PSDesigner: Automated Graphic Design with a Human-Like Creative Workflow

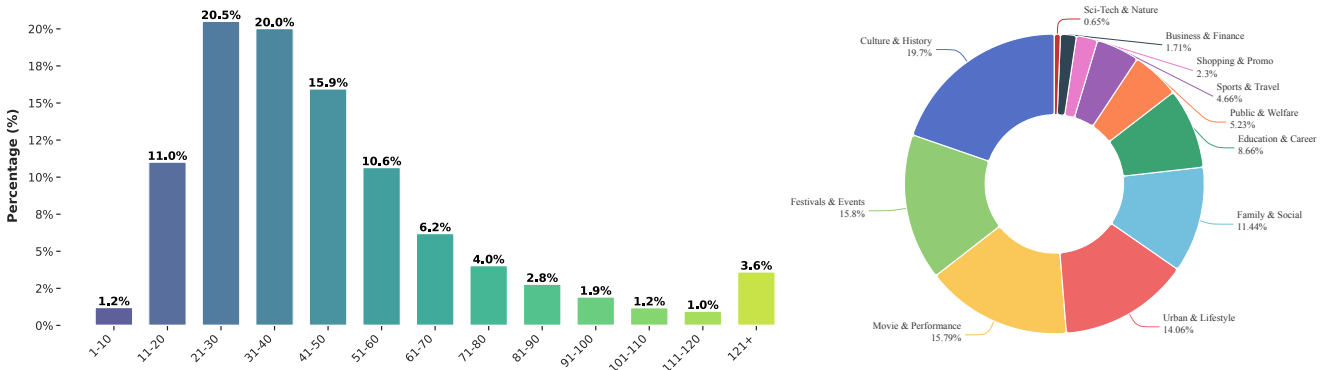
Xincheng Shuai[1*] Song Tang[1*] Yutong Huang[1] Henghui Ding[1✉] Dacheng Tao[2]
[1]Institute of Big Data, College of Computer Science and Artificial Intelligence, Fudan University, China
[2]Generative AI Lab, College of Computing and Data Science, Nanyang Technological University, Singapore
henghui.ding@gmail.com dacheng.tao@gmail.com
https://henghuiding.com/PSDesigner/

A. More Details of CreativePSD

A.1. Dataset Statistics

Fig. I (a) illustrates the distribution of layer counts across our CreativePSD dataset. The x-axis represents different layer count intervals (such as 1–10, 11–20, 21–30, *etc*.), while the y-axis shows the percentage of data samples that fall within each interval. In total, the average layer count is 48.35. The distribution of layer counts in CreativePSD reflects the complexity of professional designs created by human experts.

Fig. I (b) shows the distribution of the design scenarios in CreativePSD. As shown in the figure, the proposed dataset covers a wide range of common scenarios, making it an effective resource for training models to accomplish designs for various purposes.



(a) Distribution of Layer Counts **(b) Distribution of Design Scenarios**

Figure I. Dataset statistics. (a) Distribution of layer counts demonstrates that our CreativePSD contains most of the PSD files with complicated layer hierarchies, reflecting the complexity of professional designs created by human experts. (b) Distribution of design scenarios indicates that our CreativePSD covers a wide range of common scenarios, making it an effective resource for handling various design purposes.

A.2. Layer and Attribute Types

Tab. I enumerates primary layer types used in the CreativePSD, showing their typical usages. Tab. II discusses the basic information of the PSD file and the attribute set of the layer. Specifically, we divide the layer attributes into the basic set that applies to all layer types, along with the attributes for each specific type. All of the above information is recorded in the metadata, which is discussed in Sec. 3.1 of the main paper. Due to the complexity of the "effect" attribute, we provide more

*Equal contribution.
✉ Corresponding author (henghui.ding@gmail.com).

details in Tab. III. As shown in the table, CreativePSD includes a variety of layer effects, which enhance the visual appeal of the design.

Table I. The layer types used in CreativePSD.

Layer Type	Description
Adjustment Layer	Adjustment layers apply modifications (such as brightness, contrast, hue/saturation, etc.) to lower layers in a non-destructive way, i.e., without changing the original pixel data. They are commonly used for fine control over image appearance and are a standard mechanism for parametric edits in Photoshop workflows.
Smart Object Layer	Smart object layers encapsulate raster or vector data, enabling non-destructive transformations and filters. The content of a Smart Object retains its original characteristics, allowing users to scale, skew, or apply other operations repeatedly without quality degradation. This is widely used for compositing and flexible editing in complex documents.
Text Layer	Type layers store editable text. The textual content, style, font, and related properties can be modified at any time, making them essential for designs involving text. Edits to a type layer do not deform raster pixels until the layer is rasterized.
Group Layer	Group layers (layer groups) are used to organize multiple layers into hierarchical folders, improving file structure management and enabling group operations such as moving or transforming several layers at once. They do not contain image data themselves but serve as containers.
Pixel Layer	Pixel layers contain bitmap (raster) image data and allow for direct manipulation of pixels. They are fundamental to most image editing in Photoshop, and typical operations include painting, erasing, and filtering at the pixel level.

Table II. The basic information of the PSD file and the attribute set of each layer type.

Attribute	Description
1. Basic Information of the PSD File	
`filename`	The name of the PSD file.
`height`	The height of the image in pixels.
`width`	The width of the image in pixels.
`resolution`	The resolution of the image (DPI).
`colorMode`	The color mode of the image (RGB/CMYK/Grayscale/Lab).
`fill_color`	The fill color of the image.
`fill_color.red`	The red component of the fill color (0-255).
`fill_color.green`	The green component of the fill color (0-255).
`fill_color.blue`	The blue component of the fill color (0-255).
2. The Basic Attribute Set	
`kind`	The type of the layer (pixel/smart_object/group/text/adjustment, etc.).
`name`	The name of the layer.
`layer_id`	The ID of the layer.
`visible`	Whether the layer is visible.
`opacity`	The opacity of the layer (0-255).
`blend_mode`	The blending mode of the layer.
`left`	The left boundary of the layer.
`top`	The top boundary of the layer.
`right`	The right boundary of the layer.
`bottom`	The bottom boundary of the layer.
`width`	The width of the layer in pixels.
`height`	The height of the layer in pixels.

has_mask	Whether the layer has a mask.
mask	The array of mask contour points.
has_clip_layers	Whether the layer has clipping layers.
clip_layers	The array of clipping layer IDs.
clipped_to	The ID of the layer this layer is clipped to.
has_effects	Whether the layer has effects.
effects	The array of effects applied to the layer.
order	The order within the current group (not applicable to group layers).
3. Attribute Set of the Group Layer	
children	The array of child layers within the group.
group_order	The order of the group layers (only for group layers).
4. Attribute Set of the Smart Object Layer	
file_path	The relative file path to the smart object source file.
5. Attribute Set of the Pixel Layer	
file_path	The file path to external pixel data if stored separately.
6. Attribute Set of the Text Layer	
text	The text content of the layer.
multiline	Whether the text is multiline.
font_size	The font size of the text in pixels.
postscript_font_name	The PostScript font name used for the text.
text_color	The color of the text.
text_color.red	The red component of the text color (0-255).
text_color.green	The green component of the text color (0-255).
text_color.blue	The blue component of the text color (0-255).
bounds	The bounding box of the text.
bounds.top	The top boundary of the text.
bounds.left	The left boundary of the text.
bounds.bottom	The bottom boundary of the text.
bounds.right	The right boundary of the text.
leading	The leading (line spacing) in points.
tracking	The tracking (spacing between characters).
horizontal_scale	The horizontal scale of the text.
vertical_scale	The vertical scale of the text.
text_orientation	The orientation of the text (Horizontal/Vertical).
7. Attribute Set of the Adjustment Layer	
adjustment_type	The type of adjustment layer.
adjustment_data	The parameters of the adjustment layer, depending on the type.
Adjustment Layer Types and Parameters	
SolidColorFill	
red_fill	The red fill value (0-255).
green_fill	The green fill value (0-255).
blue_fill	The blue fill value (0-255).
PatternFill	
pattern_name	The name of the pattern.
angle	The angle of the pattern.
scale	The scale of the pattern.
GradientFill	
type	The type of the gradient fill.
gradient_parameters	The gradient color.
angle	The angle of the gradient.
ColorLookup	
lookup_type	The type of the color lookup.

`dither`	Whether dithering is enabled.
BrightnessContrast	
`brightness`	The brightness adjustment value.
`contrast`	The contrast adjustment value.
`use_legacy`	Whether the legacy algorithm is used.
Curves	
`rgb_curve`	The RGB curve points array `[(x, y), ...]`.
Exposure	
`exposure`	The exposure adjustment value.
`offset`	The offset adjustment value.
`gamma`	The gamma adjustment value.
Levels	
`channel`	The channel being adjusted (RGB).
`input_floor`	The minimum input value.
`input_ceiling`	The maximum input value.
`output_floor`	The minimum output value.
`output_ceiling`	The maximum output value.
`gamma`	The gamma adjustment value.
Vibrance	
`vibrance`	The vibrance adjustment value.
`saturation`	The saturation adjustment value.
HueSaturation	
`enable_colorization`	Whether colorization is enabled.
`colorize_hue`	The hue value when colorization is enabled.
`colorize_saturation`	The saturation value when colorization is enabled.
`colorize_lightness`	The lightness value when colorization is enabled.
`master_hue`	The master hue value when colorization is not enabled.
`master_saturation`	The master saturation value when colorization is not enabled.
`master_lightness`	The master lightness value when colorization is not enabled.
`red_hue`	The red hue value when colorization is not enabled.
`red_saturation`	The red saturation value when colorization is not enabled.
`red_lightness`	The red lightness value when colorization is not enabled.
`yellow_hue`	The yellow hue value when colorization is not enabled.
`yellow_saturation`	The yellow saturation value when colorization is not enabled.
`yellow_lightness`	The yellow lightness value when colorization is not enabled.
`green_hue`	The green hue value when colorization is not enabled.
`green_saturation`	The green saturation value when colorization is not enabled.
`green_lightness`	The green lightness value when colorization is not enabled.
`blue_hue`	The blue hue value when colorization is not enabled.
`blue_saturation`	The blue saturation value when colorization is not enabled.
`blue_lightness`	The blue lightness value when colorization is not enabled.
`purple_hue`	The purple hue value when colorization is not enabled.
`purple_saturation`	The purple saturation value when colorization is not enabled.
`purple_lightness`	The purple lightness value when colorization is not enabled.
`pink_hue`	The pink hue value when colorization is not enabled.
`pink_saturation`	The pink saturation value when colorization is not enabled.
`pink_lightness`	The pink lightness value when colorization is not enabled.
ColorBalance	
`shadows`	The shadow adjustment values `[cyan-red, magenta-green, yellow-blue]`.
`midtones`	The midtone adjustment values `[cyan-red, magenta-green, yellow-blue]`.
`highlights`	The highlight adjustment values `[cyan-red, magenta-green, yellow-blue]`.
`luminosity`	Whether luminosity is preserved during adjustment.

BlackAndWhite	
reds	The red adjustment value.
yellows	The yellow adjustment value.
greens	The green adjustment value.
cyans	The cyan adjustment value.
blues	The blue adjustment value.
magentas	The magenta adjustment value.
use_tint	Whether a tint is used.
tint_color	The tint color in the form of {red, green, blue} (if using tint).
PhotoFilter	
L	The L value in the Lab color space.
a	The a value in the Lab color space.
b	The b value in the Lab color space.
density	The density of the photo filter.
luminosity	Whether luminosity is preserved.
ChannelMixer	
monochrome	Whether monochrome is enabled.
red	The red channel mix values.
green	The green channel mix values.
blue	The blue channel mix values.
constants	The constants for the channel mixer.
Invert	No parameters.
Posterize	
posterize	The number of levels for posterization.
Threshold	
threshold	The threshold value for the adjustment.
SelectiveColor	
method	The adjustment method (relative/absolute).
reds	The red adjustment values.
yellows	The yellow adjustment values.
greens	The green adjustment values.
cyans	The cyan adjustment values.
blues	The blue adjustment values.
magentas	The magenta adjustment values.
whites	The white adjustment values.
neutrals	The neutral adjustment values.
blacks	The black adjustment values.
GradientMap	
dither	Whether dithering is enabled for the gradient map.
reverse	Whether the gradient direction is reversed.
gradient_parameters	The gradient color of the gradient map.

Table III. Details of layer effects.

Attribute	Description
Effects	
effect_type	The type of effect applied to the layer (*e.g.*, DropShadow, InnerShadow, etc.).
parameters	Parameters specific to the effect type, detailed below.
Effect Types and Parameters	
DropShadow	
blend_mode	The blending mode of the shadow.

opacity	The opacity/transparency of the shadow.
angle	The angle of the shadow.
distance	Distance from the object.
spread	The spread of the shadow.
size	The size of the shadow.
contour	The contour curve for the shadow.
anti_aliased	Whether anti-aliasing is enabled for smoothness.
noise	Amount of noise applied to the shadow.
layer_knocks_out	Whether the shadow knocks out the layer color.
color	The color of the drop shadow.
use_global_light	Whether the global light is used.
InnerShadow	
blend_mode	The blending mode of the inner shadow.
opacity	The opacity of the inner shadow.
angle	The angle of the inner shadow.
use_global_light	Whether the global light is used.
distance	Distance from the object for the inner shadow.
choke	Spread amount for the inner shadow.
size	The size of the inner shadow.
contour	The contour curve of the inner shadow.
noise	Amount of noise for the inner shadow.
anti_aliased	Whether anti-aliasing is enabled for the inner shadow.
color	The color of the inner shadow.
OuterGlow	
blend_mode	The blending mode of the outer glow.
opacity	The opacity of the outer glow.
noise	Amount of noise for the outer glow.
technique	The glow rendering technique (*e.g.*, softer, precise).
spread	Spread of the glow.
size	Size of the glow.
contour	Contour curve for the glow.
anti_aliased	Whether anti-aliasing is enabled for the glow.
range	Range of the glow (how far the glow extends).
jitter	Jitter amount for the glow.
color	The color of the outer glow when gradient_parameters is not enabled.
gradient_parameters	The gradient color of the outer glow when color is not enabled.
InnerGlow	
blend_mode	The blending mode of the inner glow.
opacity	The opacity of the inner glow.
noise	Amount of noise for the inner glow.
technique	The glow rendering technique.
source	Source of the glow (edge/center).
choke	Spread amount for the inner glow.
size	Size of the inner glow.
contour	Contour curve for the inner glow.
anti_aliased	Whether anti-aliasing is enabled for the inner glow.
range	The range of the inner glow effect.
jitter	Jitter amount for the inner glow.
color	The color of the inner glow when gradient_parameters is not enabled.
gradient_parameters	The gradient color of the inner glow when color is not enabled.
ColorOverlay	
blend_mode	The blending mode of the color overlay.
opacity	The opacity of the color overlay.

color	The overlay color.
GradientOverlay	
blend_mode	The blending mode of the gradient overlay.
dither	Whether dithering is enabled for smoother gradients.
opacity	The opacity of the gradient overlay.
reverse	Whether the gradient direction is reversed.
style	The gradient style (*e.g.*, linear, radial).
angle	The angle of the gradient.
scale	The scale/size of the gradient.
method	The method for gradient overlay.
align_with_layer	Whether the effect is aligned with the layer.
gradient_parameters	The gradient color.
PatternOverlay	
blend_mode	The blending mode of the pattern overlay.
opacity	The opacity of the pattern overlay.
pattern_name	The name of the pattern.
angle	The angle of the pattern.
scale	The scale of the pattern.
link_with_layer	Whether the effect is aligned with the layer.
Satin	
blend_mode	The blending mode of the satin effect.
opacity	The opacity of the satin effect.
color	The color of the satin effect.
angle	The angle of the satin.
distance	Distance for the satin effect.
size	Size of the satin effect.
contour	Contour curve for the satin effect.
anti_aliased	Whether anti-aliasing is enabled for satin.
inverted	Whether the satin effect is inverted.
BevelEmboss	
style	Bevel/emboss style (*e.g.*, inner bevel, outer bevel).
type	Bevel/emboss type.
depth	Depth of the bevel.
direction	Direction of the bevel (up/down).
size	Size of the bevel.
soften	Edge softness of the bevel.
angle	Light angle for the bevel.
use_global_light	Whether to use global light settings.
altitude	Light source altitude for the bevel.
gloss_contour	Contour curve for the bevel.
highlight_mode	Blending mode of the highlight.
highlight_color	Color of the highlight.
highlight_opacity	Opacity of the highlight.
shadow_mode	Blending mode for the shadow.
shadow_color	Color of the shadow.
shadow_opacity	Opacity of the shadow.
anti_aliased	Whether anti-aliasing is enabled.
contour_parameters	The parameters of the contour.
texture_parameters	The parameters of the texture.
Stroke	
size	Stroke width.
position	Position of the stroke (inside, outside, center).
blend_mode	Blending mode for the stroke.

`opacity`	Opacity of the stroke.
`overprint`	Whether the stroke overprints other elements.
`fill_type`	Stroke fill type (*e.g.*, solid, gradient).
`fill_parameters`	The configuration of the stroke.

A.3. Metadata and Tool Calls

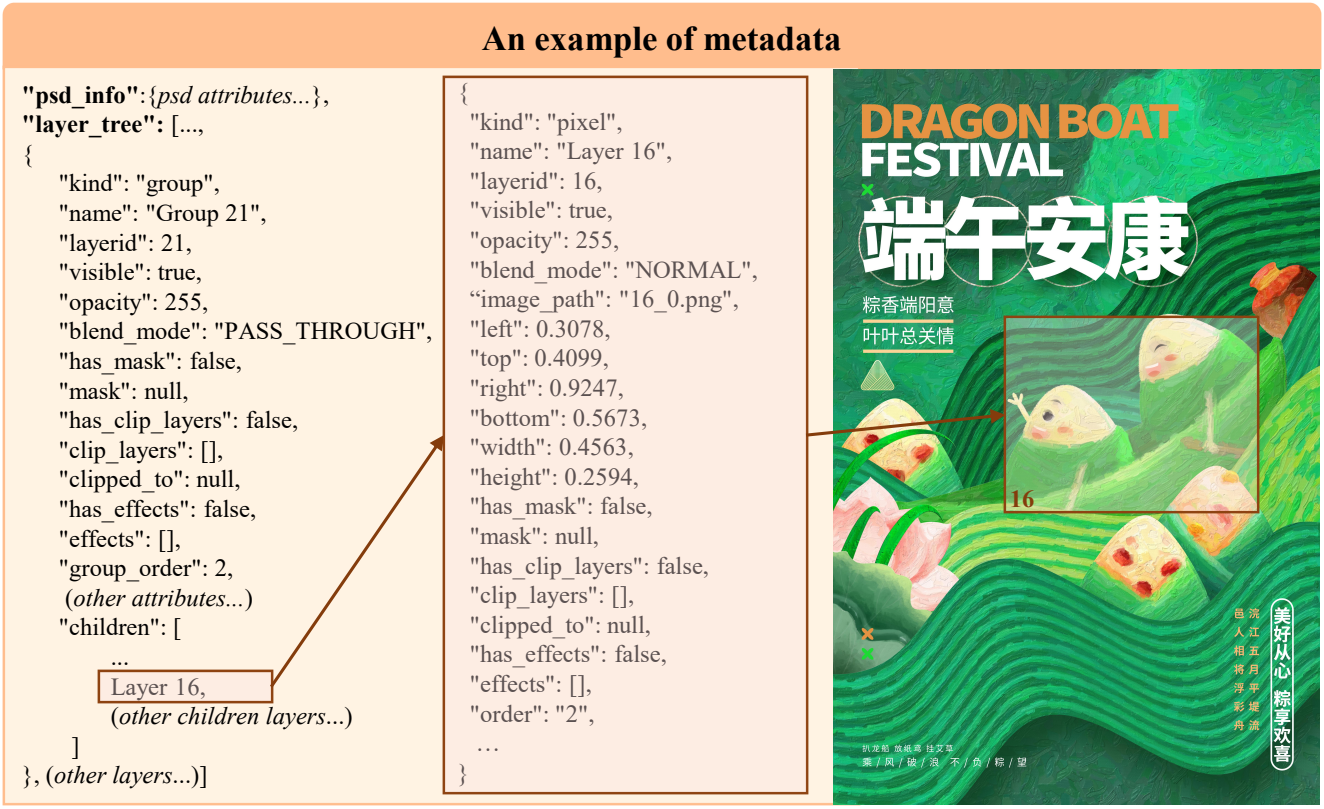


Figure II. The format of the metadata extracted in CreativePSD.

Fig. II presents a typical example of the metadata extracted from Stage II of the dataset pipeline (Sec. 3.1), where the global information is recorded in the "**psd_info**" field, and the layer hierarchy along with the attributes of each layer are recorded in the "**layer_tree**" field. These factors are detailed in Sec. A.2. Then, **Algorithm** 1 illustrates the pseudocode for converting the metadata into tool calls. Tab. IV introduces a comprehensive summary of tool calls constructed in CreativePSD.

The tool calls cover a wide range of common operations in Adobe Photoshop, including layer creation and manipulation, and file IO, *etc*., which are beneficial for models to obtain powerful tool-use capabilities and achieve diverse design purposes. Specifically, the conversion process consists of the following steps.

1. First, we initialize the PSD file by generating tool calls that configure its basic parameters, including width, height, and resolution, *etc*., establishing the foundational canvas for the design.
2. Then, we need to construct the tool calls for integrating and configuring the layers. As described in Sec 3.1 of the main paper, since all isolated layers have been grouped, we are ready to process the group layers in a bottom-up manner using **Algorithm** 2. Specifically, for each group layer: **1).** We first construct tool calls to create the group layer. **2).** Then, we recursively traverse its child layers in a bottom-up manner, dynamically constructing tool calls based on each layer's type (*e.g.*, text, pixel layers). In particular, these tool calls will create the corresponding layer while configuring both basic attributes (such as effects) and layer-specific attributes. **3).** Finally, we generate tool calls to finalize the current group.
3. Finally, we generate the saving tool calls to export the fully constructed PSD file to the user's local file system, enabling persistent storage and subsequent access.

Algorithm 1: The main function of converting the metadata to tool calls.

```
Input: metadata
Output: tool_calls
Function generate_tool_calls(metadata):
    psd_info,layer_tree ← metadata["psd_info"], metadata["layer_tree"];
    tool_calls ← [];
    /* 1. Construct tool calls for initializing the PSD file. */
    tool_calls.append(init_document(psd_info));
    /* 2. Traverse group layers in a bottom-up manner. */
    for group_layer in reversed(layer_tree) do
        /* 2.1 Construct tool calls for the group layer and its children. */
        group_tool_calls ← process_group(group_layer, psd_info);
        tool_calls.extend(group_tool_calls);
    end
    /* 3. Construct tool calls for saving the PSD file. */
    tool_calls.append(finish_document());
    return tool_calls;
```

For example, the algorithm uses `create_smart_object_layer` function to construct tool calls for the smart object layer. Based on the metadata of the layer, this function creates a sequence of tool calls, such as: `insert_image` → `adjust_current_image` → `add_drop_shadow`.

For $\mathcal{X}_{\text{edt}}$, we construct the corresponding tool calls from the Tab. IV to restore the modified attributes in the distorted metadata to their original values.

B. More Details of AssetCollector

AssetCollector is designed to collect and organize the theme-related assets based on the user instructions. Fig. III illustrates the pipeline, which consists of the following steps.

Visual Concept Extraction. Given a user instruction, AssetCollector employs a powerful LLM to analyze the user intention and identify essential visual concepts, using a well-designed query prompt indicated in Fig. XV. Specifically, LLM is required to identify all primary assets for each concept and determine their order, resulting in a JSON-format hierarchical structure. Fig. III illustrates an example. The predicted fields for each asset are:

- "**group_order**" field specifies the rendering order of the visual concept.
- "**group_name**" field provides a simple description for the visual concept.
- "**asset_order**" field determines the order of assets within each visual concept.
- "**type**" field specifies the asset type, which is "image" or "text".
- "**prompt**" field contains detailed image generation prompts used to instruct the image generation models [1, 2] (only for type="image").
- "**query**" field can be used to retrieve the image asset from the internet or database (only for type="image").
- "**text_content**" field represents the text content (only for type="text").
- "**need_artistic_prompt**" field indicates whether text needs artistic styling (only for type="text").
- "**artistic_prompt**" field is used to instruct the image generation models to generate artistic text images.

Asset Collection. Based on the response from LLM, the text-typed asset can be directly derived from the "**text_content**" field. However, for assets where the "**need_artistic_prompt**" field is set to true, AssetCollector leverages text-to-image generation models to generate stylized text images based on the recorded "**artistic_prompt**". For image-typed assets, AssetCollector collects them by retrieving from the internet or database, or using the text-to-image generation models [2, 3]. Specifically, we acquire multiple candidates for each image. Next, a pretrained VLM is instructed to assess the aesthetic quality and the alignment of the image and the "**query**" or "**prompt**" fields. Finally, we select the candidate with the highest score as the final image for each asset.

Algorithm 2: Construct tool calls for the group layer and its children.

```
Input: group_layer, psd_info
Output: group_tool_calls
Function process_group(group_layer, psd_info):
    group_tool_calls ← [];
    group_id ← group_layer["layer_id"]);
    /* 1. Construct tool calls for creating the group layer. */
    group_tool_calls.append(create_group_layer(group_layer, psd_info, group_id));
    /* 2. Traverse child layers in a bottom-up manner. */
    for layer in reversed(group_layer["children"]) do
        switch layer["kind"] do
            case "text" do
                group_tool_calls.append(create_text_layer(layer, psd_info, group_id));
            end
            case "smart_object" do
                group_tool_calls.append(create_smart_object_layer(layer, psd_info, group_id));
            end
            case "pixel" do
                group_tool_calls.append(create_pixel_layer(layer, psd_info, group_id));
            end
            case "adjustment" do
                group_tool_calls.append(create_adjustment_layer(layer, psd_info, group_id));
            end
            case "group" do
                group_tool_calls.extend(process_group(layer, psd_info));
            end
        end
    end
    /* 3. Construct tool calls for finishing the current group. */
    group_tool_calls.append(close_group_layer(group_id));
    return group_tool_calls;
```

Asset Post-Processing. To enable seamless integration of the asset into the design, AssetCollector employs a pretrained segmentation model [4] to segment the subject, thereby eliminating the influence of irrelevant background content. Users can also add/remove/adjust assets for their specific purposes.

C. More Details of GraphicPlanner

More Details of the Layer Information M. As one of the inputs of $\mathcal{X}_{\text{gen}}$ and $\mathcal{X}_{\text{edt}}$ modes, the layer information M encapsulates fundamental attributes of layers already integrated within the current group, thereby enabling the model to predict appropriate tool calls based on the current state. An example of M is illustrated in Fig. IV, which captures essential information of each layer, including bounding box, and opacity, *etc*.

More Details of the Input Formats. For $\mathcal{X}_{\text{gen}}$ and $\mathcal{X}_{\text{edt}}$ modes, we organize the inputs according to the formats shown in the Fig. V and Fig. VI, which are then fed to GraphicPlanner. Specifically, as illustrated in Fig. V, we use distinct templates for different layer types to indicate the asset information in the context.

More Details of the Output Format in $\mathcal{X}_{\text{gen}}$ Mode. We further demonstrate the output format of GraphicPlanner in $\mathcal{X}_{\text{gen}}$ mode. Unlike "smart object", "text", and "pixel" layer types in Tab. I, the "adjustment" layers do not have associated assets, which are typically used to enhance the aesthetic quality of the design. To enable the model to appropriately add such layers, we append specialized tokens to the end of the tool calls, such as "⟨need_adjustment⟩", indicating that the adjustment layer

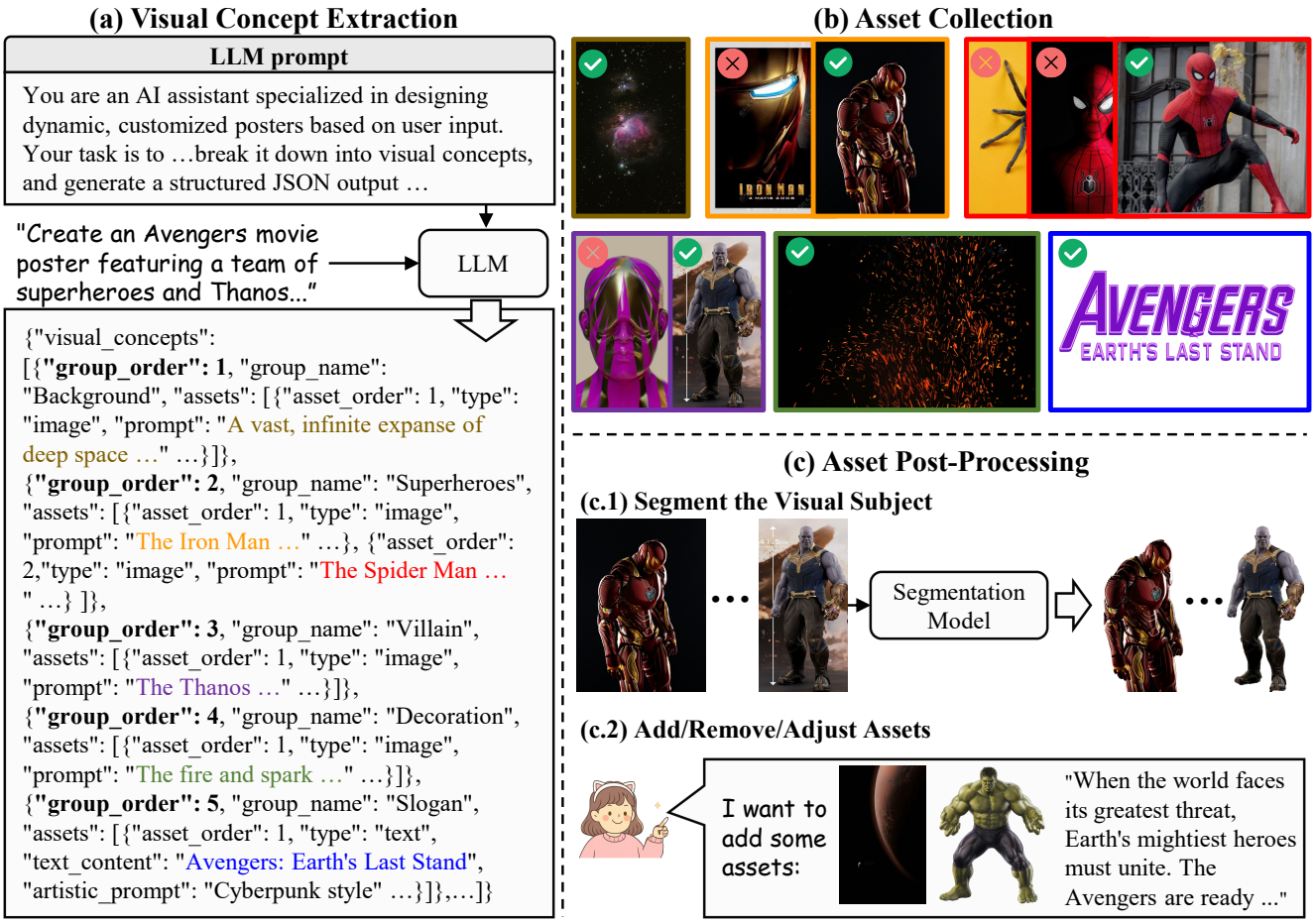


Figure III. The pipeline of AssetCollector.

should be added before the next asset is incorporated. Therefore, when GraphicPlanner outputs the specialized tokens during inference, it will construct the corresponding layer template in the input, as shown in Fig. V, while predicting the tool calls to integrate the adjustment layer in the next iteration, rather than processing the next asset.

More Details of the Reward Function. For the reinforcement learning stage of GraphicPlanner, the used reward function r is expressed as:

$$r = \frac{1}{N_{\text{param}}} \sum_{i=1}^{N'_{\text{param}}} r_{\text{param}}(v_i, v_i^{\text{gt}}), \tag{i}$$

where N_{param} denotes the total number of parameters across all tool calls in the ground truth. N'_{param} is the number of parameters for which both the tool call name and the parameter name are correctly matched between the predicted and ground truth tool calls. v_i and v_i^{gt} represent the predicted and ground truth parameter values, respectively. r_{param} calculates the reward for each individual parameter based on the data type of the parameter value:

1. For string-typed parameters, r_{param} takes the form as $r_{\text{param}}(v, v^{\text{gt}}) = 1 - \text{Normalized\_Edit\_Distance}(v, v^{\text{gt}})$.
2. For boolean-typed parameters, r_{param} is calculated as $r_{\text{param}}(v, v^{\text{gt}}) = \mathbb{I}_{v=v^{\text{gt}}}$, where $\mathbb{I}$ is the indicator function.
3. For other numerical parameters, we compute r_{param} as $r_{\text{param}}(v, v^{\text{gt}}) = \exp\left(-\frac{|v-v^{\text{gt}}|}{|v^{\text{gt}}|}\right)$.
4. For parameters of list or dictionary types, we compute the reward for each element individually and then take the average.

Through the above definition, the reward function r is within the range of $[0, 1]$.

The format of layers information *M*

```
[{
   "layerid": 15,          "kind": "pixel" ,                    "opacity": 100,
   "blend_mode": "Normal" ,                                  "clipped_to": null,
   "bbox": {
      "top": 0.4472,    "bottom": 0.6481 ,   "left": 0.0000 ,   "right": 0.2833 ,
 },
 {
   "layerid": 16,          "kind": "pixel" ,                    "opacity": 100,
   "blend_mode": "Normal" ,                                  "clipped_to": null,
   "bbox": {
      "top": 0.4099,    "bottom": 0.5673 ,   "left": 0.3078 ,   "right": 0.9247 ,
 },
 {
   "id": 17,               "kind": "pixel" ,                    "opacity": 100,
   "blend_mode": "Normal" ,                                  "clipped_to": null,
   "bbox": {
      "top": 0.5284,    "bottom": 0.7974 ,   "left": 0.5018 ,   "right": 0.8972 ,
 }]
```

Figure IV. The format of layer information M used in GraphicPlanner.

The input format of GraphicPlanner in $\mathcal{X}_{\text{gen}}$ mode

You are a professional designer, proficient in Adobe Photoshop.
The canvas size for this design is [canvas_width]×[canvas_height] pixels (width × height).
Here is the current rendered image, where all previous layers within the current group are incorporated into the design: $[R]$.
Here is the information of the previous layers within the current group: $[M]$.

- This is the start of a new group. You should insert the **group layer** and set the group attributes.
- You need to insert the **smart object/pixel layer** as an independent layer, or used as the clipping mask. You can also set layer mask to constrain the visible region. Resource image: $[a]$. Path: [The image path of a]. Aspect Ratio: [The aspect ratio of a].
- You need to insert the **text layer** as an independent layer, or used as the clipping mask. You can also set layer mask to constrain the visible region. The text to be rendered is $[a]$.
- You need to insert an **adjustment layer** as an independent layer, or used as the clipping mask. You can also set layer mask to constrain the visible region.

Based on the above information, output the Adobe Photoshop tool call trajectory.

Figure V. The input format of GraphicPlanner in $\mathcal{X}_{\text{gen}}$ mode.

The input format of GraphicPlanner in $\mathcal{X}_{\text{edt}}$ mode

You are a professional designer, proficient in Adobe Photoshop. You need to detect the deficiencies in the design and fix them.
The canvas size for this design is [canvas_width]×[canvas_height] pixels (width × height).
Here is the rendered image before the layers from the current group are incorporated: $[G]$.
Here is the current rendered image: $[R]$.
Here is the information of the layers within the current group: $[M]$.
Based on the above information, output the Adobe Photoshop tool call trajectory.

Figure VI. The input format of GraphicPlanner in $\mathcal{X}_{\text{edt}}$ mode.

D. More details of ToolExecutor

Building upon the previous work*, we implement ToolExecutor using Adobe Photoshop's Unified Extensibility Platform (UXP), enabling it to faithfully execute all tool calls listed in Tab. IV. Furthermore, we design a client-server architecture: the client transmits the predicted tool calls from GraphicPlanner over the network, while the server forwards these tool calls to ToolExecutor for performing actual manipulations on the PSD file.

The query prompt for generating user instructions

Please give me some user intention prompts (e.g., poster, advertisement, or flyer), which are used by the generation models to generate excellent designs. The intention prompt can incorporate some Chinese or/and English texts, used as main copy, descriptions, headlines, or other key information. Please indicate the following key aspects in user intention prompts.
1. Design scenarios: Provide a detailed description of the scenario you want the design to reflect. Please imagine as many scenarios as possible, such as a holiday poster, an event, a promotion, a product advertisement, etc.
2. Chinese or English texts (Optional, note that the Chinese texts need to be expressed in Chinese):
    a. Main Copy/Advertising Slogan: Insert a compelling advertising slogan or key message that should be featured prominently in the design.
    b. Headline: Insert a bold and eye-catching headline.
    c. Subheadline/Tagline: Insert a brief subtitle or secondary message that supports the headline.
3. Visual elements:
    a. Background: Specify what kind of background the design should have – for example, "bright and colorful," "minimalist and clean," or "urban and edgy".
    b. Images or graphics: Provide any suggestions for images, icons, or graphics that should appear in the design, such as "a photo of a new product," "abstract shapes," or "people celebrating".
    c. Overall mood: Describe the mood or vibe of the design, such as "energetic," "luxurious," or "fun and playful".
    d. Font style and size (Optional): Provide details on preferred font styles or sizes, such as "bold sans-serif for the headline" or "elegant serif for the subheading".
4. Additional requirements: List any other specific elements or features you want the design to include, such as a specific color scheme, brand logo, social media handles, or a call to action.
Please compose a complete sentence instead of returning structural information.

Figure VII. The query prompt for generating user instructions.

The query prompt for design evaluation

As a professional designer, please evaluate the design image based on the user-provided design intention and the criteria listed below. For each dimension, assign a score from 1 to 10, where 1 represents a very poor quality and 10 represents an excellent quality. Provide your score based on your expertise and the overall impression of the design, considering each of the following factors:
1. "Aesthetic quality" evaluates the visual appeal and artistic style of the design. Consider the overall beauty and the emotional impact the design evokes. How well does the design look visually?
2. "Design layout" assesses the organization and structure of the design elements. How well are the visual elements arranged, and how do they interact with each other? Does the layout contribute to a clear and engaging presentation?
3. "Content relevance" evaluates how well the design content aligns with the intended message or purpose. Does the design convey the intended concept or information effectively and clearly?
4. "Color harmony" assesses the color choices and how they complement each other. Are the colors used in a way that feels balanced, cohesive, and appropriate for the design's context?
5. "Innovation" evaluates the creativity and originality of the design. How innovative is the design? Does it offer a fresh or unique approach that stands out?

Figure VIII. The query prompt for VLM evaluation.

*https://github.com/mikechambers/adb-mcp

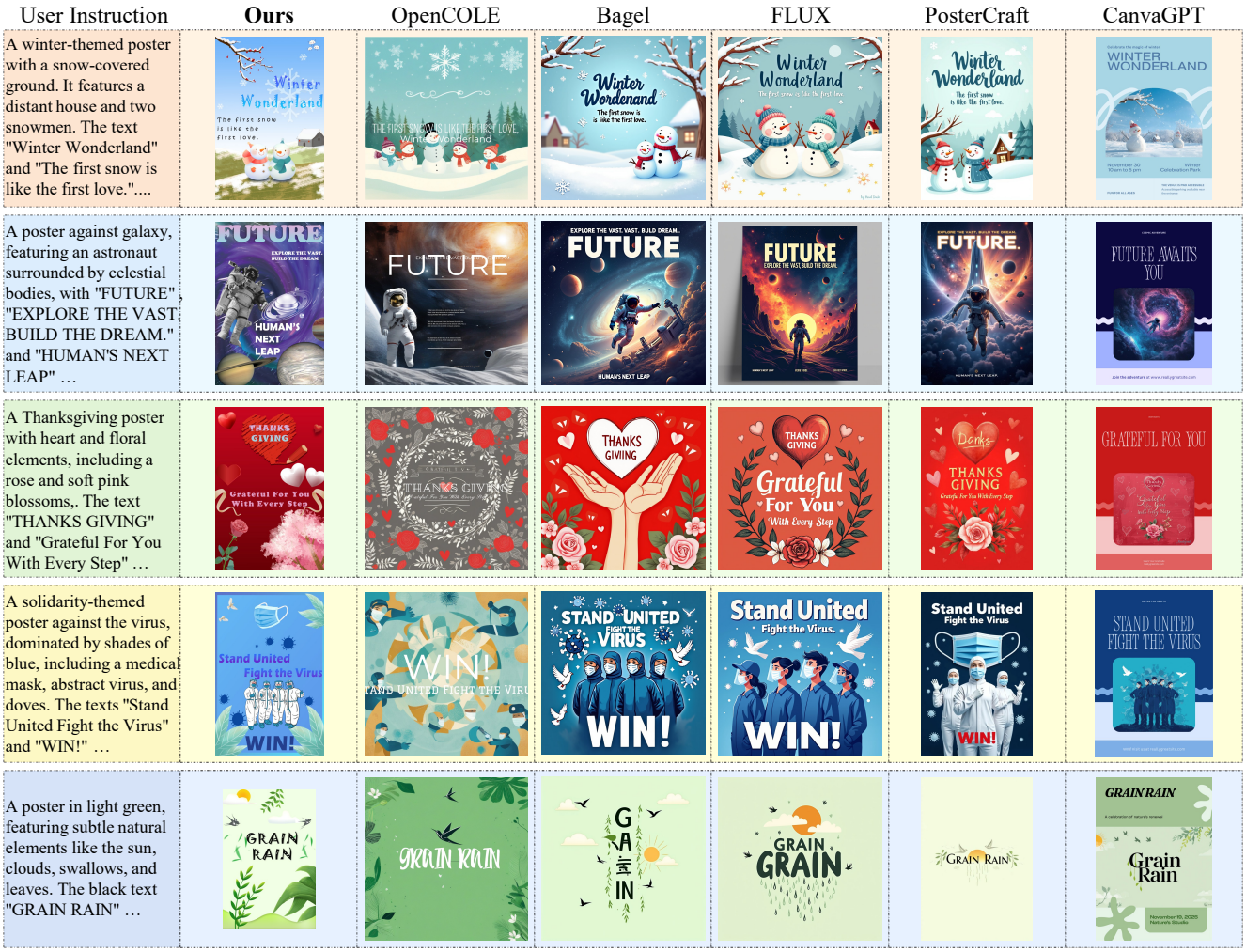


Figure IX. The results of translating the user intention into the final design for English scenarios.

E. More Details of Benchmarks

To evaluate the model's capability to directly translate user intentions into the final designs, we construct 250 user instructions for both English and Chinese scenarios, sourced from GPT-5 and human annotators. The query prompt used for GPT-5 is shown in Fig. VII.

Moreover, to assess the model's ability in graphic design composition, we leverage the Crello-v5 [5] test set and 200 copyright-free PSD files collected from the internet, evaluating performance in both simple and complex layer hierarchies. Specifically, using the annotations from Crello-v5, we acquire all the image and text assets from each design. For the PSD files, we use the same approach discussed in Sec.3 of the main paper, extracting the corresponding raw assets and their grouping information. Based on the organized assets, we are able to evaluate our method.

F. More Details of the Evaluation

We first leverage Qwen2.5-VL-7B [6] to assess the design in the following aspects. **1). Aesthetic quality (Qua.)** assesses the visual quality, such as artistic style and the overall appeal. **2). Design layout (Lay.)** determines the organization quality of visual elements, including the arrangement and the interaction of each element. **3). Content relevance (Rel.)** evaluates the semantic alignment between the visual content and user intention. **4). Color harmony (Har.)** assesses the coherence of color usage. **5). Innovation(Inn.)** reflects the originality and the creativity of the design. The query prompt is demonstrated

Figure X. The results of translating the user intention into the final design for Chinese scenarios.

in Fig. VIII.

In addition, we also employ 14 professional designers and 32 general users to conduct the user study with the generated designs from different methods. Similar to the VLM evaluation, they are asked to assess each design in the above dimensions. For both VLM and human evaluation, all of the scores are within the range of 1 to 10. For each score, we average the results.

G. More Comparison Results

We provide more comparison results in this section. Fig. IX-Fig. X represent the results of the methods in translating the user instructions to final designs, for English and Chinese scenarios, respectively. As shown in the figures, most of the methods struggle to generate accurate texts, particularly for Chinese characters from Fig. X. Furthermore, the outputs from these methods are either non-editable or contain a few layers with simple attributes, hindering their professionalism and flexibility.

In addition, Fig. XI also illustrates the model performance on graphic design composition, using the test set from Crello-v5 [5]. Since LaDeCo [7] was trained on Crello-v4 [5], it exhibits higher similarity to the ground truth. However, this method sometimes produces suboptimal layouts that lead to occlusion of key elements. In contrast, our GraphicPlanner achieves more creative and coherent layouts, resulting in visually superior outcomes. Furthermore, Fig. XII-Fig. XIII demonstrate the results of our GraphicPlanner on the collected PSD files. These results demonstrate that our method exhibits superior performance in both simple and complex layer hierarchies, indicating that GraphicPlanner is equipped with a powerful tool-use capability and a human-like creative design workflow.

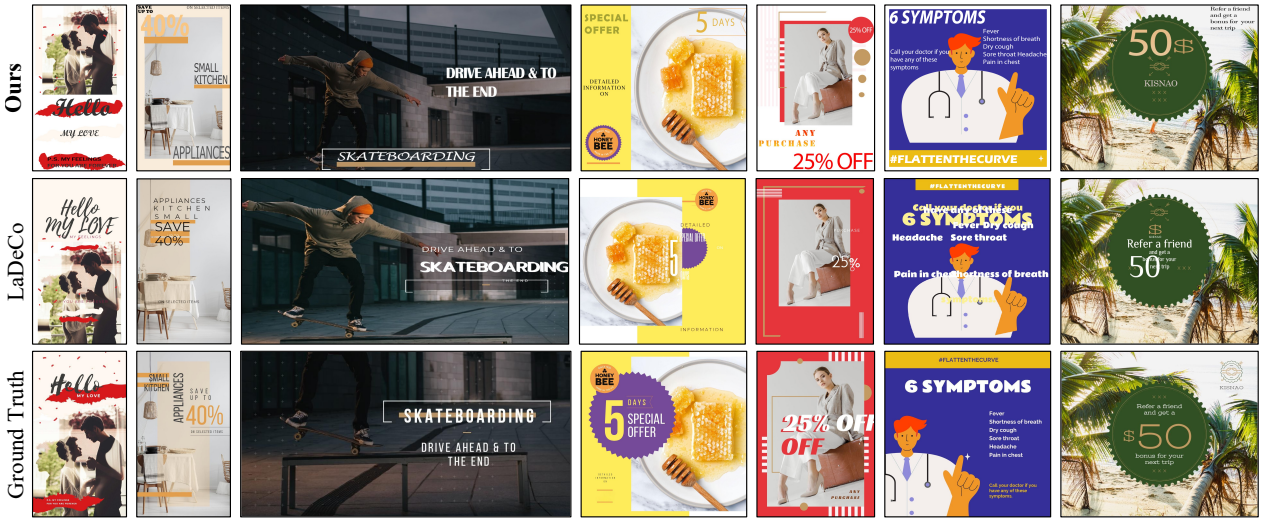


Figure XI. The results of graphic design composition on Crello-v5 [5] test set.

H. More Ablation Studies

Fig. XIV presents more results of ablation studies, demonstrating the effects of our key designs.

1. **w/o $\mathcal{X}_{\text{edt}}$.** With only $\mathcal{X}_{\text{gen}}$ mode, the model lacks awareness of subsequent assets within the same group when inserting the current layer, resulting in inaccurate tool call predictions. Therefore, the model without $\mathcal{X}_{\text{edt}}$ is unable to refine the inferior layers, resulting in suboptimal arrangements. As indicated in the 3rd example in Fig. XIV, several text layers are incorrectly scaled up and centered, severely degrading visual harmony.
2. **w/o M.** Without the layer information M, the model is unaware of the integrated layers within the current group. Consequently, the model tends to generate designs with multiple overlaps. As shown in most examples of Fig. XIV, some elements are stacked, producing a low-quality layout.
3. **w/o RL.** In the absence of reinforcement learning, the model struggles to predict precise parameter values in tool calls, leading to suboptimal outcomes. As shown in the 2nd example of Fig. XIV, the text layers are placed in incorrect locations.

In contrast, our proposed GraphicPlanner achieves the most visually coherent results, consistently outperforming all ablated variants.

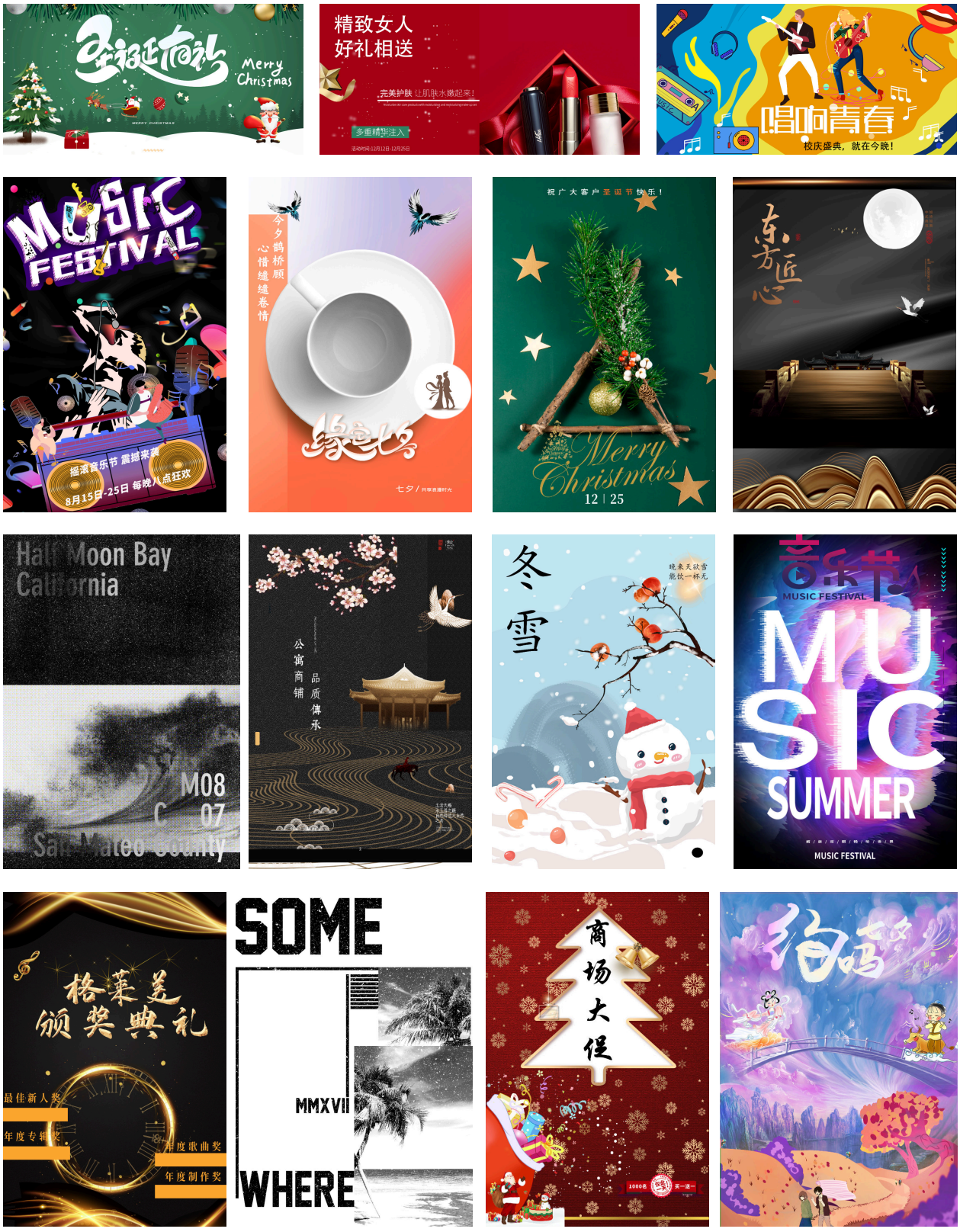


Figure XII. The results of graphic design composition on the collected PSD files.

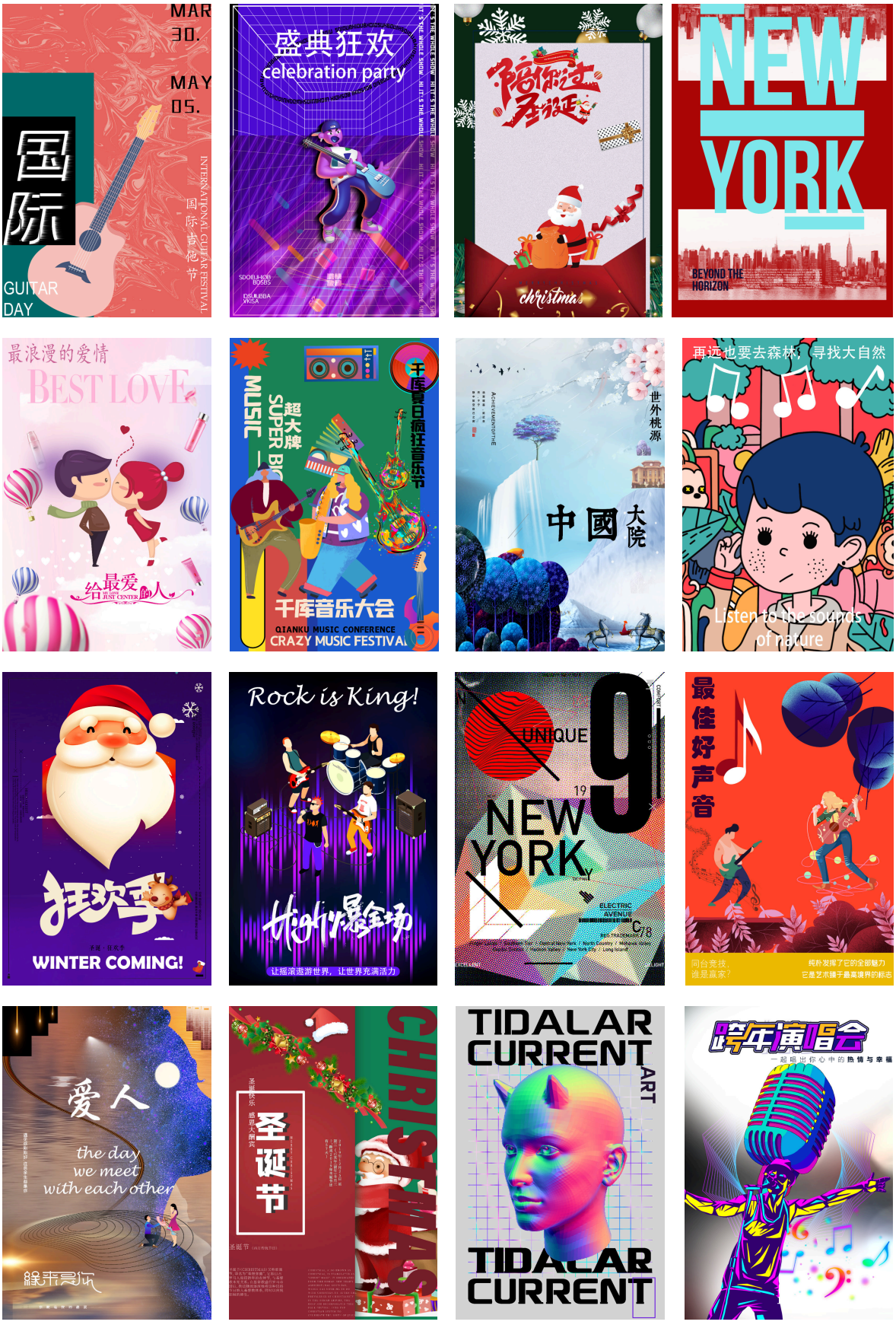


Figure XIII. The results of graphic design composition on the collected PSD files.

Figure XIV. Ablation studies on **(a)** Crello-v5 [5] test set and **(b)** our collected PSD files.

The query prompt for LLM used in AssetCollector

You are an AI assistant specialized in designing dynamic, customized posters based on user input. Your task is to analyze the user's request, break it down into visual concepts, and generate a structured JSON output containing all the necessary assets for poster creation.

Overview:

Your goal is to:

1. Analyze the user's poster request
2. Identify key visual concepts (layers) needed for the design
3. Generate detailed specifications for each visual asset (multiple assets per group)
4. Output a structured JSON format that can be used by a design system

Important Guidelines:

1. Do NOT use fixed templates. Each poster should be uniquely designed based on the specific user request
2. Visual concepts should be determined dynamically based on the theme, style, and elements mentioned by the user
3. Each visual concept group MUST contain multiple assets (minimum 1-3 per group) to provide variety and design flexibility

Key Instructions:

1. Analyze User Input

Carefully analyze the user's request to understand:

- Theme: What is the overall theme or genre? (e.g., superhero, horror, romance, sports)
- Key Elements: What specific elements should be included? (e.g., characters, objects, settings)
- Mood/Tone: What emotions or atmosphere should the poster convey? (e.g., dramatic, playful, mysterious)
- Text Elements: What titles, taglines, or text should appear on the poster?
- Style Preferences: Are there any specific visual styles mentioned? (e.g., retro, modern, minimalist)

2. Identify Visual Concepts

Based on your analysis, determine the visual concept groups needed. These typically include:

- Background layers: Environmental settings, atmospheric effects
- Character/Object layers: Main subjects, secondary elements
- Effect layers: Special effects, lighting, textures
- Text layers: Titles, subtitles, taglines, credits

Each visual concept group should be logically organized and ordered for proper layering in the final design.

Critical Requirement: Each group MUST contain multiple assets (minimum 1-3 per group). When planning groups, think about:

- Multiple variations of the same concept
- Different perspectives or angles
- Alternative styles or treatments
- Complementary elements within the same visual theme

3. Generate Asset Specifications

For each asset within a visual concept group, generate appropriate specifications:

For Image Assets (type: "image"):

- **prompt:** Create a detailed, descriptive prompt for AI image generation. Include:
    - Subject description
    - Pose/action/positioning
    - Visual style and artistic direction
    - Lighting and atmosphere
    - Color palette suggestions
    - Composition details
- **query:** Use this only when real-time data or current information is needed (e.g., "latest design trends for 2024", "current popular movie posters"). Otherwise, set to `null`.

For Text Assets (type: "text"):

- text_content: The actual text to display (e.g., title, tagline)
- need_artistic_prompt: Set to `true` if the text requires special styling, custom fonts, or artistic rendering
- artistic_prompt: If `need_artistic_prompt` is true, provide a detailed description of the desired font style, effects, and visual treatment

4. Asset Organization Rules

- group_order: Determines the layering order of visual concept groups (1 = bottom layer, higher numbers = top layers)
- group_name: Descriptive name for the visual concept group (e.g., "Background Layer", "Hero Characters", "Title Treatment")
- asset_order: Determines the order of assets within a visual concept group (for rendering or selection purposes)
- Multiple Assets Per Group: Each visual concept group MUST contain multiple assets (at least 1-3 assets per group) to provide variety and options. This allows for richer compositions and gives flexibility in the final design. Think about variations in:
    - Different perspectives or angles
    - Multiple characters or objects within the same category
    - Variations in style or intensity
    - Alternative compositions or arrangements

The query prompt for LLM used in AssetCollector

5. Quality Guidelines

Image Prompts Must Include:

- Specific details about the subject
- Clear description of visual style
- Composition and framing details
- Lighting and atmosphere
- Color palette or color mood
- Artistic medium or rendering style (photorealistic, illustrated, painted, etc.)

Text Treatments Should Consider:

- Font personality matching the theme
- Visual effects (glow, shadow, texture, distress, etc.)
- Integration with the overall design aesthetic
- Readability while maintaining artistic appeal

6. Output Format

Your response must be a valid JSON object with the following structure:

```
{
 "visual_concepts": [
   {
    "group_order": <rendering order of the visual concept>,
    "group_name": "<descriptive name>",
    "assets": [
      {
       "asset_order": < rendering order of the asset>,
       "type": "image" | "text",
       "prompt": "<detailed prompt for image generation>" | null,
       "query": "<search query for the image>" | null,
       "text_content": "<actual text content>" | null,
       "need_artistic_prompt": <is it necessary to generate artistic text image>,
       "artistic_prompt": "<font/style description>" | null
      }
    ]
   }
 ]
}
```

Field Specifications:

- **group_order:** Integer (1, 2, 3, ...) – Rendering order of the visual concept.
- **group_name:** String - Descriptive name for this group/visual concept
- **asset_order:** Integer (1, 2, 3, ...) – Rendering order of assets within the group/visual concept
- **type:** String - Either "image" or "text"
- **prompt:** String or null - Detailed image generation prompt (only for type="image" when AI generation is sufficient)
- **query:** String or null - Search query (only for type="image")
- **text_content:** String or null - Actual text to display (only for type="text")
- **need_artistic_prompt:** Boolean - Whether text needs artistic styling (only for type="text")
- **artistic_prompt:** String or null - Description of font/text styling (only when need_artistic_prompt=true)

Validation Rules:

1. For visual concept groups:

- Each group MUST contain at least 1-3 assets (preferably more for richer designs)
- Assets within a group should be related to the same visual concept but provide variety
- Single-asset groups are NOT acceptable unless there's a strong justification (e.g, background)

2. For image assets:

- Either `prompt` OR `query` must be provided (not both, not neither)
- `text_content` must be null
- `need_artistic_prompt` must be false
- `artistic_prompt` must be null

3. For text assets:

`text_content` must be provided
`prompt` must be null
`query` must be null
If `need_artistic_prompt` is true, `artistic_prompt` must be provided
If `need_artistic_prompt` is false, `artistic_prompt` must be null

Response Format:

Always respond with ONLY the valid JSON object. Do not include any explanatory text before or after the JSON. The response should be parseable by a JSON parser.

Figure XV. The query prompt of the LLM used in AssetCollector.

Table IV. Summary of tool calls constructed in CreativePSD.

Function Name	Description	Parameters
select_polygon	Create polygon selection and select the specified layer	layer_id (int): Layer ID; feather (int): Feather pixels (0-1000); anti_alias (bool): Anti-aliasing option; points (list[dict]): List of points to define the polygon, each point has x, y
add_layer_mask_from_selection	Create layer mask from current selection	layer_id (int): Layer ID
set_layer_properties	Set layer blending mode and opacity properties	layer_id (int): Layer ID; blend_mode (str): Blending mode; layer_opacity (int): Layer opacity (0-100); fill_opacity (int): Fill opacity (0-100); is_clipping_mask (bool): Whether it is a clipping mask
save_document_image_as_png	Capture Photoshop document and save as PNG file	file_path (str): PNG file save path
select_layer_group	Select the parent group of the specified layer	layer_id (int): Layer ID, its parent group will be selected
get_current_effects	Get all effects applied to the current active layer	layer_id (int): Layer ID
get_document_info	Get information about the current active document	None
get_active_layer	Get the ID and information of the current active layer	None
get_active_group_layer_count	Get the number of layers in the current selected group	None
move_layer	Move layer position in the layer stack	layer_id (int): Layer ID; position (str): Position option (TOP/BOTTOM/UP/DOWN)
create_document	Create a new Photoshop document	filename (str): New document name; width (int): Width; height (int): Height; resolution (int): Resolution; fill_color (dict): RGB color dictionary; color_mode (str): Color mode
save_document	Save the current Photoshop document	None

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
save_document_as	Save the current document in a specified location and format	file_path (str): Save path; file_type (str): File format (PSD/PNG/JPG)
close_document	Close the current document (without saving changes)	None
save_document_as_and_close	Save the document and then close it	file_path (str): Save path; file_type (str): File format
create_group	Create a new group layer	name (str): Group name; opacity (int): Opacity; blend_mode (str): Blending mode
add_group_layer	Create a new group layer	group_name (str): Group name; opacity (int): Opacity; blend_mode (str): Blending mode
close_current_group	Close the current group layer	None
place_image	Place an image on the specified layer	layer_id (int): Layer ID; image_path (str): Image file path
rename_layers	Rename one or more layers	layer_data (list[dict]): List of layer rename information, each dictionary contains layer_id and new_layer_name
insert_image	Insert an image	layer_name (str): Layer name; opacity (int): Opacity; blend_mode (str): Blending mode; image_path (str): Image path
get_layer_bounds	Get the pixel boundaries of the specified layer	layer_id (int): Layer ID
select_active_layer	Select the specified layer by its ID	layer_id (int): Layer ID
scale_layer	Scale the specified layer by percentage	layer_id (int): Layer ID; width (float): Horizontal scaling percentage; height (float): Vertical scaling percentage; anchor_position (str): Anchor position; interpolation_method (str): Interpolation method
rotate_layer	Rotate the specified layer by angle	layer_id (int): Layer ID; angle (int): Rotation angle (-359 to 359 degrees); anchor_position (str): Rotation anchor position; interpolation_method (str): Interpolation method

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
translate_layer	Move the layer in the X and Y axes	layer_id (int): Layer ID; x_offset (float): Horizontal offset (negative for left, positive for right); y_offset (float): Vertical offset (negative for down, positive for up)
adjust_current_image	Adjust the current image	normalize_resize_width (float): Normalized width; normalize_resize_height (float): Normalized height; normalize_position_x (float): Normalized X position; normalize_position_y (float): Normalized Y position; angle (float): Angle
create_multi_line_text_layer	Create multi-line text layer	text (str): Text content; postscript_font_name (str): Font name; font_size (float): Font size (points); text_color (dict): RGB color dictionary; blend_mode (str): Blending mode; position (list): [x, y] position; bounds (dict): Text bounding box; leading (float): Line spacing; tracking (float): Letter spacing; text_orientation (str): Text orientation (horizontal/vertical); layer_name (str): Layer name; opacity (int): Layer opacity
create_single_line_text_layer	Create single line text layer	text (str): Text content; postscript_font_name (str): Font name; font_size (float): Font size (points); text_color (dict): RGB color dictionary; blend_mode (str): Blending mode; position (list): [x, y] position; bounds (dict): Text bounding box; leading (float): Line spacing; tracking (float): Letter spacing; text_orientation (str): Text orientation (horizontal/vertical); layer_name (str): Layer name; opacity (int): Layer opacity
add_hue_saturation_adjustment	Apply hue/saturation adjustment layer to current layer	enable_colorization (bool): Enable colorization; colorize_hue (int): Hue value when colorization is enabled; colorize_saturation (int): Saturation value when colorization is enabled;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		colorize_lightness (int): Lightness value when colorization is enabled; master_hue (int): Master hue value when colorization is not enabled; master_saturation (int): Master saturation value when colorization is not enabled; master_lightness (int): Master lightness value when colorization is not enabled; red_hue (int): Red hue value when colorization is not enabled; red_saturation (int): Red saturation value when colorization is not enabled; red_lightness (int): Red lightness value when colorization is not enabled; yellow_hue (int): Yellow hue value when colorization is not enabled; yellow_saturation (int): Yellow saturation value when colorization is not enabled; yellow_lightness (int): Yellow lightness value when colorization is not enabled; green_hue (int): Green hue value when colorization is not enabled; green_saturation (int): Green saturation value when colorization is not enabled; green_lightness (int): Green lightness value when colorization is not enabled; blue_hue (int): Blue hue value when colorization is not enabled; blue_saturation (int): Blue saturation value when colorization is not enabled; blue_lightness (int): Blue lightness value when colorization is not enabled; purple_hue (int): Purple hue value when colorization is not enabled; purple_saturation (int): Purple saturation value when colorization is not enabled; purple_lightness (int): Purple lightness value when colorization is not enabled; pink_hue (int): Pink hue value when colorization is not enabled; pink_saturation (int): Pink saturation value when colorization is not enabled; pink_lightness (int): Pink lightness value when colorization is not enabled; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_curves_adjustment	Apply curves adjustment layer to current layer	rgb_curve (list): RGB curve points array [(x, y), ...] layer_blend_mode (str): Blending mode;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		layer_opacity (int): Layer opacity
add_brightness_contrast_adjustment	Apply brightness/contrast adjustment layer to current layer	brightness (int): Brightness adjustment value; contrast (int): Contrast adjustment value; use_legacy (bool): Whether the legacy algorithm is used; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_color_balance_adjustment	Apply color balance adjustment layer to current layer	shadows (list): Shadow adjustment values [cyan-red, magenta-green, yellow-blue]; midtones (list): Midtone adjustment values [cyan-red, magenta-green, yellow-blue]; highlights (list): Highlight adjustment values [cyan-red, magenta-green, yellow-blue]; luminosity (bool): Whether luminosity is preserved during adjustment; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_levels_adjustment	Apply levels adjustment layer to current layer	channel (str): Channel being adjusted (RGB); input_floor (int): Minimum input value; input_ceiling (int): Maximum input value; output_floor (int): Minimum output value; output_ceiling (int): Maximum output value; gamma (float): Gamma adjustment value; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_vibrance_adjustment	Apply vibrance adjustment layer to current layer	vibrance (int): Vibrance adjustment value; saturation (int): Saturation adjustment value; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_solid_color_fill_adjustment	Create solid color fill layer above current layer	red_fill (int): Red fill value (0-255); green_fill (int): Green fill value (0-255); blue_fill (int): Blue fill value (0-255); layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_selective_color_adjustment	Apply selective color adjustment layer to current layer	method (str): Adjustment method (relative/absolute); reds (list): Red adjustment values; yellows (list): Yellow adjustment values; greens (list): Green adjustment values;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		cyans (list): Cyan adjustment values; blues (list): Blue adjustment values; magentas (list): Magenta adjustment values; whites (list): White adjustment values; neutrals (list): Neutral adjustment values; blacks (list): Black adjustment values layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_gradient_fill_adjustment	Create gradient fill layer above current layer	type (str): The type of the gradient fill; gradient_parameters (dict): The gradient parameters; angle (float): The gradient angle; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_gradient_map_adjustment	Apply gradient map adjustment layer to current layer	dither (bool): Whether dithering is enabled for the gradient map; reverse (bool): Whether the gradient direction is reversed; gradient_parameters (dict): The gradient parameters; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_exposure_adjustment	Apply exposure adjustment layer to current layer	exposure (float): Exposure adjustment value; offset (float): Offset adjustment value; gamma (float): Gamma adjustment value layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_photo_filter_adjustment	Apply photo filter adjustment layer to current layer	L (float): L value in the Lab color space; a (float): a value in the Lab color space; b (float): b value in the Lab color space; density (float): Density of the photo filter; luminosity (bool): Whether luminosity is preserved; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_black_and_white_adjustment	Apply black and white adjustment layer to current layer	reds (int): Red adjustment value; yellows (int): Yellow adjustment value; greens (int): Green adjustment value; cyans (int): Cyan adjustment value; blues (int): Blue adjustment value; magentas (int): Magenta adjustment value;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		use_tint (bool): Whether a tint is used; tint_color (dict): Tint color in the form of {red, green, blue} (if using tint); layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_colorlookup_adjustment	Apply color lookup layer to current layer	lookup_type (str): The type of the color lookup; dither (bool): Whether dithering is enabled; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_channel_mixer_adjustment	Apply channel mixer adjustment layer to current layer	monochrome (bool): Whether monochrome is enabled; red (list): Red channel mix values; green (list): Green channel mix values; blue (list): Blue channel mix values; constants (list): Constants for the channel mixer; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_pattern_fill_adjustment	Create pattern fill layer above current layer	pattern_name (str): Name of the pattern; scale (str): The scale of the pattern; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_threshold_adjustment	Apply threshold adjustment layer to current layer	threshold (int): Threshold value for the adjustment; layer_blend_mode (str): Blending mode; layer_opacity (int): Layer opacity
add_gradient_overlay	Add gradient overlay effect to current layer	blend_mode (str): The blending mode for the gradient overlay; dither (bool): Whether the dithering is applied; opacity (float): The opacity of the gradient overlay; reverse (bool): Whether the gradient direction is reversed; style (str): The gradient style; angle (float): The angle of the gradient; scale (float): The scale/size of the gradient; method (str): The method for gradient overlay; align_with_layer (bool): Whether the effect is aligned with the layer; gradient_parameters (dict): Gradient parameters
add_color_overlay	Add color overlay effect to current layer	blend_mode (str): The blending mode for the color overlay; opacity (float): The opacity of the color overlay; color (dict): The color of the color overlay
add_pattern_overlay	Add pattern overlay effect to current layer	blend_mode (str): The blending mode for the pattern overlay;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		opacity (float): The opacity of the pattern overlay; pattern_name (str): The pattern name of the pattern overlay; angle (float): The angle of the pattern overlay; scale (float): The scale of the pattern overlay; link_with_layer (bool): Whether the effect is aligned with the layer
add_bevel_emboss	Add bevel emboss effect to current layer	style (str): The style of the bevel; type (str): The type of the bevel; depth (float): The depth of the bevel; direction (str): The direction of the bevel; size (float): The size of the bevel; soften (float): Edge softness of the bevel; angle (float): The angle of the bevel; use_global_light (bool): Whether the global light is enabled; altitude (float): Light source altitude for the bevel; gloss_contour (str): Contour curve for the bevel; highlight_mode (str): Blending mode of the highlight; highlight_color (dict): Color of the highlight; highlight_opacity (float): The opacity of the highlight; shadow_mode (str): Blending mode of the shadow; shadow_color (dict): Color of the shadow; shadow_opacity (float): The opacity of the shadow; anti_aliased (bool): Whether anti-aliasing is enabled; contour_parameters (dict): The parameters of the contour; texture_parameters (dict): The parameters of the texture;
add_drop_shadow	Add drop shadow effect to current layer	blend_mode (str): The blending mode for the drop shadow; opacity (float): The opacity of the shadow; angle (float): The angle of the shadow; distance (float): The distance of the shadow; spread (float): The spread of the shadow; size (float): The size of the shadow; contour (str): The contour of the shadow; anti_aliased (bool): Whether anti-aliasing is enabled; noise (float): The noise of the shadow; layer_knocks_out (bool): Whether the shadow knocks out the layer color; color (dict): The color of the drop shadow;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		use_global_light (bool): Whether the global light is enabled
add_stroke	Add stroke effect to current layer	blend_mode (str): The blending mode for the stroke; size (float): The size of the stroke; position (float): The position of the stroke; opacity (float): The opacity of the stroke; overprint (bool): Whether the stroke overprints other elements; fill_type (str): The stroke type; fill_parameters (dict): The stroke parameters
add_satin	Add satin effect to current layer	blend_mode (str): The blending mode for the satin; opacity (float): The opacity of the satin; color (dict): The color of the satin; angle (float): The angle of the satin; distance (float): The distance of the satin; size (float): The size of the satin; contour (str): The contour of the satin; anti_aliased (bool): Whether anti-aliasing is enabled for the satin; inverted (bool): Whether the satin effect is inverted
add_inner_shadow	Add inner shadow effect to current layer	blend_mode (str): Blending mode for the inner shadow; opacity (float): Opacity of the inner shadow; angle (float): Angle of the inner shadow; use_global_light (bool): Whether to use global light settings; distance (float): Distance from the object for the inner shadow; choke (float): Spread amount for the inner shadow; size (float): Size of the inner shadow; contour (str): Contour curve of the inner shadow; noise (float): Amount of noise for the inner shadow; anti_aliased (bool): Whether anti-aliasing is enabled for the inner shadow; color (dict): The color of the inner shadow
add_outer_glow	Add outer glow effect to specified layer	blend_mode (str): Blending mode for the outer glow; opacity (float): Opacity of the outer glow; noise (float): Amount of noise for the outer glow; color (dict): Color of the outer glow; technique (str): Glow rendering technique (*e.g.*, softer, precise); spread (float): Spread of the glow; size (float): Size of the glow; contour (str): Contour curve for the glow; anti_aliased (bool): Whether anti-aliasing is enabled for the glow;

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
		range (float): Range of the glow (how far the glow extends); jitter (float): Jitter amount for the glow
add_inner_glow	Add inner glow effect to current layer	blend_mode (str): Blending mode for the inner glow; opacity (float): Opacity of the inner glow; noise (float): Amount of noise for the inner glow; color (dict): Color of the inner glow; technique (str): Glow rendering technique (*e.g.*, softer, precise); source (str): Glow source (*e.g.*, edge); choke (float): Choke of the glow; size (float): Size of the glow; contour (str): Contour curve for the glow; anti_aliased (bool): Whether anti-aliasing is enabled for the glow; range (float): Range of the glow (how far the glow extends); jitter (float): Jitter amount for the glow
fill_selection	Fill the selected area with color on the specified layer	layer_id (int): Layer ID; color (dict): Fill color; blend_mode (str): Fill blending mode; opacity (int): Fill opacity
delete_selection	Delete the selected area within the specified layer	layer_id (int): Layer ID
invert_selection	Invert the current selection	None
clear_selection	Clear/cancel the current selection	None
select_rectangle	Create a rectangular selection and select the specified layer	layer_id (int): Layer ID; feather (int): Feather pixels (0-1000); anti_alias (bool): Anti-aliasing; bounds (dict): Rectangle bounding box
select_ellipse	Create an elliptical selection and select the specified layer	layer_id (int): Layer ID; feather (int): Feather pixels (0-1000); anti_alias (bool): Anti-aliasing; bounds (dict): Ellipse bounding box
align_content	Align the content of the specified layer to the current selection	layer_id (int): Layer ID; alignment_mode (str): Alignment mode

Continued on next page

Table IV – Continued from previous page

Function Name	Description	Parameters
select_subject	Automatically select the subject in the specified layer	layer_id (int): Layer ID
select_sky	Automatically select the sky in the specified layer	layer_id (int): Layer ID
clip_to_below_layer	Clip the current layer to the layer below	None
add_layer_mask	Create a layer mask using the specified boundary box on the current layer	masks (list): Mask information
get_group	Get the current group layer information	None
get_layers	Return a nested list containing layer information and order	None
set_opacity	Set the opacity of a layer	layer_id (int): Layer ID; opacity (int): Opacity value (0-255)
set_rotation	Set the rotation of a layer	layer_id (int): Layer ID; target_angle (float): Target rotation angle in degrees
set_transform	Set the transformation (position and size) of a layer	layer_id (int): Layer ID; normalize_resize_width (float): Normalized width; normalize_resize_height (float): Normalized height; normalize_position_x (float): Normalized X position; normalize_position_y (float): Normalized Y position
set_blendmode	Set the blending mode of a layer	layer_id (int): Layer ID; blend_mode (str): Blending mode

References

[1] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv*, 2023. 9

[2] Black Forest Labs. Flux.1-dev, 2024. 9

[3] Yu Gao, Lixue Gong, Qiushan Guo, Xiaoxia Hou, Zhichao Lai, Fanshi Li, Liang Li, Xiaochen Lian, Chao Liao, Liyang Liu, et al. Seedream 3.0 technical report. *arXiv*, 2025. 9

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 10

[5] Kota Yamaguchi. Canvasvae: Learning to generate vector graphic documents. In *ICCV*, 2021. 14, 15, 16, 19

[6] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv*, 2024. 14

[7] Jiawei Lin, Shizhao Sun, Danqing Huang, Ting Liu, Ji Li, and Jiang Bian. From elements to design: A layered approach for automatic graphic design composition. In *CVPR*, 2025. 15