the two primary tasks in our evaluation paradigm: state recovery through decomposition and state transformation through restyling. Eight participants took part in the study. For restyling, we stratify 30 decks from the test set, with 10 decks for each target style. Participants evaluate the same outputs using the same six-dimensional rubric as the VLM judge. For decomposition, participants evaluate the same annotated component visualizations presented to the VLM judge.

Method	Restyling ↑	Decomposition ↑
SLIDEFORGE	**7.91**	**4.68**
AutoPresent	5.34	–
GPT-Image-2 + Claude	6.91	–
Claude Code (Opus 4.7)	–	4.09
SlideCoder	–	3.07

Table 4: Human evaluation of restyling and decomposition. Restyling is evaluated on a 1–10 scale, while decomposition is evaluated on a 1–5 scale.

Task	Spearman ρ	Same winner	Pairwise ranking
Restyling	0.480	73.3%	72.6%
Decomposition	0.781	96.0%	92.6%

Table 5: Agreement between VLM-judge scores and mean human judgments.

Human judgments are consistent with the automatic and VLM-based evaluations: SLIDEFORGE achieves the highest overall score for both restyling (7.91) and decomposition (4.68), as shown in Table 4.

To quantify the reliability of the VLM-based evaluation, we additionally measure agreement between judge scores and mean human scores. We report item-level Spearman correlation, same-winner agreement, and pairwise ranking agreement.

As shown in Table 5, judge–human agreement is particularly strong for decomposition, with $\rho = 0.781$, 96.0% same-winner agreement, and 92.6% pairwise ranking agreement. This is consistent with decomposition criteria such as granularity, coverage, and non-redundancy being relatively well-defined structural properties. Restyling is more subjective, but still exhibits positive agreement across all three measures.

Finally, we test whether the human-evaluated restyling improvements are statistically significant. We compute per-deck score differences between SLIDEFORGE and each baseline for seven metrics—the six rubric dimensions and the overall score—across 87 evaluated decks. We apply two-sided Wilcoxon signed-rank tests followed by Holm correction over the 14 comparisons. SLIDEFORGE significantly outperforms the corresponding baseline in 13 of the 14 comparisons. Twelve comparisons have Holm-adjusted $p < 0.001$, while the comparison against GPT-Image-2 + Claude on deck-level coherence remains significant at $p = 0.012$. The only non-significant comparison is deck-level coherence against AutoPresent ($p = 0.157$).

Operation	# Elem.	≥ 4	≥ 3
Text modification	431	75.2%	91.6%
Text-panel restyling	24	95.8%	100%
Chart/table	11	54.5%	72.7%
Icon/diagram	132	47.7%	71.2%
Image replacement	8	75.0%	100%

Table 6: Element-wise evaluation of localized slide-native editing.

Beyond these task-quality judgments, we further evaluate perceptual realism through a Turing-test-style human study in Appendix A.

5.6 Controllability and Efficiency Analysis

Element-wise controllability. Beyond deck-level restyling quality, we evaluate whether SLIDEFORGE reliably executes localized slide-native editing operations. We evaluate every edited element from 30 sampled decks, covering text modification, text-panel restyling, chart/table and icon/diagram reconstruction, and generated-image replacement. As shown in Table 11, text-centric operations are particularly reliable: 91.6% of text modifications receive a score of at least 3, while text-panel restyling and image replacement reach 100%. More structurally complex chart/table and icon/diagram reconstruction achieve 72.7% and 71.2%, respectively. These results show that the deck-level improvements arise from reliable localized operations over the structured slide state rather than whole-slide regeneration.

Efficiency. On five six-slide decks, the full pipeline costs $4.59 per deck and takes 1080 s on average, including $2.18/416 s for decomposition and $2.41/664 s for restyling. No evaluated deck fails to converge, and restyling requires an average of only 3.0 localized self-corrections per deck; full statistics are provided in Appendix B.

6 Conclusion

We introduced SLIDEFORGE, a framework for controllable slide editing that treats decks as structured artifacts rather than rendered images. By connecting visual components, native PPTX objects, and preservation constraints through a Deck State Graph, SLIDEFORGE enables slide-native editing, theme-preserving reconstruction, and rendered verification. Extensive experiments show that SLIDEFORGE improves content preservation, readability, restyling quality, and native editability.

7 Limitations

Several opportunities remain for further optimization. The main remaining limitation in decomposition is the quality–efficiency trade-off. SLIDEFORGE prioritizes high-coverage, low-redundancy state recovery through iterative proposal refinement and visual review, which improves decomposition fidelity but introduces additional inference cost on content-dense academic slides.

Our pipeline also relies on frontier VLMs and rendered-state verification, which introduce substantial computational cost from both model inference and API calls. In addition, multiple verifier loops may propagate judgment errors when the verifier misidentifies visual or structural failures.

Our current task scope is also limited to controllable slide restyling and structured editing. The same framework could be extended to broader slide tasks, such as creative content rewriting, layout redesign, animation editing, and cross-deck consistency repair, but doing so would require more compute, richer tool support, and broader evaluation settings and more budgets.

Acknowledgements

This work was supported in part by NSF grants CNS-2437204, CNS-1900875, CCF-2217144, IIS-2008387, IIS-2045586, and IIS-2106825, as well as SBC UT Dallas grant 23011912 and SBC SUNY Albany grant 3-105218. This research used the Delta advanced computing and data resource, which is supported by the National Science Foundation (award OAC 2005572) and the State of Illinois. The opinions expressed in this publication are those of the authors and do not necessarily reflect those of the funding agencies.

References

Anthropic. 2026. Introducing claude opus 4.7. Accessed: 2026-05-25.

James P. Baker, Alan K. Goodboy, Nicholas David Bowman, and Alyssa A. Wright. 2018. Does teaching with powerpoint increase students' learning? a meta-analysis. *Comput. Educ.*, 126:376–387.

Richard Daffner. 2003. On improvement of scientific presentations: Using powerpoint. *AJR. American journal of roentgenology*, 181:47–9.

Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Reddy Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. 2023. LayoutGPT: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 18225–18250.

Tsu-Jui Fu, William Yang Wang, Daniel McDuff, and Yale Song. 2022. DOC2PPT: Automatic presentation slides generation from scientific documents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 634–642.

Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, and Trevor Darrell. 2025. AutoPresent: Designing structured visuals from scratch. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2902–2911.

Google DeepMind. 2026. Gemini 3.5 flash. `https://deepmind.google/models/model-cards/gemini-3-5-flash/`. Model Card, accessed August 19, 2026.

Yue Hu and Xiaojun Wan. 2013. PPSGen: Learning to generate presentation slides for academic papers. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2099–2105.

Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. 2023. LayoutDM: Discrete diffusion model for controllable layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10167–10176.

Daesik Jang, Morgan Lindsay Heisler, Linzi Xing, Yifei Li, Edward Wang, Ying Xiong, Yong Zhang, and Zhenan Fan. 2026. DECKBench: Benchmarking multi-agent frameworks for academic slide generation and editing. *Preprint*, arXiv:2602.13318.

Kyudan Jung, Hojun Cho, Jooyeol Yun, Soyoung Yang, Jaehyeok Jang, and Jaegul Choo. 2026. Talk to Your Slides: High-efficiency slide editing via language-driven structured data manipulation. *Preprint*, arXiv:2505.11604.

Sarah Kaplan. 2011. Strategy and powerpoint: An inquiry into the epistemic culture and machinery of strategy making. *Organization Science*, 22(2):320–346.

Xin Liang, Xiang Zhang, Yiwei Xu, Siqi Sun, and Chenyu You. 2025. SlideGen: Collaborative multimodal agents for scientific slide generation. *Preprint*, arXiv:2512.04529.

Yuan-Hong Liao, Sven Elflein, Liu He, Laura Leal-Taixé, Yejin Choi, Sanja Fidler, and David Acuna. 2025. Longperceptualthoughts: Distilling system-2 reasoning for system-1 perception. *Preprint*, arXiv:2504.15362.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36.

Ananth Muppidi, Tarak Das, Sambaran Bandyopadhyay, Tripti Shukla, and Dharun D A. 2025. Taming llms with negative samples: A reference-free framework to evaluate presentation content with actionable feedback. *Preprint*, arXiv:2505.18240.

Hui Nie, Zhao Zhang, Yutao Cheng, Maoke Yang, Gonglei Shi, Qingsong Xie, Jie Shao, and Xinglong Wu. 2025. Decomposition of graphic design with unified multimodal model. In *Proceedings of the 42nd International Conference on Machine Learning*.

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

OpenAI. 2026. Introducing chatgpt images 2.0. Accessed: 2026-05-25.

Wei Pang, Kevin Qinghong Lin, Xiangru Jian, Xi He, and Philip Torr. 2025. Paper2Poster: Towards multimodal poster automation from scientific papers. *Preprint*, arXiv:2505.21497.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. *Preprint*, arXiv:2103.00020.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, and 1 others. 2024. SAM 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*.

Chengxin Shi, Qinnan Cai, Zeyuan Chen, Long Zeng, Yibo Zhao, Jing Yu, Jianxiang Yu, and Xiang Li. 2026. APEX: Academic poster editing agentic expert. *Preprint*, arXiv:2601.04794.

Edward Sun, Yufang Hou, Dakuo Wang, Yunfeng Zhang, and Nancy X. R. Wang. 2021. D2S: Document-to-slide generation via query-based text

summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1405–1418, Online. Association for Computational Linguistics.

Tomoyuki Suzuki, Kang-Jun Liu, Naoto Inoue, and Kota Yamaguchi. 2025. LayerD: Decomposing raster graphic designs into layers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17783–17792.

Wenxin Tang, Jingyu Xiao, Wenxuan Jiang, Xi Xiao, Yuhang Wang, Xuxin Tang, Qing Li, Yuehe Ma, Junliang Liu, Shisong Tang, and Michael R. Lyu. 2025. Slidecoder: Layout-aware rag-enhanced hierarchical slide generation from design. *Preprint*, arXiv:2506.07964.

Shengming Yin, Zekai Zhang, Zecheng Tang, Kaiyuan Gao, Xiao Xu, Kun Yan, Jiahao Li, Yilei Chen, Yuxiang Chen, Heung-Yeung Shum, Lionel M. Ni, Jingren Zhou, Junyang Lin, and Chenfei Wu. 2025. Qwen-Image-Layered: Towards inherent editability via layer decomposition. *Preprint*, arXiv:2512.15603.

Wenzheng Zeng, Mingyu Ouyang, Langyuan Cui, and Hwee Tou Ng. 2026. SlideTailor: Personalized presentation slide generation for scientific papers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 34584–34592.

Jiahao Zhang, Ryota Yoshihashi, Shunsuke Kitada, Atsuki Osanai, and Yuta Nakashima. 2025a. VASCAR: Content-aware layout generation via visual-aware self-correction. *Preprint*, arXiv:2412.04237.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. *Preprint*, arXiv:1801.03924.

Zhilin Zhang, Xiang Zhang, Jiaqi Wei, Yiwei Xu, and Chenyu You. 2025b. PosterGen: Aesthetic-aware paper-to-poster generation via multi-agent llms. *Preprint*, arXiv:2508.17188.

Zhuohao Jerry Zhang, Ruiqi Chen, Mingyuan Zhong, and Jacob O. Wobbrock. 2025c. SlideAudit: A dataset and taxonomy for automated evaluation of presentation slides. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*.

Hao Zheng, Xinyan Guan, Hao Kong, Wenkai Zhang, Jia Zheng, Weixiang Zhou, Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. 2025. PPTAgent: Generating and evaluating presentations beyond text-to-slides. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14402–14418, Suzhou, China. Association for Computational Linguistics.

Hao Zheng, Guozhao Mo, Xinru Yan, Qianhao Yuan, Wenkai Zhang, Xuanang Chen, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. 2026. DeepPresenter: Environment-grounded reflection for agentic presentation generation. *Preprint*, arXiv:2602.22839.

Appendix

Appendix overview. The appendix provides additional evaluation, implementation details, evaluation prompts, results, and documentation for SLIDEFORGE. We first provide Section A provides an additional **Turing-test-style human evaluation** of whether reconstructed slides exhibit perceptible machine-generated artifacts. Section B provides **reproducibility, runtime, and cost analysis**, including training details for the SAM3-based component detector and end-to-end inference cost, runtime, model-call, and retry statistics. Section C examines **LLM-as-a-judge reliability** through independent evaluation with Claude Sonnet 4.6 and Gemini 3.5 Flash (Google DeepMind, 2026). Section D further evaluates **element-wise controllable editing**, measuring the reliability of localized operations including text modification, chart/table and icon/diagram reconstruction, and image replacement.

Section E details **the decomposition pipeline**, including the prompts used in Stage I (App. E.1), baseline implementations for SlideCoder and Claude Code (App. E.2.1; App. E.2.2), and the VLM-judge prompts used to evaluate component validity and holistic decomposition quality (App. E.3). We further provide qualitative decomposition examples in App. E.4. Section F describes **the style adaptation pipeline**, including detailed quantitative results (App. F.1), an additional qualitative case study (App. F.2), limitations and failure cases (App. F.3), the prompts used by the style adaptation pipeline (App. F.4), and the VLM-judge prompt used for style evaluation (App. F.5). Section G documents the dataset scope, privacy review, and content-safety checks, while Section H summarizes our use of LLMs for writing support, pipeline construction, and annotation assistance. Finally, Section I provides the human evaluation instructions.

A Turing-Test-Style Human Evaluation

We complement the main human evaluation in Section 5.5 with a Turing-test-style study that assesses whether reconstructed slides exhibit perceptible machine-generated artifacts. Eight participants took part in the study.

We conduct a single-stimulus Turing-test-style evaluation to measure whether reconstructed slides exhibit perceptible machine-generated artifacts. Each slide is presented individually, and participants are asked to classify it as either human-made or AI-generated. Human-original slides are included to estimate the false-alarm rate. To avoid direct pairwise comparison, participants never observe multiple versions of the same source slide.

Source	Judged as AI	d'	p-value
Human originals	23.6%	–	–
SLIDEFORGE	10.7%	-0.52	0.1751
Claude Code	77.4%	1.47	< 0.0001
SlideCoder	33.3%	0.29	0.6823

Table 7: Turing-test-style human evaluation. "Judged as AI" denotes the fraction of slides classified as AI-generated. Human originals provide the false-alarm baseline used to compute d'. Claude Code uses Opus 4.7.

We report the fraction of slides judged as AI-generated, together with d' and the corresponding p-value. Here, d' measures detectability relative to the human-original false-alarm rate: $d' \approx 0$ indicates similar detectability to human originals, whereas a larger positive value indicates that a method is more readily recognized as AI-generated. A negative d' indicates that the method is classified as AI-generated less frequently than the human originals; we therefore conservatively interpret negative values as providing no evidence of increased machine detectability. The p-value tests whether the AI-judgment rate differs significantly from the human-original false-alarm rate.

As shown in Table 7, SLIDEFORGE is judged as AI-generated in only 10.7% of cases, compared with a 23.6% false-alarm rate for human originals. We do not observe a significant difference between these rates ($p = 0.1751$), providing no evidence that SLIDEFORGE reconstructions are more detectable as machine-generated than human-made slides in this setting. In contrast, Claude Code (Opus 4.7) is identified as AI-generated substantially more often than human originals (77.4% vs. 23.6%, $p < 0.0001$). The strong separation for this baseline indicates that the protocol is sensitive to visible machine-generated artifacts. Together, these results suggest that SLIDEFORGE substantially reduces the perceptual human-machine gap in slide reconstruction.

B Reproducibility, Runtime, and Cost

We provide additional implementation and efficiency details for reproducibility. For the SAM3-

based component detector used during slide decomposition, we freeze the vision-language backbone and train only a lightweight decoder adaptation. This results in 30.4M trainable parameters, corresponding to 3.6% of the full model. The training set contains 127 academic PPTX decks, comprising 3,510 slides, 25,132 component bounding boxes, and 306 open-vocabulary component types. Training takes 4.7 hours on two NVIDIA RTX 4090 GPUs. On the slide-disjoint held-out split, the detector achieves a mean IoU of 0.873, with 95.3% of predictions reaching IoU@0.5 and 88.1% reaching IoU@0.75.

We additionally profile end-to-end inference on five test decks, each containing six slides. Table 8 reports average monetary cost, wall-clock runtime, model-call count, and retry statistics per deck.

Here, a repair or self-correction is a localized modification triggered by rendered-state verification, whereas a non-converged deck is one for which the pipeline fails to produce a valid final artifact after the complete retry and repair loop. In this evaluation, no deck fails to converge and no API-level retries are required. The restyling stage invokes an average of 3.0 localized self-corrections per deck, illustrating that rendered-state verification typically resolves errors without restarting the full generation process.

The framework is also model-adaptive. For example, replacing Opus 4.7 with Sonnet 4.6 for decomposition reduces the average decomposition cost from $2.18 to $0.98 per deck, at the expense of a runtime trade-off. These results make explicit the quality-efficiency trade-off of the agentic pipeline and provide practical estimates for reproducing the full system.

C Inter-Judge Validation of LLM-as-a-Judge Reliability

Because several evaluation dimensions require semantic or aesthetic judgment, we examine whether the reported conclusions depend on the choice of evaluator model. We independently evaluate the outputs using Claude Sonnet 4.6 and Gemini 3.5 Flash and compare the resulting method rankings.

For restyling, the two judges produce highly consistent conclusions. Claude Sonnet 4.6 ranks SLIDEFORGE first on six of the seven reported dimensions. The only exception is deck-level coherence, where AutoPresent receives a slightly higher score, consistent with its use of a more uniform background treatment. Gemini 3.5 Flash ranks SLIDEFORGE first on all seven dimensions.

We observe the same pattern for decomposition. As shown in Table 10, SLIDEFORGE obtains the highest average score under both evaluators. The only dimension-level exception is Claude Sonnet 4.6's cohesion score, for which Claude Code is marginally higher. All other decomposition dimensions favor SLIDEFORGE under both judges.

Overall, the relative ranking of methods is stable across the two evaluator models. The improvements of SLIDEFORGE therefore do not appear to be an artifact of a particular LLM judge.

D Details of Element-Wise Controllable Editing Evaluation

Section 5.6 reports the main results of our element-wise controllable editing evaluation. Here, we provide additional details on the evaluation protocol.

We evaluate every edited element from 30 sampled decks. For each element, Claude Sonnet 4.6 evaluates the final edited state on a 1–5 scale while conditioning on the full rendered slide, the element localization, and a local crop. We report the mean score as well as the percentages of elements receiving scores of at least 4 and at least 3.

Table 11 shows that text-centric operations and image replacement are particularly reliable. Text modification achieves a score of at least 3 for 91.6% of 431 elements, while text-label restyling and image replacement reach 100% at the same threshold. Chart/table and icon/diagram reconstruction remain more challenging, with 72.7% and 71.2% of elements, respectively, receiving scores of at least 3. These failures primarily arise when complex visual elements require more precise palette adaptation or reconstruction than the underlying models can reliably provide. The results nevertheless show that the deck-level gains of SLIDEFORGE are supported by successful localized edits across a broad range of element types.

Stage	Cost / deck	Runtime / deck	Calls / deck	Call breakdown	Failure / retry statistics
Decomposition	\$2.18	416 s	74.8	74.8 Opus	repair: 0; non-converged: 0
Restyling	\$2.41	664 s	74.0	44.4 Opus + 24.8 Sonnet + 4.8 image	self-correction: 3.0/deck; non-converged: 0; API retry: 0
Full pipeline	\$4.59	1080 s	148.8	119.2 Opus + 24.8 Sonnet + 4.8 image	–

Table 8: Average runtime, API cost, and model-call statistics measured on five six-slide test decks.

	Claude Sonnet 4.6			Gemini 3.5 Flash		
Metric	Ours	AutoPresent	GPT/C	Ours	AutoPresent	GPT/C
Deck-level coherence	6.88	**7.06**	6.63	**8.51**	7.94	8.16
Slide-level coherence	**5.68**	4.80	4.99	**5.95**	3.77	4.86
Information fidelity	**6.70**	5.60	4.27	**6.53**	5.12	2.85
Text readability	**6.61**	5.85	3.90	**6.65**	4.49	2.81
Content–decoration balance	**6.04**	4.46	5.14	**6.73**	3.28	3.86
Fitness for purpose	**5.32**	4.23	3.34	**5.41**	2.83	1.98
Overall	**6.20**	5.33	4.71	**6.63**	4.57	4.09

Table 9: Inter-judge validation for restyling. GPT/C denotes the GPT-Image-2 + Claude baseline. Bold indicates the best score under each judge.

	Claude Sonnet 4.6			Gemini 3.5 Flash		
Metric	Ours	Claude Code	SlideCoder	Ours	Claude Code	SlideCoder
Cohesion	4.552	**4.587**	4.463	**4.70**	4.54	4.54
Label fit	**4.343**	2.462	2.288	**4.50**	2.82	2.68
Granularity	**4.620**	3.592	3.364	**4.68**	3.90	3.18
Non-redundancy	**4.789**	4.163	3.864	**4.87**	3.89	3.73
Average	**4.576**	3.701	3.501	**4.688**	3.788	3.533

Table 10: Inter-judge validation for decomposition. Bold indicates the best score under each judge.

Operation type	# Elements	Avg. ↑	≥ 4	≥ 3
Text modification / recoloring / reformatting	431	3.90	75.2%	91.6%
Text-label restyling with background	24	4.54	95.8%	100%
Chart / table reconstruction	11	3.64	54.5%	72.7%
Icon / diagram reconstruction	132	3.27	47.7%	71.2%
Image update / generated-image replacement	8	4.12	75.0%	100%

Table 11: Element-wise evaluation of slide-native controllable editing on 30 sampled decks.

E Decomposition Pipeline Details

E.1 Prompt Used in Decomposition Pipeline

Prompt organization. This section lists the prompts used in the perception-aligned decomposition stage of SLIDEFORGE. These prompts correspond to the Stage I pipeline described in Section 3.2.

Prompt 1: Text-Mediated Component Proposal

```
System:
You are a vision assistant that decomposes
     presentation slides into their human-
     perceivable components. Return a list
     of short noun-phrase labels (2-5
     words) that a human would consider
     one discrete visual unit.

User:
Look at this presentation slide. List
     every distinct component a human
     would treat as one whole when
     authoring the slide.

Granularity rules:
- A bullet list is ONE component, not one
     per bullet.
- A chart (including axes, title, legend)
     is ONE component.
- A logo + company name together may be
     ONE component if visually grouped.
- Decorative background shapes count as
     components if they are visually
     intentional.

Return STRICT JSON only, no prose, no code
     fences, with this exact shape:
{"components": ["<short phrase 1>", "<
     short phrase 2>", ...]}
```

Prompt 2: Taxonomy Mapping for Component Proposals

```
System:
You map free-form visual component
     descriptions onto a fixed taxonomy.
     You never invent new classes. If
     nothing fits, omit the phrase.

User:
Here is the fixed class taxonomy (use ONLY
     these):
{class_list_json}

Here are the phrases to map:
{phrases_json}

For each phrase, pick the single best-fit
     class from the taxonomy, or null if
     none fit reasonably.
Return STRICT JSON only, no prose, no code
     fences:
{{"mappings": [{{"phrase": "<phrase>", "
     class": "<one of the taxonomy or null
     >"}}, ...]}}
```

Prompt 3: Visual Taxonomy Selection

```
System:
You are a vision assistant. Given a slide
     image and a fixed class taxonomy,
     select every class that is actually
     present in the image. Be precise and
     prefer recall over precision, but do
     not include classes that are not
     visible.

User:
Class taxonomy (pick ONLY from these exact
     strings):
{class_list_json}

Look at the slide image and select every
     class whose instance is visibly
     present.
Return STRICT JSON only, no prose, no code
     fences:
{{"classes": ["<class 1>", "<class 2>",
     ...]}}
```

Prompt 4: Rendered Coverage Review

```
System:
You are a QA reviewer. You look at a 'cleaned' slide image from which an automated pipeline has tried to remove every distinct component (text, charts, photos, shapes, etc. are replaced with white). Alongside the cleaned image you may also receive a coverage overlay: the ORIGINAL slide with red rectangles drawn around every region the pipeline ALREADY extracted. You decide whether the cleaning is complete, or whether another pass of the pipeline is warranted because obvious slide components OUTSIDE the red rectangles are still visible.

User:
You are given two images:

- IMAGE 1 ("cleaned image"): the output of a component-removal pipeline. Ideally every substantive visual component (title, bullets, paragraphs, charts with their content, photographs, diagrams, captions, etc.) has been replaced with white/blank pixels.
- IMAGE 2 ("bbox overlay", may be omitted): the ORIGINAL slide with red rectangles drawn around every region the pipeline already extracted as a component. Anything inside a red rectangle has been CLAIMED as extracted.

Your job: decide whether the cleanup is acceptable, or whether it is worth running the pipeline again on this image to recover MISSED or FAILED-EXTRACTION components.

Critical rule about the red rectangles (only when IMAGE 2 is provided):
- Red rectangles show attempted extractions. If you see only faint mask-cleanup residue **inside** a red rectangle  broken texture, dotted ghosts, scattered pixels, empty frames, thin lines, anti-aliasing halos  DO NOT trigger a rerun for that alone.
- **Important exception**: If a substantive, readable component remains largely intact **inside** a red rectangle (e.g., a paragraph you can read, a chart with its data still visible, a recognizable photograph/diagram/table, a logo/banner with legible text), treat this as a FAILED EXTRACTION. It DOES warrant a rerun, even though it is boxed.
- For regions **outside** all red rectangles, evaluate normally, but be sensitive to importance: even a small element counts if it is clearly intentional and meaningful (a short title, a key number, a single icon, a 13 word label, one bullet).

Set needs_rerun = TRUE when at least one SUBSTANTIVE component is still visibly intact, either:
- (A) mostly OUTSIDE every red rectangle, OR
- (B) mostly INSIDE a red rectangle but clearly not extracted (failed extraction).

SUBSTANTIVE components (trigger rerun):
- Readable text forming a meaningful unit (title, heading, bullet, label, caption, paragraph (even 13 words if clearly intentional, not random noise).
- A chart/graph with data curves, bars, fills, or legible axis labels.
- A photograph, illustration, diagram, table, infographic with recognizable content.
- A large icon, logo, or banner with readable elements.
- A key number, percentage, code, or symbol that conveys information.

Set needs_rerun = FALSE (i.e., "clean enough") when the remaining content is ONLY the following kinds of residue, regardless of location:
- Faint mask-cleanup residue inside red rectangles (as defined above).
- Chart or panel FRAMES / borders / rounded outlines with empty interiors.

- Thin horizontal or vertical LINES (axes without labels, rules, dividers, underlines).
- Random 13 character fragments that do not form a word/number/label.
- Thin DECORATIVE STRIPS / ribbons / bands without text.
- Anti-aliasing halos, stray pixels, tick marks, bullet dots.
- Page margins, blank background.

Positive examples (needs_rerun = FALSE / clean enough):
- Empty chart frames with no curves/labels, plus a thin colored bar at bottom edge, plus a 2-letter random fragment  FALSE.
- Blank slide with just a thin footer line  FALSE.
- Only axis lines and tick marks visible  FALSE.
- A red-boxed photograph area shows only broken texture/ghost, no recognizable subject  FALSE (already extracted).

Negative examples (needs_rerun = TRUE):
- A full chart with its curve and labels still visible, outside any red rectangle  TRUE.
```

```
- A paragraph or bullet text with most
    words intact, outside any red
    rectangle  TRUE.
- A photograph of an object clearly
    visible, outside any red rectangle
    TRUE.
- A red-boxed region still contains a
    readable paragraph (extraction failed
    )  TRUE.
- A single-word title "Results" outside
    any box, clearly intentional  TRUE.
- A small key number "42%" in the corner,
    outside any box  TRUE.

Return STRICT JSON only, no prose, no code
     fences:
{"needs_rerun": true|false, "reason": "<
    one short sentence>", "
    remaining_components": ["<short
    phrase>", ...]}
```

Prompt 5: Component Validity and Text Extraction

```
System:
You review an extracted slide-component
    crop. You decide: (1) whether the
    crop contains meaningful content or
    is essentially blank / noise /
    residue; (2) whether the crop is TEXT-
    ONLY (the entire component is just
    text, with no surrounding diagram /
    shape / chart / icon / photo context)
    ; and (3) when text-only, you
    transcribe the text into LaTeX (
    preserving any formulas, sub/
    superscripts, symbols). This crop is
    a bbox rectangle from the original
    slide  it is NOT mask-cut  so the
    pixels show the actual rendered
    region, possibly with some
    surrounding whitespace from the slide.


User:
You are given:

- IMAGE 1: ONE extracted component crop
    from a slide-decomposition pipeline.
- IMAGE 2 (optional, when present): the
    FULL ORIGINAL SLIDE with a red
  rectangle outline showing exactly where
       this crop's bbox sits on the
  slide. Use it to disambiguate narrow /
       sparse / character-shaped crops
  (e.g. brackets, mathematical operators,
       single letters, dividers,
  vertical bars) where IMAGE 1 alone is
       hard to interpret. The slide
  context tells you what content the bbox
       actually encloses and what
  surrounds it.

The pipeline believed IMAGE 1 held an
    instance of class "{class_name}" (
    filename: {filename}).
IMAGE 1 is an opaque rectangular region
    from the slide (bbox-only, no masking
    ).

When IMAGE 2 is provided, ALWAYS look at
    it before judging IMAGE 1
a thin or partial-looking crop is often a
    fully legitimate character /
delimiter / divider whose role is only
    clear in context.

Answer FOUR things:

1. valid (bool): Is this crop a real
    component?
   - FALSE when the crop is essentially
       blank, only 1-3 stray character
       fragments, a faint smeared ghost,
       a thin line, anti-aliasing halo, or
        pure edge debris.
   - TRUE when you can see at least one
       recognisable slide element:
       readable text (even a single word),
        a shape, icon, logo, arrow, chart
       fragment, diagram, photograph, or
       a coloured region that is clearly
       a deliberate graphic.

2. is_text_only (bool): Is this component
    composed ENTIRELY of text?
   - TRUE when the component is just
       rendered text (a title, a label, a
        paragraph, a caption, a formula,
       a numeric value, etc.) with no
       graphical element beyond the
       typographic shapes themselves.
   - FALSE when the component contains non
       -text graphics as part of the
       element: e.g. an icon with a label,
        a chart with axis numbers, a
       diagram with callouts, a labeled
       shape, a photograph with watermark
        text. The presence of any non-
       text graphical content that is
       part of the component makes it NOT
        text-only.
   - Only set is_text_only = TRUE if valid
        is also TRUE. If the crop is
       invalid (blank / residue), set
       is_text_only = FALSE.

3. ocr_latex (string): If (and only if)
    is_text_only is TRUE, transcribe the
    visible text into LaTeX source.
   - Preserve formulas: e.g. $E = mc^2$, \
       frac{{a}}{{b}}, x^{{2}}, \sum_{{i
       =1}}^{{n}} a_i.
   - Preserve line breaks with \\.
   - Preserve bullet markers as \item or
       plain hyphens inside an itemize-
       style block if you see a list.
   - Use plain LaTeX escaping for special
       characters ( % & _ # $ ).
   - If is_text_only is FALSE, return an
       empty string for ocr_latex.

4. bbox_is_tight (bool): Does the bbox
    tightly enclose the actual content,
    or does it
   include a noticeable amount of empty
       whitespace beyond the content?
   - This question matters ONLY when
       is_text_only is TRUE. For non-text
        crops, set
     bbox_is_tight = TRUE (the pipeline
          does not rewrite non-text bboxes)
          .
   - For text crops:
     - TRUE: the text fills most of the
         crop's width AND height;
         trailing /
       leading / top / bottom whitespace
           is small (each side roughly
           less
       than ~10% of the text's extent on
           that axis).
     - FALSE: there is substantial
         whitespace on one or more sides
         the
       text occupies clearly less than
           the crop's full extent (e.g.
           the text
```

```
        ends well before the right edge
            and a large white strip
            remains, or
        the text sits in the upper portion
             with a tall blank band below
            ).
        A typical sign: the bbox extends
            ~30%+ further than the actual
             text on
        one axis and the empty area could
            fit other slide content.
   - Set bbox_is_tight = TRUE if uncertain
       or marginal  only flag FALSE when
     the whitespace is obvious and
         substantial.

5. bbox_is_too_small (bool): Is the bbox
     too small, causing the text content
     to be
   visibly cut off or clipped at the edges
       ?
   - This question matters ONLY when
       is_text_only is TRUE. For non-text
        crops, set
     bbox_is_too_small = FALSE.
   - For text crops:
      - TRUE: the bounding box is too
          restrictive. Text glyphs,
          descenders,
        ascenders, or characters at the
            very edges are visibly
            chopped off.
      - FALSE: the text is fully
          contained within the crop. No
          parts of the
        characters are clipped by the
            edges.
   - Set bbox_is_too_small = FALSE if
       uncertain  only flag TRUE when the
        text is
     clearly missing parts due to the crop
          boundaries. Note: A bounding box
     cannot be both "too small" (TRUE here
         ) and have substantial whitespace
     (FALSE for bbox_is_tight).

Return STRICT JSON only, no prose, no code
      fences:
{{"valid": true|false, "is_text_only":
     true|false, "ocr_latex": "<latex
     string or empty>", "bbox_is_tight":
     true|false, "bbox_is_too_small": true
     |false, "reason": "<one short
     sentence>"}}
```

Prompt 6: Missed Region Validation

```
System:
You judge whether a small visual element
     inside a slide is a STANDALONE
     component or just a SUB-PART of a
     larger surrounding element. An icon
     or shape drawn inside a larger
     diagram is a sub-part of the diagram,
      not a standalone component. A title
     sitting above a chart is standalone.

User:
Look at the slide image. Two overlapping
     rectangular regions were detected by
     a grounding model:

- Region A (smaller, class="{class_a}"):
     bbox xyxy = {bbox_a}
- Region B (larger, class="{class_b}"):
     bbox xyxy = {bbox_b}

Region A is contained (fully or mostly)
     inside Region B.

Decide:

- is_part_of = TRUE if the content inside
     Region A is a sub-element, a
     decoration, or an integral visual
     piece of Region B's content. That is,
      treating B as a whole already
     visually includes A; separating A out
      would damage B's meaning. Examples:
     A is an icon drawn inside a diagram;
     A is a small shape that's part of a
     bigger illustration; A is a caption/
     label that belongs with a callout
     body.

- is_part_of = FALSE if Region A is an
     independently-authored component that
      merely happens to spatially overlap
     B. Examples: A is a standalone title,
      a caption or paragraph sitting on
     top of a decorative backdrop; A is a
     separate bullet/icon placed near B; A
      is a textual label sitting next to
     but not "inside" B's content.

Return STRICT JSON only, no prose, no code
      fences:
{{"is_part_of": true|false, "reason": "<
     one short sentence>"}}
```

Prompt 7: Missed Region Localization

```
System:
You review a slide-decomposition pipeline's coverage. The pipeline occasionally misses regions outright  substantive content sits outside every extracted bbox. Your sole job in this call is to identify those missed regions so they can be added as first-class components.

User:
You are given:

- IMAGE 1: the original slide.
- IMAGE 2: the same slide with a red rectangle drawn over every bbox the pipeline currently extracted (combined across all iterations, after cross-iter dedup and per-component validity pruning).

For reference, here is the structured list of those existing bboxes:

{components_json}

Allowed taxonomy classes (use ONLY these exact strings for `class`):
{taxonomy_json}

Identify SUBSTANTIVE content visible OUTSIDE every red rectangle that should have been a component (e.g. a callout label, a sub-diagram block, an output text label connected by a dashed arrow, a body paragraph the pipeline did not detect, a "Predictions" / "Output" terminal box at the end of a flowchart).

For each missed region provide:

- "bbox": [x1, y1, x2, y2] in ORIGINAL slide pixel coordinates.
- "class": one taxonomy class (MUST be from the list above).
- "description": one short sentence describing the missed content.

CRITICAL RULE  report regardless of downstream merging:

Always report a missed region whenever a HUMAN annotator would call it its own component, EVEN IF you suspect it will later be merged into a larger semantic group (e.g. you think the "Output" callout at the end of a flowchart will be merged into the flowchart in the next step). The pipeline keeps both granularities: every missed region is saved as its own fine-grained component crop on disk, AND the next step may additionally include it in a merge_group. Skipping a missed region here permanently loses its standalone fine crop. Lean toward OVER-reporting in this phase; the next phase deduplicates by merging.

DO report:
- Standalone text labels / paragraphs not in any red rectangle.
- Output / terminal callouts of a flowchart connected by an arrow  even if you think they semantically belong to the flowchart, report the bbox of just the callout text + its arrow here.
- Sub-elements of a diagram that ended up uncovered  report each uncovered sub-element separately, not as one giant union bbox.
- Any content a human would consider its own component.

DO NOT report:
- Anti-aliasing fringes, mask-cleanup residue, or stray pixels inside existing red rectangles.
- Empty page margins, decorative strips along edges.
- Orphan 1-3 character letter fragments.
- Frames / outlines with empty interiors.
- Thin connector arrows / lines on their own (only report a callout arrow if it is part of a textual / labelled callout).

Note: do NOT make merge decisions in this call. Just list missed regions at fine granularity. A separate later step will decide whether the missed regions plus existing components should be combined into larger semantic units. Trust the two-step design: report fine here, merge later.

Return STRICT JSON only, no prose, no code fences:
{{
  "missed_regions": [
    {{"bbox": [x1,y1,x2,y2], "class": "...", "description": "..."}}
  ]
}}

If nothing is missed, return ``{{"missed_regions": []}}``.
```

Prompt 8: Point-Based Missed Region Grounding

```
System:
You review a slide-decomposition pipeline's coverage. Substantive content sometimes sits outside every extracted bbox. Your job in this call is to locate those MISSED regions by giving ONE anchor POINT per missed region  SAM (a downstream segmentation model) will use that point to ground the actual mask + bbox. You DO NOT estimate bboxes yourself (VLMs are unreliable at pixel-precise bbox coordinates); you only point at where the missed content sits on the slide, and SAM does the geometric grounding.

User:
You are given:

- IMAGE 1: the original slide.
- IMAGE 2: the same slide with a red rectangle drawn over every bbox the pipeline currently extracted (combined across all iterations, after cross-iter dedup and per-component validity pruning).

For reference, here is the structured list of those existing bboxes:

{components_json}

Allowed taxonomy classes (use ONLY these exact strings for `class`):
{taxonomy_json}

Identify SUBSTANTIVE content visible OUTSIDE every red rectangle that should have been a component (e.g. a callout label, a sub-diagram block, a "Predictions" / "Output" terminal box at the end of a flowchart, a body paragraph the pipeline did not detect, a separate photograph).

For each missed region, give ONE "anchor point"  pick a pixel coordinate (x, y) that clearly sits ON the missed content (the darkest text glyph, the centre of a shape, the middle of a photograph, etc.). SAM will use this point to figure out the full mask + bbox.

DO report:
- Standalone text labels / paragraphs not in any red rectangle.
- Output / terminal callouts of a flowchart connected by an arrow.
- Sub-elements of a diagram that ended up uncovered.
- A standalone photograph / chart / illustration not in any red rectangle.
- Any content a human would consider its own component.

DO NOT report:
- Empty page margins, decorative strips along edges with no content.
- Faint background gradients with no glyphs / strokes.
- Anti-aliasing fringes around an existing red rectangle.
- A point that lands on EMPTY whitespace adjacent to an already-extracted component (e.g. do NOT point at the empty space next to a photo claiming it's the "other half" of that photo). Only point at pixels that visibly have content.
- Frames / outlines with empty interiors.
- Page-edge decorative bars that already touch an existing component.

Note: do NOT make merge decisions in this call. Report fine-grained missed regions; a later step will decide whether they should be merged with existing components.

Pick the point CAREFULLY:
- (x, y) coordinates are in ORIGINAL slide pixel space (top-left=0,0).
- Pick a pixel that is INSIDE the missed content, not on its edge.
- For text content, pick a pixel that lands on a visible glyph (not in the gap between letters).
- For photos / illustrations, pick a pixel near the visual centre.

Return STRICT JSON only, no prose, no code fences:

{{
  "missed_regions": [
    {{"point": [x, y], "class": "...", "description": "<one sentence>"}}
  ]
}}

If nothing is missed, return ``{{"missed_regions": []}}``.
```

Prompt 9: Missed Region Validation

```
System:
You QA-check ONE proposed 'missed region'
    from a slide-decomposition pipeline's
     layout reviewer. The reviewer
    claimed it found a real component
    that the pipeline's main detector
    missed. Your job is to decide whether
     the proposed bbox actually contains
    a real, useful slide component  or
    whether it's a mistake (an empty
    patch, anti-aliasing fringe, a sliver
     of background, an off-by-far
    rectangle, a duplicate of an already-
    extracted component, etc.) that
    should be discarded.

User:
You are given:

- IMAGE 1: the bbox CROP of the proposed
    missed region (what would
  appear as the new component's image if
      we accept the proposal).
- IMAGE 2: the full original SLIDE with
    the proposed bbox outlined in
  red. (Helps you judge what surrounds the
       bbox and whether the
  rectangle accurately frames a real
      component.)

Proposal details:
- proposed class: "{class_name}"
- proposed description: "{description}"
- proposed bbox xyxy: {bbox}

Decide: is this proposal a real, useful
    slide component that the
pipeline should add?

Mark valid = TRUE when:
- IMAGE 1 clearly shows recognisable
    content matching (or close to) the
  proposed class  readable text, a visible
       shape / icon / arrow,
  a photograph, a chart fragment, a
      labelled callout, etc.
- The bbox in IMAGE 2 reasonably tightly
    frames the actual content
  (small bbox padding is fine; large empty
       padding is NOT fine).
- The content is not already captured by
    an existing red rectangle in
  IMAGE 2 (i.e. it's genuinely a NEW
      component, not a duplicate).

Mark valid = FALSE when:
- IMAGE 1 is essentially blank, uniform
    background, or only contains
  anti-aliasing fringe / edge pixels.
- IMAGE 1 contains content but the bbox is
     far off  most of the
  meaningful content sits OUTSIDE the
      proposed rectangle.
- The proposed region is just empty
    whitespace adjacent to an already
    extracted component (e.g. the proposal
      claims "the other half of
  the photo" but there are no photo pixels
       there).
- The content inside is just a 1-3
    character fragment of a larger text
  block that's already extracted elsewhere.

- The content is a thin frame / divider
    line / decorative edge that
  the pipeline standard says should be
      ignored.

Return STRICT JSON only, no prose, no code
     fences:
{{"valid": true|false, "reason": "<one
    short sentence>"}}
```

Prompt 10: Perceptual Merge Group Proposal

```
System:
You review a slide-decomposition pipeline's component layout. The pipeline tends to extract sub-elements of complex structures (flowcharts, architecture diagrams, multi-panel figures) as many separate small bboxes instead of one coherent component. Your sole job in this call is to decide which existing components should be MERGED into a single semantic component.

User:
You are given:

- IMAGE 1: the original slide.
- IMAGE 2: the same slide with a red rectangle drawn over EVERY current fine-grained component the pipeline has decided on. This includes both components extracted by SAM3 across all iterations AND components newly identified as missed in the previous step. Treat all red rectangles uniformly  there is no semantic difference between them at this stage.

Here is the structured list of every red-rectangle component (use these filenames to refer to them in your output):

{components_json}

Allowed taxonomy classes (use ONLY these exact strings for `merged_class`):
{taxonomy_json}

Identify groups of 2 or more red rectangles that together cover sub-elements of ONE coherent semantic component (e.g. multiple boxes / arrows / labels of one flowchart, multiple cells of one architecture diagram, multiple icons of one icon-collection figure, a flowchart's internal blocks plus its terminal output callout label, OR two bboxes that the pipeline produced as fragments of the same underlying piece of text).

For each group provide:

- "member_filenames": the filenames of the bboxes to merge (must all appear in the components list above; can mix any combination of iter-prefixed and missed-prefixed filenames).
- "merged_class": one taxonomy class that names the merged component (e.g. "flowchart", "diagram", "architecture diagram", "image collection", "bullet list", "footer"). MUST be from the taxonomy list above.
- "merged_bbox": [x1, y1, x2, y2]  the union bbox covering all members. Use ORIGINAL slide pixel coordinates.
- "reason": one short sentence.

PRIORITY RULE  fragment-overlap merge (overrides every "do not merge" rule below):

- If two or more bboxes OVERLAP HEAVILY the smaller is mostly contained in the larger (>= ~70% of the smaller's area inside the larger), OR they share most of their area pairwise  AND they textually/visually represent the SAME content (one is a fragment of the other; they are two halves of the same text line; the smaller's OCR is a substring of the larger's OCR), they are pipeline fragmentation artefacts and MUST be merged. Heavy spatial overlap + same/substring content is a strong, deterministic signal  apply this rule even if one or more members are normally "complete units" (footer, copyright notice, title).
- Examples that MUST merge under this rule:
  - The last bullet of a list extracted both as "bullet list" (a wide bbox covering all bullets) AND as a separate "caption/label" or "bullet point text" inside that wide bbox  merge into one bullet list.
  - Two overlapping bboxes at the bottom of the slide both reading parts of the same copyright string  merge into one footer.
  - A "page number" bbox sitting inside a "footer" bbox both covering the same trailing text  merge into one footer.
  - A heading split into two adjacent overlapping bboxes  merge.

DO merge (in addition to the priority rule):
- Multiple boxes/arrows/labels that together form a single flowchart, including its terminal output callout.
- Multiple cells/tiles of a single grid figure.
- A title + sub-headings + a body that together form a single labelled panel.

DO NOT merge:
- Visually adjacent but semantically independent components that DO
```

```
  NOT overlap heavily (two separate text
      blocks side by side; bullet
  list to the left of a chart on the right
      ; standalone slide title
  above a separate diagram).
- A whole standalone footer / logo / title
     with an independent
  component on a different part of the
      slide. (But DO merge two
  overlapping fragments of the SAME footer
        this is the fragment-
  overlap priority rule.)
- A whole component with another whole
    component just because they
  share a row or column AND there is no
      spatial overlap between them.

Return STRICT JSON only, no prose, no code
     fences:
{{
  "merge_groups": [
    {{"member_filenames": [...], "
        merged_class": "...", "merged_bbox
        ": [x1,y1,x2,y2], "reason":
        "..."}}
  ]
}}

If nothing should be merged, return ``{{"
    merge_groups": []}}``.
```

Prompt 11: Merge Candidate Arbitration

```
System:
You decide whether a single candidate
    slide-component should be merged into
     an already-formed semantic merge
    group. The pipeline merged some
    components into a coherent unit; a
    non-member candidate now overlaps the
     group's region. Make a per-candidate
     yes/no merge decision.

User:
You are given:

- IMAGE 1: the original slide.
- IMAGE 2: the slide with two highlighted
    regions:
  - GREEN polygon outline = the current
      merge group's region (members
      listed below).
  - YELLOW thick rectangle = the CANDIDATE
       component being considered for
      inclusion.

Existing merge group (already a coherent
    unit):
- merged_class: "{merged_class}"
- members:
{merged_members_json}

Candidate:
- filename: {candidate_filename}
- class: {candidate_class}
- bbox (x1,y1,x2,y2): {candidate_bbox}

Decide: should the CANDIDATE be merged
    INTO this group, becoming part of
the same semantic component?

Set should_merge = TRUE when (any of):
- The candidate is the missing complement
    of the merged unit (e.g. the
  RHS of an equation whose LHS is in the
      group; a legend that explains
  chart elements in the group; a caption
      beneath a figure in the group;
  the terminal callout of a flowchart
      whose body is in the group).
- The candidate is a fragment of the same
    content as a member (e.g. a
  heading split into two adjacent
      overlapping bboxes; two halves of
      the
  same footer line).
- Removing the candidate would make the
    merged group semantically
  incomplete.

Set should_merge = FALSE when:
- The candidate is an independently
    meaningful component (a separate
  text block, a different figure, an
      unrelated label) that just happens
  to sit inside the merged group's
      bounding rectangle because the
      group
  is wide / tall / spans much of the slide.

- The candidate's content is self-
    contained and does NOT extend or
  complete the merged group.
- They are visually adjacent but
    semantically independent.

Then ALSO decide ``visually_overlaps``:
    do the actual pixels of the
candidate share screen space with any
    member's actual pixels, or do
they only share BBOX space?

Set visually_overlaps = FALSE (just bbox
    overlap, "L-shape") when:
- The group's bbox is L-shaped or wide/
    tall and visually wraps around
  the candidate, but the group members'
      real strokes/glyphs/pixels do
  not touch the candidate's strokes/glyphs
      /pixels. E.g. an L-shaped
  flowchart whose bbox encloses an
      unrelated photo in its inner corner;

  a tall bullet list whose bbox extends
      past the actual text into a
  chart on the right.
- The group's bounding rectangle just
    spans across the candidate's
  region because of one outlier member,
      but no member visually overlaps
  the candidate.

Set visually_overlaps = TRUE (real visual
    overlap) when:
- The candidate sits on top of / inside /
    behind a member's actual
  content  a logo or watermark over a
      photo, an icon embedded inside a
  diagram, a label whose glyphs are drawn
      directly over a chart's bars.
- Their actual pixels mix in the same
    screen region.

When should_merge=TRUE, you can set
    visually_overlaps to whatever is
true; it will be ignored downstream.

Return STRICT JSON only, no prose, no code
     fences:
{{"should_merge": true|false, "
    visually_overlaps": true|false, "
    reason": "<one short sentence>"}}
```

Prompt 12: Overlap Ownership Arbitration

```
System:
You arbitrate ownership of an overlap
    region between two slide components.
    Two extracted bboxes overlap on the
    slide; for each side you decide
    whether THAT side's actual content
    extends into the overlap region (true
    ) or whether the bbox just sloppily
    extends into the other component's
    territory without owning that region
    (false). Use the visual evidence on
    the slide, not just the bbox geometry.


User:
Two extracted slide components have
    overlapping bboxes. Decide who owns
    the overlap.

- IMAGE 1: original slide.
- IMAGE 2: same slide with two highlighted
     bboxes:
  - RED rectangle = component A
  - BLUE rectangle = component B
  - the overlap region is shaded YELLOW.

Component A:
- class: "{class_a}"
- bbox (x1,y1,x2,y2): {bbox_a}

Component B:
- class: "{class_b}"
- bbox (x1,y1,x2,y2): {bbox_b}

Look at IMAGE 1 inside the overlap region
    (the yellow shaded area in IMAGE 2)
and decide for each side independently:

- ``a_owns_overlap``: TRUE iff component
    A's actual visual content (its
  text glyphs, graphic strokes, photo
      pixels, etc.) is genuinely present
  inside the overlap region  i.e. that
      region is part of A. FALSE when
  A's bbox just sloppily extends into the
      overlap area while A's real
  content sits elsewhere (typical for text
       classes whose bbox is too
  wide; the empty bbox padding overlapping
       another component does NOT
  count as A owning the region).

- ``b_owns_overlap``: same question for B.


Decision matrix:
- TRUE / TRUE  both genuinely overlap (
    legitimate co-located content). No
    carve.
- TRUE / FALSE  A owns the overlap; B's
    bbox is the suspect side. Pipeline
    will carve B by A.
- FALSE / TRUE  B owns the overlap; A's
    bbox is the suspect side. Pipeline
    will carve A by B.
- FALSE / FALSE  ambiguous / unrelated;
    pipeline will leave both bboxes
    unchanged.

Common patterns:
- A is a bullet list / body text whose
    bbox extends past the actual text
    into a photograph or chart on the
    right: a_owns=false, b_owns=true
    carve A.
- A is a slide title whose bbox extends
    into a logo / icon on the right:
    a_owns=false, b_owns=true.
- A photo and a small caption beneath it
    that genuinely co-located in the
    overlap: typically the smaller bbox
    is fully inside the bigger but each
    owns its part of the overlap. Use
    TRUE/TRUE only if both clearly have
    content there.

Return STRICT JSON only, no prose, no code
     fences:
{{"a_owns_overlap": true|false, "
    b_owns_overlap": true|false, "reason":
     "<one short sentence>"}}
```

Prompt 13: Segmentation Order Planning

```
System:
You plan the SAM3 segmentation order for one slide. The pipeline will call SAM3 once per LAYER in the order you produce. Each layer is a SEPARATE SAM3 call  its detector sees ONLY the classes in that layer, and SAM3's built-in cross-class dedup only suppresses duplicates within a single call. So if you split classes that describe the same visual content into different layers, both will fire and produce duplicate bboxes. KEEP classes that target the same visual content in the SAME layer.

Each layer after the first runs on the slide with all earlier layers' mask pixels already removed. Order classes so that the ones most prone to absorbing UNRELATED neighbouring pixels are segmented LAST (after their neighbours are gone), and the ones most likely to be ABSORBED BY a wider class go EARLIER (so they are claimed before the wide class fires).

You DO NOT add, remove, or rename classes  output the exact same set, just partitioned and ordered.

User:
You are given:

- IMAGE: a presentation slide.
- CLASS LIST (every class to segment on this slide):

{class_list_json}

Partition the class list into 2 OR 3 ordered LAYERS (NOT 4 unless the slide truly has four visually distinct depth tiers  most slides need just 2). Layer 1 is segmented first on the original slide. Layer 2 runs on the cleaned image after Layer 1's masks are removed. And so on.

PRIMARY GOAL: keep classes that pick up the SAME visual region in the SAME layer, so SAM3's internal cross-class dedup can suppress duplicates. If a class is split off into a later layer where SAM3 only sees it (no competing class), SAM3 will produce a fresh bbox even when the same region was already extracted  this is by far the worst failure mode of layered seg.

SECONDARY GOAL: prevent mask leakage by ordering. When mode (a) and mode (b) both apply, pick whichever is the bigger risk for THIS slide:

  (a) A small / on-top / tightly-bordered class is at risk of being ABSORBED by a larger class's mask  put the small one EARLIER. Examples: a math symbol on top of a chart, a callout label over a photograph, brackets / single characters adjacent to a wide illustration.

  (b) A small / well-bordered class's mask is at risk of HALLUCINATING adjacent decorative pixels into itself (e.g. a "bullet list" mask grabbing the green edge of an area chart that sits beside it; a "logo" mask grabbing background gradient pixels)  put the WIDE adjacent class EARLIER (so the adjacent decoration is cleared before the small class is segmented).

CRITICAL RULES  break any of these and the layered pipeline will produce worse output than no layering at all:

R1. Generic / catch-all classes that have weak visual priors ("decorative element", "frame", "whitespace", "shape", "icon collection", "mixed content", "composite figure", "graph diagram", "diagram", "figure", "illustration", "label", "text", "annotation" when used as a generic catch-all) MUST share a layer with at least one concrete class. NEVER put any of them alone in their own (late) layer  SAM3 will hallucinate slide-spanning false-positive bboxes against the emptied image.

R2. Classes that overlap heavily in their visible region MUST be in the same layer. Examples to detect from the IMAGE:
   - Multiple bullet-related classes ("bullet list", "bullet marker", "bullet point", "bullet point text") all targeting one bullet block  same layer.
   - "Venn diagram" + "bubble chart" both targeting one set of overlapping circles  same layer.
   - "axis" + "axis label" + "axes" + "axis labels"  same layer.
   - "arrow" + "arrow annotation"  same layer.
   - "photograph" + "image" or "figure" if both describe the same photo  same layer.

R3. If after applying R1 and R2 you cannot find a clean visual depth
```

```
    split, just return ONE layer with
        every class. One layer is
    strictly better than a bad multi-layer
         split.

R4. Each layer should contain at least 2
     classes (or be the only
    layer). Single-class layers triggered
        by R1/R2 should be merged
    with the next layer.

Output schema (strict JSON, no prose, no
     code fences):

{{
  "layers": [
    {{"layer": 1, "classes": ["...",
        "..."], "reason": "<one short
        sentence rooted in what's visible
        on THIS slide>"}},
    {{"layer": 2, "classes": ["..."], "
        reason": "..."}}
  ]
}}

If you cannot confidently stratify the
     slide, return:

{{"layers": [{{"layer": 1, "classes": [<
     every input class>], "reason": "no
     clear stratification  better to let
     SAM3 dedup across all classes in a
     single call"}}]}}
```

E.2 Baseline Details

E.2.1 SlideCoder

We evaluate SlideCoder (Tang et al., 2025) as a key baseline by reconstructing PowerPoint presentations (`.pptx`) from visual references. The process begins by taking the slide images as input to the SlideCoder framework, which utilizes VLM to generate corresponding `python-pptx` scripts. The generated scripts are then programmatically executed using a dedicated driver to synthesize physical `.pptx` files. Finally, the reconstructed slides are rendered back into high-resolution images to enable downstream automatic evaluation alongside our proposed method.

E.2.2 Claude Opus 4.7

We evaluate Claude Opus 4.7 as a strong proprietary agentic baseline. Rather than using the Claude API directly, we use Claude Code to assess the capability of its built-in coding and execution harness. For each deck, a Claude Code monitor agent launches a worker agent with the prompt below. The worker agent then reasons over the slide decomposition task, reconstructs the slide state, and outputs an editable PPTX artifact.

Baseline Prompt: Claude Opus 4.7 Slide Decomposition

```
Slide Decomposition Prompt

Reconstruct slide images into editable .
    pptx files. Do NOT take the shortcut
    of embedding the source PNG full-
    bleed as a single picture.

Objective

For each input image, build a .pptx that
    is visually identical to the source
    when rendered: same layout, text
    content, fonts, colors, sizes,
    positioning, and visual elements.
    Text must be reproduced as real
    editable text boxes, not images. Use
    python-pptx text boxes, shapes, and
    connectors for typography and vector
    content. For regions you genuinely
    cannot re-vectorize, such as photos,
    intricate bitmap diagrams, logos, or
    charts, crop just that region with
    PIL and embed it as a picture in its
    correct slot.

Inputs and outputs

You will be given:

- An input directory containing one or
    more slide images, such as PNG or
    JPEG files.
- An output directory where the
    corresponding .pptx files must be
    saved.
- A path to the manifest JSON for progress
     tracking.

For each input <NAME>.png or <NAME>.jpeg,
    the corresponding output is <
    output_dir>/<NAME>.pptx. Create the
    output directory if it does not exist.


Per-image workflow

1. Read the image with the multimodal Read
     tool so you can actually see it.

2. Observe the layout: background, title,
    body text, bullets, tables, shapes,
    arrows, image regions, font choices,
    weights, alignments, approximate
    positions and sizes. Note the image's
     pixel dimensions, which determine
    the target slide canvas. Use 9144000
    x 6858000 EMU, i.e., 10" x 7.5", for
    4:3 sources by default, and use
    12192000 x 6858000 EMU for 16:9
    sources.

3. Write a namespaced Python script using
    python-pptx and PIL that reconstructs
     the slide. Follow these conventions:
   - Use Pt(), Inches(), and Emu() for
       sizes.
   - Reproduce text verbatim with matching
        font, size, weight, color, and
       alignment.
   - Recreate vector shapes, such as
       rectangles, ovals, arrows,
       connectors, and lines, with
       correct fills, lines, and
       positions.
   - For complex raster regions, crop with
        PIL and embed them as pictures at
        the correct position and size.

4. Run the script with python3.

5. Verify the output: the file must exist,
     have size greater than 1024 bytes,
    and its first four bytes must be b'PK
    \x03\x04', since every valid .pptx is
     a zip file. If invalid, retry up to
    two more times with a simpler
    approach, using fewer shapes, fewer
    embedded crops, or a simpler layout.

6. Update the manifest after each image
    with the locked snippet below.

Manifest update, parallel-safe

The manifest is shared across concurrent
    agents. Always update it under an
    exclusive file lock. Substitute <
```

```
     DECK_ID>, <NAME>, <STATUS>, <
     OUTPUT_PATH>, <ERROR_OR_EMPTY>, and <
     MANIFEST_PATH>. The value of <STATUS>
      must be success or failed.

python3 - <<'PY'
import fcntl, json, os
from pathlib import Path

M = Path("<MANIFEST_PATH>")
LOCK = M.with_suffix(".json.lock")
LOCK.touch()

with open(LOCK, "r+") as lf:
    fcntl.flock(lf, fcntl.LOCK_EX)
    try:
        m = json.loads(M.read_text()) if M.
             exists() else {"decks": {}}
        d = m["decks"].setdefault("<
             DECK_ID>", {})
        d.setdefault("slides", {})["<NAME
             >"] = {
            "status": "<STATUS>",
            "output": "<OUTPUT_PATH>",
            "error": "<ERROR_OR_EMPTY>",
        }
        tmp = M.with_suffix(".json.tmp." +
              str(os.getpid()))
        tmp.write_text(json.dumps(m,
             indent=2, sort_keys=True))
        os.replace(tmp, M)
    finally:
        fcntl.flock(lf, fcntl.LOCK_UN)
PY

When all images are processed

- Use the same locked block to set the
     deck-level entry's status to success
     if all images are successful, or
     partial if some failed.
- Set finished_at to the current ISO
     timestamp.
- Print exactly one line as the very last
     line of your reply: a compact JSON
     summary, with no preamble, postscript,
      explanations, or markdown blocks.

{"deck": "<DECK_ID>", "ok": N, "fail": M,
     "outputs": ["..."]}

Constraints

- Do not commit anything to git. Do not
     push.
- Write only to /tmp, the assigned output
     directory, and the manifest.
- Stay focused on the assigned input set.
     Do not touch other agents' work.
- python-pptx and Pillow are already
     installed. Do not pip install
     anything.
```

E.3 Prompt Used for VLM-judge in Evaluation Paradigm

This section provides the VLM-judge (Gemini-3.5-flash) prompts used to evaluate component validity and holistic state quality in slide decomposition.

Component Validity Prompt

```
System:
You are an evaluation judge for a slide-
    decomposition system. You receive one
     cropped image at a time and a label
    the system assigned to it. Rate
    strictly on two axes from 1 to 5 and
    return JSON only.

User:
The cropped image is the system's
    prediction of one slide component.
The system labelled it: "{label}".

Rate:
  cohesion (1-5): Is the crop a single
      coherent semantic unit? 5 = clean
      single unit (e.g. one chart, one
      title, one merged flowchart). 3 =
      mostly fine but with stray
      fragments. 1 = multiple unrelated
      things or noise.
  label_fit (1-5): Does the label describe
       the dominant content? 5 = perfect.
       3 = vague but not wrong. 1 = wrong.


Return JSON: {{"cohesion": int, "label_fit
    ": int, "reason": short string}}.
```

Holistic State Quality Prompt

```
System:
You are an evaluation judge for slide-
    decomposition. You will see two
    images:
1. The ORIGINAL slide annotated with
    numbered, colored bounding boxes (e.g.
     #1, #2).
2. The RECONSTRUCTION built by pasting
    each predicted component crop onto a
    blank canvas at its predicted bbox,
    annotated with matching numbered,
    colored bounding boxes.
Use these visual annotations to judge the
    decomposition's coverage, granularity,
     and redundancy. CRITICAL RULES FOR
    OVERLAPS AND FRAGMENTATION:
- Be TOLERANT of fragmentation and
    overlaps if they serve the purpose of
     "editability" in presentation
    software. For example, separating
    text from its background container
    shape, or decomposing complex
    graphics into layered nested shapes,
    is highly desirable for editing and
    should NOT be penalized.
- ONLY penalize UNNECESSARY fragmentation
    (e.g., a single continuous sentence
    broken into individual words, or a
    coherent flowchart shattered into
    meaningless unrecognizable bits) and
    UNNECESSARY redundancy (e.g.,
    completely duplicate cloned boxes
    capturing the exact same content
    multiple times, or bad layouts that
    visually occlude distinct text).
In your JSON response, always refer to
    specific components by their exact
    IDs (e.g., '#3 overlaps with #5', or
    '#1 is a merged chart title') in the
    'reason' field. Return JSON only.

User:
Image 1 is the original slide with
    numbered bounding boxes. Image 2 is
    the reconstruction canvas with
    matching numbered bounding boxes.
The predicted components list is:
{components}

Rate each axis 1-5:
  coverage (1-5): Are all meaningful slide
       elements present in the
      reconstruction? 5 = nothing
      important missing. 1 = many
      elements absent.
  granularity (1-5): Are the chosen units
      the right grain? 5 = grain is
      perfect. 1 = unnecessarily overly
      fragmented or wrongly fused. NOTE:
      Be forgiving of fragmentation if it
       separates text from shapes or
      creates editable nested graphical
      layers. Only penalize senseless
      shattering (e.g. single sentences
      broken into words, or unified
      charts broken into 50 tiny
      uneditable lines).
  non_redundancy (1-5): Does the system
      avoid duplicate components and
      erroneous overlaps? 5 = no
      erroneous redundancy. 1 = many
      duplicates or severe layout
      occlusions. NOTE: Do NOT penalize
      overlaps that represent logical
      layering (e.g. text layered over a
      background box, or nested graphics).
       Only penalize actual duplicate
      detections of the identical visual
      content or destructive occlusions.

Return JSON: {{"coverage": int, "
    granularity": int, "non_redundancy":
    int, "reason": short string (<80
    words, must cite specific visual
    component IDs like #1, #2 to justify
    your ratings)}}.
```

E.4 Slide Decomposition Qualitative Results

Figure 6 visualizes representative slide decomposition results. The red boxes show the initial SAM3-grounded component proposals, while the blue boxes show the final perception-aligned editable components produced by our pipeline. These examples illustrate how dense local grounding is converted into higher-level slide units that better match human perception and support native editing.

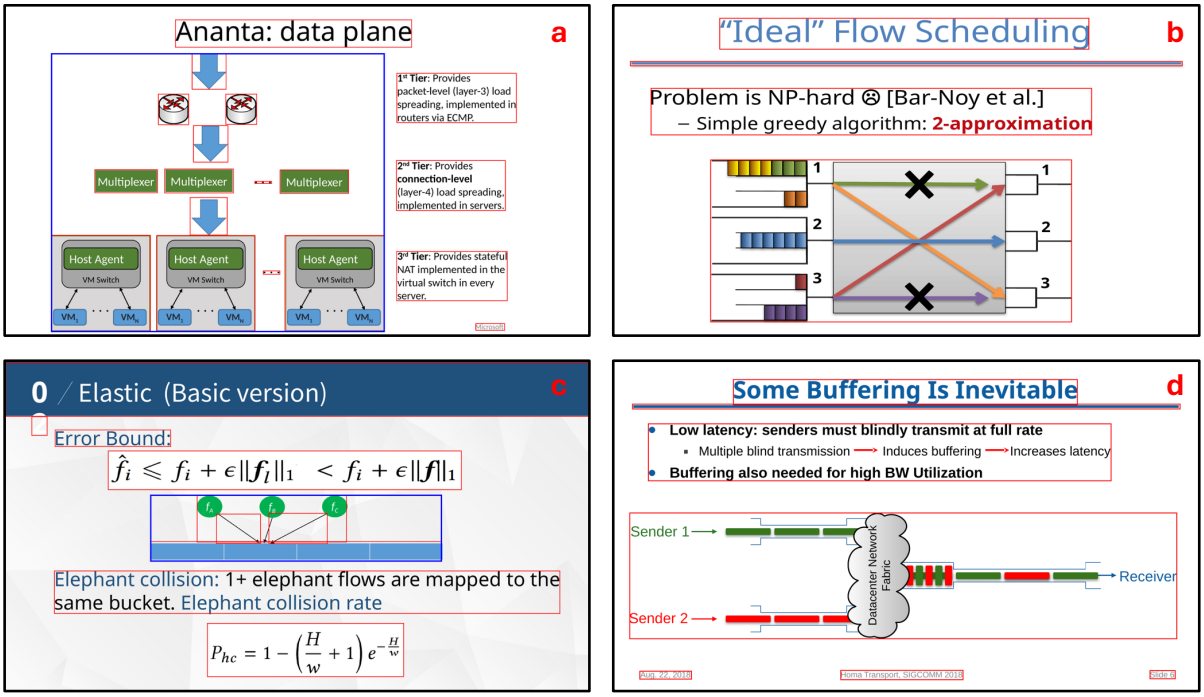


Figure 6: Qualitative results for slide decomposition. Panels (a)–(d) show representative academic slides with diverse layout structures, including hierarchical diagrams, flow-scheduling figures, equations, and network illustrations. Red boxes denote the initial component proposals grounded by SAM3. In panels (a) and (c), blue boxes highlight perception-aligned editable components produced by our decomposition pipeline after consolidating compatible local proposals. Across these examples, the raw proposals provide dense local grounding but may split a semantic slide object into fragments or include overlapping regions. Our pipeline consolidates these proposals into higher-level units that better match human perception, such as complete diagrams, equations, titles, and text blocks.

F Style Adaptation Pipeline Details

F.1 Detailed evaluation results of style adaptation

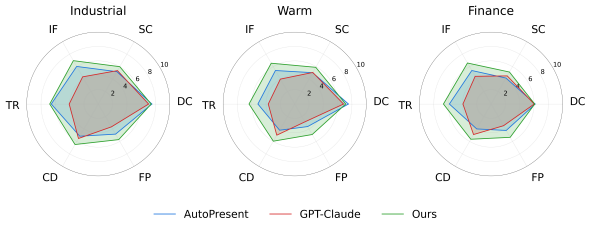


Figure 7: Six dimension VLM as a judge results for style adaptation. The abbreviations IF, SC, TR, DC, CD, and FP correspond to Information Fidelity, Slide level Coherence, Text Readability, Deck level Coherence, Content and Decoration Balance, and Fitness for Purpose.

Figure 7 summarizes the six dimension VLM as a judge scores across the three styles. SLIDEFORGE attains the largest radar area everywhere, with the clearest gains on the dimensions that demand preserving source information while changing appearance, namely information fidelity, text readability, content and decoration balance, and fitness for purpose. This supports treating restyling as structured artifact transformation rather than full slide image generation. The GPT and Claude baseline produces styled outputs but loses fidelity and readability because it regenerates each slide as an image before recovering structure, while AutoPresent reaches competitive deck level coherence yet its weaker slide level coherence shows that global consistency alone is insufficient. More detailed results are available in section More cases available in section F.1.

F.2 Another Example case of style adaptation

Figure 4 shows another successful case study on six slides under a dark industrial machinery style target. SLIDEFORGE repaints the deck with a dark metallic theme and warm orange accents while keeping every title, bullet, and diagram legible and faithful to the input, and it restyles the SoftRAN control flow and the backhaul topology as native shapes rather than flattening them into pictures. AutoPresent stamps a heavy gear photograph onto every slide, but the busy background overwhelms the text and its diagrams collapse into generic clip art that no longer conveys the original structure. The GPT and Claude pipeline reaches a plausible industrial look yet duplicates and overflows text across several slides, visible in the repeated Future Vision heading and the doubled THANKS line. The comparison shows that SLIDEFORGE adapts the visual theme while preserving the readability, diagram semantics, and layout that the baselines lose.

F.3 Limitations of style adaptation

Figure 9 illustrates these two failure modes. In the complex shape recognition example, the $\tau = 4$ shape composition modality misreads the densely packed antenna icons of the original STAR slide and rebuilds them as generic triangular marks, so the output keeps the layout but corrupts the meaning of the diagram. In the transparency example, the $\tau = 5$ raster path cannot obtain transparent assets from GPT-Image-2, so the embedded figures and the sensor board photo retain an opaque rectangular fill that sits awkwardly on the restyled background. Both cases stem from the limited capacity of the base model rather than from the harness, and stronger control over the VLMs would mitigate them.

F.4 Prompt Used in Style Adaptation Pipeline

Our style adaptation pipeline runs in six consecutive steps, step 0 through step 5. Step 0 only organises the upstream decomposition output into a uniform per-page layout and issues no model call. The five remaining steps drive a mixture of Claude Opus 4.7 prompts, used for analysis, decomposition planning, and verification, and GPT-image-2 prompts, used for styled background generation and class 5 foreground restyling. Where a single step uses both models, the Claude prompt and the GPT-image-2 prompt are shown in separate blocks below.

Step 0. Front-bg Preparation. We materialise a uniform per-page directory containing the original slide, the cleaned background, the segmented foreground components, and a metadata file with per-component bounding boxes. No language or image model is queried at this step, so no prompt is shown.

Step 1. Background Style Generation and Palette. A short Claude prompt first picks one interior slide as the content exemplar so that GPT-image-2 sees a representative middle page when it generates the three role-specific styled backgrounds. Three GPT-image-2 calls then produce the opening, content, and closing backgrounds. Finally a Claude prompt extracts a unified theme



Figure 8: Industrial Style Case study: SlideForge outperforms the baselines.

(a) Complex Shape Recognition Error

(b) Unable to Produce Transparent Image (Right 2 images)

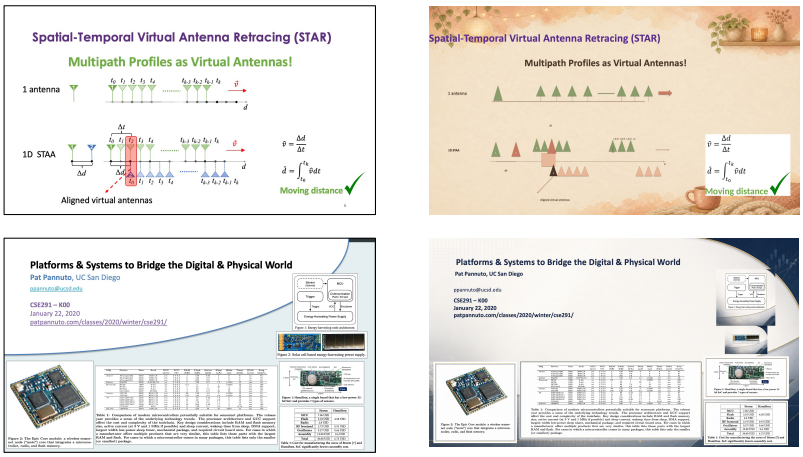


Figure 9: Failure Cases.

palette from the three styled backgrounds, which step 4 will later use as the allowed color universe.

Claude Opus 4.7 – Content-Page Picker

```
You are choosing which MIDDLE page of a
    slide deck to use as the "content"
    exemplar that will be sent to a style-
    transfer image model alongside the
    opening and closing pages.

For each candidate above you saw the
    ORIGINAL slide image and the CLEANED
    background after the upstream
    segmenter removed foreground.

Pick the candidate whose CLEANED
    background best satisfies BOTH
  a. cleanest reference with minimal
       residual foreground artifacts,
  b. most representative of a typical
       content slide for THIS deck.

Reply with STRICT JSON only.
{"pick_index": <1-N>, "reason": "<one
    short sentence>"}
```

GPT-image-2 – Opening Background

```
You are given 2 input images for the
    OPENING / TITLE page of a multi-page
    PowerPoint deck.
  Image 1, the CLEANED background of this
      page.
  Image 2, the ORIGINAL page. Use it ONLY
      to see where foreground elements
      such as title, authors, and logos
      will sit, so your generated
```

```
      decorations do NOT intrude into
      those bbox regions.

Output requirements.
- Output EXACTLY ONE full-canvas styled
    background that fills the ENTIRE
    output image, edge to edge.
- Do NOT split the output into thumbnails,
     panels, grids, stripes, side-by-side
     variations, or composites. The
    output IS the slide canvas, one slide,
     one output image.
- Decorations belong along the edges,
    corners, and margins. Leave the large
     central region visually quiet so the
     title and authors can be placed on
    top later.
- This is the opening / title page, so
    decoration can be on the more
    elaborate hero end.
- Image 1 may still contain residual logos
     or text that the upstream segmenter
    failed to remove. IGNORE them and do
    NOT preserve them in your output.
```

The CONTENT and CLOSING role prompts share the same scaffold as the opening prompt, with one extra input image set to the already generated opening so the deck inherits a consistent palette and motif vocabulary across slides while differing in composition. The CONTENT prompt additionally constrains decoration to the top and bottom edges only and demands a quiet central band, since interior slides routinely host bullets, diagrams, and charts that extend toward the side margins.

Claude Opus 4.7 – Palette Extractor

```
You are a senior visual designer analyzing
     the visual style of a slide deck.
    You have just been shown THREE slide
    background images from the SAME deck,
     ordered as opening, content, closing.


Treat the three images as ONE coherent
    visual system and derive a unified
    design palette that describes the
    deck as a whole rather than each
    image individually. Produce.

1. THEME COLOR. The single most
    representative dominant color.
    Provide an uppercase hex string and a
     short human-readable name.
2. THEME PALETTE. An ordered list of 4 to
    6 colors from MOST prominent to LEAST
     prominent. Give hex, name, and a one-
    phrase role for each entry, such as "
    background", "primary accent", or "
    supporting tone".
3. FONT PALETTE. An ordered list of
    recommended font colors to use ON TOP
     of the theme. Sort STRICTLY from
    highest contrast to lowest contrast.
    For each entry give hex, name, an
    approximate WCAG contrast ratio, and
    a one-phrase recommended use.

Respond with STRICT JSON only. No markdown
     fences and no commentary.
```

Step 2. Modality-Typed Element Classification. For each foreground component prepared in step 0, a Claude prompt assigns one of six modality classes that decides which reconstruction primitive is used downstream. The classifier is conservative on semantically meaningful elements and never sends them to the discard class.

Claude Opus 4.7 – Component Classifier

```
You are analyzing ONE foreground element
    extracted from a PowerPoint
slide.

The FIRST image is the ORIGINAL FULL SLIDE
    . Use it as context for the element's
     role within the slide.
The SECOND image is the FOREGROUND ELEMENT
     to classify, isolated from the slide
     on a transparent background.

The filename hint produced by an upstream
    detector is "<<FILENAME_HINT>>". The
    hint is informative but NOT
    guaranteed.

Choose ONE class.
  0 decorative content with no semantic
      meaning, drop.
  1 pure text or formula without a
      background fill.
  2 text or formula on a colored or shaped
       background.
  3 structured text that must remain
      editable, typically a table.
  4 shape composition approximable by
      primitive PPTX shapes.
  5 raster-only content such as photo,
      complex logo, screenshot, or
    data chart.

Be conservative. Elements that may carry
    semantic content, such as logos,
    charts, labels, arrows, annotations,
    or page numbers, must NOT be assigned
     to class 0.

Respond with STRICT JSON only.
{"class": <0-5>, "reason": "<one short
    sentence>"}
```

Step 3. Class-Conditional Reconstruction. Every class above 0 has its own reconstruction driver. Classes 1, 2, 3, and 4 are reconstructed natively as

editable PPTX shapes following a shared render-and-verify loop where a VLM proposal is rendered headlessly and the rendering is compared against the original bbox crop. Class 5 is restyled through a GPT-image-2 image edit guided by a Claude-written style prompt. We list one representative prompt per class plus the shared verifier.

Claude Opus 4.7 – Class 1 Text Extractor

```
You are looking at one foreground element
    extracted from a PowerPoint slide.

Image 1, the ORIGINAL FULL SLIDE for
    context.
Image 2, the FOREGROUND ELEMENT to encode
    as an EDITABLE TEXT BOX.

This element was classified as class 1,
    pure text or formula WITHOUT any
    background colour or shape. Reproduce
     it faithfully as a single textbox
    that fits EXACTLY inside its bounding
     box. Decide.

- the verbatim text content. Preserve
    casing, punctuation, spaces,
  and any line breaks. Use \\n to encode
     line breaks.
- the TOTAL number of visible text lines.
- the font family.
- the font size in points.
- the alignment.
- the font color drawn from the deck FONT
    palette.

Respond with STRICT JSON only.
```

Claude Opus 4.7 – Render-and-Verify Loop

```
You are verifying a PPTX textbox
    reconstruction.

Image A, the ORIGINAL foreground element
    as ground truth.
Image B, the CURRENT textbox rendered from
     the PPTX, cropped to the same region
     as the current textbox bbox.

Constraints in priority order.
  1. HARD. The rendered text must NOT
     extend beyond the textbox bbox.
  2. HARD. The rendered font size must NOT
      go below <<MIN_PT>> pt.
  3. SOFT. Aim to match the original line
     count <<ORIG_NUM_LINES>>.

If Image B violates any HARD constraint,
    propose a minimal change. Either
    extend the bbox WIDTH by no more than
     <<MAX_EXTEND_PX>> pixels inside the
    slide canvas, or shrink the font by 1
     pt, or both. If Image B already
    satisfies all HARD constraints and
    the line count is close enough,
    accept.

Respond with STRICT JSON only.
```

Claude Opus 4.7 – Class 3 Table Structure Pass

```
You are looking at the FOREGROUND TABLE
    element to encode as a NATIVE,
    EDITABLE PPTX TABLE.

THIS PASS returns ONLY the table STRUCTURE
     plus a small global STYLE ANCHOR.
    The cell content is requested in a
    separate batched call so single-call
    output never truncates. Provide.

- grid size n_rows and n_cols.
- header_row and first_col_header boolean
    flags.
- col_widths_frac and row_heights_frac,
    fractional sequences that sum to 1.
- a style anchor with default cell fill
    hex, header fill hex,
  border hex, and body text color hex
     drawn from the deck palettes.

Respond with STRICT JSON only.
```

Claude Opus 4.7

```
You are looking at a FOREGROUND ELEMENT to
     approximate as a composition of
    NATIVE, EDITABLE PPTX SHAPES.

Decompose into the SMALLEST set of
    primitive PPTX shapes that
    recognizably reproduces the element,
    at most <<MAX_SHAPES>> shapes. Use a
    fractional coordinate system rel_xyxy
     in [0, 1] inside the element
    bounding box. The build step hard-
    clamps every output shape inside the
    bbox, so any positional drift is
    automatically corrected and never
    produces overflow.

Respond with STRICT JSON only.
```

Claude Opus 4.7 – Class 5 Image-Edit Prompt Generator

```
You are preparing a single prompt that
    will be sent to GPT-image-2 to
    perform STYLE TRANSFER on a class 5
    foreground element such as a logo,
    photograph, text-and-image composite,
     or data chart.

Inputs you saw are the original full slide
    , the foreground element itself, and
    the new styled background. Produce a
```

```
    self-contained instruction that asks
    GPT-image-2 to restyle the element so
     it harmonises with the new
    background while preserving any
    identity or data-critical content.
    Logo wordmarks must remain readable,
    chart axis labels and tick values
    must remain identical, and screenshot
     text must remain identical. Do NOT
    add new icons, captions, or
    decorative elements.

Respond with STRICT JSON only, carrying a
    single field "image_prompt" whose
    value is the natural language prompt
    for GPT-image-2.
```

GPT-image-2 – Class 5 Foreground Restyle

```
<<IMAGE_PROMPT EMITTED BY CLAUDE IN THE
    PRECEDING STEP>>

Example payload for a warm-cozy chart, "
    Restyle this matplotlib line chart so
     it harmonises with the cream and
    terracotta parchment ground. Keep all
     axis ticks, axis labels, legend
    entries, line colors, and numerical
    values identical to the input chart.
    Replace only the chart background and
     axis line colors with muted amber.
    Do NOT redraw the data, do NOT add
    new labels, and do NOT introduce
    decorative props."
```

Step 4. Palette-Constrained Color Restyling. Each non-raster reconstruction from step 3 is recoloured so it harmonises with the new styled background. Geometry, font metrics, and table dimensions from step 3 remain untouched. Claude builds a functional color map under the rule that identical source hex values must map to identical target hex values, so semantic equivalence classes such as header vs body cells survive the recolor.

Claude Opus 4.7 – Table Recolor

```
You are restyling a PPTX TABLE so it
    harmonises with a NEW slide
    background.

Given.
  Image A, the new slide background.
  Image B, the table BEFORE restyling.
  THEME palette, the allowed cell
      background colours.
  FONT palette, the allowed text colours
      ranked by contrast.
  Distinct current cell-fill hex values
      with counts.
  Distinct current cell-border hex values
      with counts.

Map EACH DISTINCT current cell-fill colour
     to a NEW hex from the THEME palette,
     and EACH DISTINCT cell-border colour
     to a NEW hex from the THEME palette.

Hard rules.
- Identical source hex always maps to the
    same target hex.
- Pick fills with MODERATE contrast
    against the new background.
- Header rows must remain visually
    distinct from body rows.
- Text colors low in contrast against the
    new cell fill must be
  swapped to the highest-contrast entry
      from the FONT palette.

Respond with STRICT JSON only.
```

Claude Opus 4.7

```
You are restyling a PPTX SHAPE COMPOSITION
     so it harmonises with a NEW slide
    background.

Given the new background, the shape
    composition, the THEME and FONT
    palettes, the distinct shape-fill hex
     values, and the distinct shape-line
    hex values, map each DISTINCT fill
    colour and each DISTINCT line colour
    to a new hex from the THEME palette.

Hard rules.
- Moderate contrast against the new
    background.
- Identical source hex always maps to
    identical target hex.
- Any low-contrast text inside the shape
    must be swapped to the
  highest-contrast entry from the FONT
      palette.

Respond with STRICT JSON only.
```

Step 5. Final Assembly and Self-Critic. For each page the role-specific styled background is placed as the slide canvas, then every reconstructed component is restored on top, deep-copied from its step 3 or step 4 PPTX and anchored to the decomposition-time bounding box. A conservative Claude self-critic then reviews the assembled deck. It is restricted to two safe operations, namely block-level z-order changes and text-scale hierarchy adjustments, and it never adds, removes, or moves slide content.

Claude Opus 4.7 – Z-Order Self-Critic

```
You are reviewing the Z-ORDER of
    foreground shape blocks on ONE slide
    of a restyled PowerPoint deck.

Image A, the ORIGINAL slide BEFORE style
    transfer. Use it ONLY as ground truth
     for WHICH element should be ABOVE
    which. Image B, the CURRENT restyled
    slide as rendered.

The current slide lists foreground shape
    blocks from BACK, drawn first, to
    FRONT, drawn last. Each block is one
    logical component such as one text
    block, one image, or one shape group.
<<BLOCK_LIST>>

Only propose changes if the rendering
    shows a CLEAR occlusion or overlap
    that contradicts Image A. Never add,
    remove, or relocate content. Only
    suggest pairwise z-order swaps or
    batch moves to back or front.

Respond with STRICT JSON only.
```

F.5 Prompt Used for VLM-judge in Evaluation Paradigm

Our restyling evaluation drives a Claude Sonnet 4.6 judge once per deck. Every rendered slide of the output deck is shown to the judge in slide order, together with the original natural-language style prompt that the deck was meant to inhabit. The judge then scores six independent aspects on a 1 to 10 scale and is required to cite a specific visual element for each score. The exact system-level prompt is reproduced below. The placeholder {{STYLE_PROMPT}} is substituted with the per-deck target style description before the prompt is sent.

Claude Sonnet 4.6 – Style-Adaptation Judge

```
You are an expert design critic evaluating style-transferred presentation slides. You will be shown ONE deck of slides produced by a style-transfer approach. The deck was generated to match this TARGET STYLE.

  TARGET STYLE PROMPT. """{{STYLE_PROMPT}}"""

Evaluate the deck on the SIX independent dimensions below.

IMPORTANT RULES.
- Score each dimension INDEPENDENTLY. A deck may score high on
  some and low on others.
- For each score, cite at least ONE specific visual element you
  observed, such as "slide 3 title font" or "chart axis labels
  on slide 5". Refer to slides by the filename label shown next
  to each image, such as "slide_002.png".
- Do not let one dimension contaminate another. A deck with strong
  cross-slide consistency can still fail information fidelity, and
  vice versa.
- Judge purely on what you see. Do not assume anything about which
  method produced the deck.

DIMENSIONS, score each 1 to 10.

1. DECK-LEVEL COHERENCE, cross-slide consistency.
   Looking ACROSS all slides in the deck, do they share one consistent
   visual identity, namely the same palette, typography, motif,
   framing, and style treatment.
   Penalize slides that look like they belong to different decks, abrupt
   shifts in treatment between slides, or any slide left visually
   out of step with the rest. This measures slide-to-slide consistency
   , NOT how well any single slide matches the theme.

2. SLIDE-LEVEL COHERENCE, element-to-theme alignment within a slide.
   Looking WITHIN each slide, does EVERY element, background, icons, charts
   , diagrams, text styling, and decorative shapes, conform to the
   TARGET STYLE PROMPT above.
   Penalize any element that breaks the theme, such as a default chart
   pasted onto a themed slide, generic clipart inconsistent with
   the target aesthetic, raw original assets that the style transfer
   never reached, or decorative elements that contradict the
   prompt. This measures element-to-theme fidelity within each slide,
   NOT consistency between slides.
   Average your per-slide judgments.

3. INFORMATION FIDELITY.
   Are the original information-bearing elements, including chart data,
   axis labels, diagram structure, and technical figures, preserved
   accurately. Penalize hallucinated text, garbled labels, redrawn
   charts whose numbers no longer match, or diagrams that lost
   structural meaning.

4. TEXT READABILITY.
   Can all body text, labels, captions, and annotations be read
   effortlessly. Penalize low contrast, decorative fonts over
   busy backgrounds, text overlapping imagery, or labels too small for
   their container.

5. CONTENT-DECORATION BALANCE.
   Does decoration support content, or compete with it. Penalize both
   extremes, decoration so dominant it crowds the message, AND
   decoration so sparse the deck feels like a generic template with
   a sticker.

6. FITNESS FOR PURPOSE, academic and technical talk.
   Would this deck function in its likely use case, a research talk or
   technical presentation where the audience needs to extract precise
   information.

OUTPUT FORMAT. Strict JSON only, no prose outside the JSON block.

{
  "deck": {
    "deck_level_coherence": {"score": <1-10>, "evidence": "<specific
```

```
        observation>"},
    "slide_level_coherence": {"score":
        <1-10>, "evidence": "<specific
        observation>"},
    "information_fidelity": {"score":
        <1-10>, "evidence": "<specific
        observation>"},
    "text_readability": {"score": <1-10>,
        "evidence": "<specific observation
        >"},
    "content_decoration_balance": {"score":
         <1-10>, "evidence": "<specific
        observation>"},
    "fitness_for_purpose": {"score":
        <1-10>, "evidence": "<specific
        observation>"}
  },
  "summary": "<2-3 sentences describing
      the deck's overall strengths and
      weaknesses against the target style
      >"
}
```

G Dataset Documentation and Ethical Considerations

G.1 Data Scope and Coverage

We document that our artifacts consist of academic presentation slides, with SAM3 training data collected from AI conference presentations and the out-of-domain test set collected from system-conference presentations. The slides are primarily English and contain typical slide-level language, including titles, bullet points, captions, chart labels, legends, equations, tables, and short explanatory text. The dataset is not intended to represent demographic groups, and we do not collect or annotate demographic attributes. Personal names, when present, correspond to public academic attribution or cited researchers. These characteristics define the intended scope and limitations of the dataset.

G.2 Privacy and Identifiability

The collected academic slides may include public attribution information, such as author names, presenter names, affiliations, and cited researchers. We retain such information only when it is part of publicly available academic material and do not use it for person-level identification or profiling. We reviewed the slides for private identifiers, uniquely identifying personal details, and offensive content. Slides containing private contact information, sensitive identifiers, or offensive content were removed or anonymized. Our released annotations focus on slide structure and visual components, including bounding boxes and semantic labels, rather than personal information.

G.3 Content Safety

We did not conduct a systematic screening for offensive content; because the corpus is drawn exclusively from peer-reviewed academic conference material (both the AI/ML training set and the systems/networking OOD test set), we consider the likelihood of offensive content to be negligible.

H LLM Use

We used LLMs for writing polishing and implementation support. LLMs were also used within the proposed pipeline for prompt-based component labeling and data annotation. All LLM-generated or LLM-assisted outputs were manually reviewed by the authors before use.

I Human Evaluation Interface and Instructions

Before beginning the evaluation, participants were asked to provide basic demographic and background information, including age, gender, highest education level, field of study or work, English reading proficiency, frequency of making presentation slides, and familiarity with AI-based image or slide generation tools. Name was used to save and resume study progress, while email was optional. Participants were also asked to consent to the use of their anonymized responses and demographic information for academic research.

The following instruction text was presented verbatim to participants:

Slide Quality — Human Evaluation Study

```
Thank you for participating! You will look
     at presentation slides and rate
their quality. The study has three parts:
    Part 1 --- rate restyled slide
decks (20 sets, the first 10 are required)
    . Part 2 --- rate how well slides
were split into editable components (20
    slides, first 10 required). Part 3
--- judge single slides: human-made or AI-
    rebuilt? (20 slides, first 10
required). The required items take about
    45 minutes; anything beyond them is
an optional bonus --- every extra answer
    helps us. Your progress is saved
after every question: you can close the
    page anytime and continue later by
entering the same name. There are no right
     or wrong answers --- we want your
honest visual judgment. Please work on a
    reasonably large screen (laptop or
monitor, not a phone). You can click any
    image to view it full-screen.
```