PEAT: PROGRESSIVE EDITING VIA ASCII ART TRANSFORMS FOR RENDERING IN PIXELS

Anonymous authors
Paper under double-blind review

ABSTRACT

ASCII art represents object shapes and tones through spatial arrangements of characters. We introduce PEAT (Progressive Editing via ASCII Art Transforms), which uses ASCII art as a visual bottleneck for multi-turn image editing. This representation provides a discrete, inspectable editing state while keeping the original image as a separate appearance reference. At each turn, a large language model (LLM) updates the ASCII art and scene prompt from a user instruction. After the final turn, an image editing model synthesizes an RGB image from the updated state, the aggregated edit instruction, and the original image. Across four turns, PEAT shows the smallest decline in CLIP-I and AugCLIP among six editors and achieves the highest fourth-turn AugCLIP, MANIQA, and MUSIQ scores. Over a four-turn sequence, it uses 60.2% less inference time and 90.9% less GPU energy than sequential Qwen-Edit.



Figure 1: Overview of PEAT. Sequential RGB editing synthesizes an image at every turn, whereas PEAT accumulates edits in ASCII art and synthesizes RGB only at the final stage.

1 INTRODUCTION

Instruction-guided image editing applies changes specified in natural language while preserving other image content (Kim et al., 2025). Users inspect intermediate results and provide follow-up instructions, forming a sequence of edits in multi-turn image editing (Zhang et al., 2023a). As objects are added or their positions and poses are adjusted, the current composition guides the next instruction, and the content to preserve includes both source information and earlier edits.

Diffusion-based editors use attention control and inversion to incorporate source structure (Hertz et al., 2023; Mokady et al., 2023), or learn to apply changes directly from instructions (Brooks et al., 2023; Sheynin et al., 2024). When these editors use a previously generated RGB image as the next input, unintended changes can propagate alongside the requested edits. Multi-turn evaluation on MagicBrush shows error accumulation across turns (Zhang et al., 2023a). Prior work references both the source image and the previous result to balance editability and preservation, while still

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

synthesizing an image at each turn (Zhou et al., 2025); our goal is to retain source appearance and earlier edits across successive instructions.

One alternative is to aggregate instructions and synthesize an image only after the final turn. This leaves a representation problem: what state should carry successive edits before the image is generated? An aggregated instruction records the requests, but does not itself provide a rendered spatial state after each edit. We investigate an explicit state that can be updated from instructions, inspected between turns, and passed to final synthesis while the original image remains the appearance reference. LayoutGPT and LLM-grounded Diffusion plan object layouts for synthesis, the latter over multiple rounds (Feng et al., 2023; Lian et al., 2024), and ControlNet conditions synthesis on spatial maps (Zhang et al., 2023b). We ask three questions. (1) How does editing through a separate state affect performance degradation across turns compared with sequential RGB editing? (2) How well does ASCII art support image reconstruction compared with other spatial conditions? (3) What editing quality and inference cost does maintaining an explicit state yield compared with aggregating instructions alone?

We introduce PEAT (Progressive Editing via ASCII Art Transforms), which maintains ASCII art as the editing state and synthesizes RGB only at the final stage (Figure 1). Tone-based ASCII art represents intensity through a character grid (Xu et al., 2010); unlike bounding boxes, which give only positions and extents, the grid also encodes coarse foreground shape. Its token sequence is updated by a large language model (LLM), and its rendering serves both as a spatial preview and as a synthesis condition. An accompanying scene description carries semantic attributes that the grid does not encode.

PEAT encodes foreground ASCII art as codebook-based patch tokens and trains the LLM to update this sequence together with the scene description; the updated pair is the input state for the next instruction. After the final turn, a diffusion-based image editing model synthesizes the result from the rendered grid as the spatial condition, the original image as the appearance reference, the final description, and the aggregated instruction.

We evaluate PEAT on four-turn sequences against five editors that pass each synthesized image to the next turn. Among the six methods, PEAT has the smallest first-to-last-turn decline in CLIP-I and AugCLIP, and the highest final-turn AugCLIP, MANIQA, and MUSIQ scores. In reconstruction experiments, ASCII art achieves higher CLIP-I, SSIM, and PSNR than bounding boxes, masks, and edge maps under the evaluated setup. Aggregating instructions and synthesizing once also yields higher preservation scores than sequential RGB editing. PEAT maintains an inspectable intermediate state with similar CLIP-I and AugCLIP to this prompt-only configuration, and higher CLIP-T and MUSIQ. Per edit, PEAT uses 60.2% less time and 90.9% less GPU energy than sequential Qwen-Edit, and its measured time and energy are also below those of a single prompt-only synthesis call.

Our contributions are as follows.

- We introduce PEAT, which edits an inspectable ASCII art state across instructions and guides final synthesis with its rendering and the original image.
- We develop a codebook-based patch representation connecting LLM editing, visual inspection, and spatial conditioning, and compare its reconstruction quality with box, mask, and edge conditions.
- We show a smaller decline in CLIP-I and AugCLIP across four turns than sequential RGB editors, and report quality and measured cost against both baselines.

2 Related Work

2.1 Image Editing and Spatial Control

Diffusion-based editors preserve source content by controlling synthesis through cross-attention, inversion, edit masks, or reused source features (Hertz et al., 2023; Mokady et al., 2023; Couairon et al., 2023; Tumanyan et al., 2023; Cao et al., 2023), or by learning edits from instructions (Brooks et al., 2023; Geng et al., 2024; Sheynin et al., 2024; Fu et al., 2024; Huang et al., 2024). Large-scale editing datasets and generalist editors have raised single-turn quality (Zhao et al., 2024; Hui et al., 2025; Wei et al., 2025; Yu et al., 2025).

In multi-turn editing, errors accumulate when editors are applied to their previous outputs (Zhang et al., 2023a); SEED-Data-Edit provides multi-turn training data (Ge et al., 2024), and Zhou et al. (2025) reference both the source image and the previous result to maintain consistency. In all of these settings, the state passed between turns is a synthesized RGB image. PEAT instead passes an ASCII art sequence and a scene description, and synthesizes RGB once at the end.

Layout planning (Feng et al., 2023; Yang et al., 2024b) and grounded generation (Li et al., 2023; Xie et al., 2023) pass object boxes to synthesis, and spatial conditioning passes edge, segmentation, or depth maps (Zhang et al., 2023b; Mou et al., 2024; Zhao et al., 2023). Two lines of work are closest to ours. LLM-grounded Diffusion separates captioned layout planning from synthesis and updates the layout over multiple rounds (Lian et al., 2024); scene graph editing modifies an existing image by editing a symbolic graph while retaining source semantics (Dhamo et al., 2020). Both states are readable, but boxes and graphs show where objects are, not what shape they take. The PEAT grid encodes coarse foreground shape and intensity, is applied to an existing image with the source as the appearance reference, and shows the composition after each edit without rendering.

2.2 ASCII Art and Language-Editable Representations

Xu et al. (2010) distinguish tone-based and structure-based ASCII art, and Xu et al. (2017) synthesize it from photographs by extracting salient structures. ASCIIEval shows that recognition from raw characters depends strongly on the model and on input length (Jia et al., 2026), ArtPrompt shows that aligned LLMs struggle to read text rendered as ASCII art (Jiang et al., 2024), and ASCIIBench finds LLM-generated ASCII art hard to classify (Luo et al., 2025). PEAT therefore adds dedicated patch tokens and trains the LLM to generate and edit them.

Text2Space shows that training on ASCII layout construction improves spatial reasoning (Huang et al., 2026), and Mural generates images from ASCII layouts through a frozen LLM coupled to a diffusion expert (Jain et al., 2026). PEAT instead edits an existing image: it updates a persistent ASCII art state and keeps the source image as the appearance reference for final synthesis.

Language models also emit renderable graphics as TikZ (Belouadi et al., 2024), SVG (Rodriguez et al., 2025; Wu et al., 2025b), or stroke sequences (Vinker et al., 2025), and discrete image tokens from learned codebooks underlie autoregressive and masked generation (van den Oord et al., 2017; Esser et al., 2021; Ramesh et al., 2021; Chang et al., 2023). Vector code is built from geometric primitives and image tokens from learned latents; each PEAT token is a fixed character patch, so the grid maps foreground intensity directly and renders without a learned decoder.

3 Method

3.1 Overview and Notation

PEAT jointly updates an ASCII art sequence and a scene prompt, then combines the final state with the source image to synthesize the output. Figure 2(a) and (b) show the two LLM training stages, and (c) shows the image synthesis module. Stage 1 trains generation from scene prompts; Stage 2 trains state updates conditioned on edit instructions. At inference, we initialize the ASCII art state from the input image, apply the Stage 2 editing operation across turns, and synthesize an image from the final state.

Let I_0 denote the source image and N the number of edits. After t edits, the state consists of a scene prompt P_t and an ASCII art patch token sequence A_t. Instruction E_t specifies the next edit for $t = 0, \ldots, N-1$; the state at $t = 0$ is the initial state. The rendering function $R(\cdot)$ maps a patch token sequence back to a character grid and renders it as an image, so $R(A_t)$ visualizes the intermediate state. We denote the aggregated instruction history by $\tilde{E}_N$. Final synthesis uses I_0, P_N, $\tilde{E}_N$, and $R(A_N)$. The ASCII art sequence and prompt change across turns, while I_0 remains the appearance reference.

3.2 ASCII Art Representation

We convert the input foreground into a tone-based ASCII art grid and group adjacent characters into fixed-size patches to form an LLM token sequence. Figure 3 illustrates grid construction and

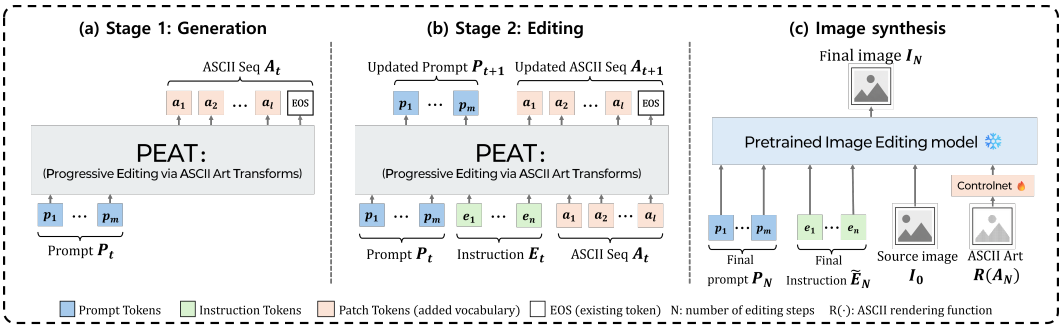


Figure 2: Overview of PEAT (Progressive Editing via ASCII Art Transforms). (a) Stage 1 trains generation of the ASCII art sequence A_t from the scene prompt P_t. (b) Stage 2 trains the LLM to update (P_t, A_t) to (P_{t+1}, A_{t+1}) given instruction E_t. (c) The image editing model receives the source image I_0, final scene prompt P_N, and aggregated instruction $\tilde{E}_N$; ControlNet receives the ASCII art rendering $R(A_N)$. The snowflake and flame indicate frozen and trainable components during training. All model parameters remain fixed during inference.

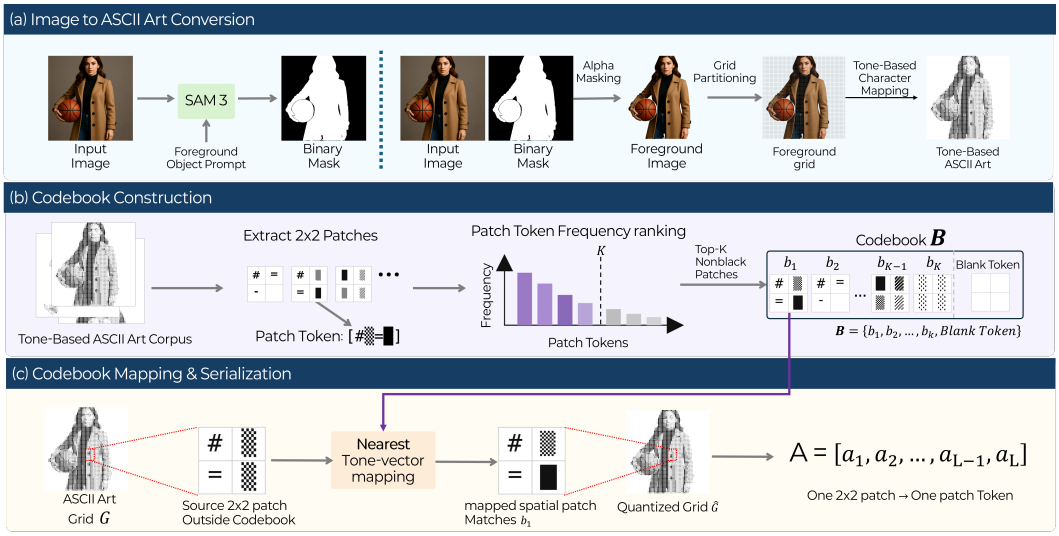


Figure 3: Construction of the ASCII art representation. (a) Foreground extraction and tone-based character assignment. (b) Frequency-based codebook construction from training grids. (c) Patch mapping and row-major serialization into a token sequence.

patch tokenization. We isolate the foreground with a SAM 3 mask (Carion et al., 2025), fill the background, convert the resulting image to grayscale, and partition it into an $H \times W$ grid. For each cell, we normalize its mean intensity and assign a character from an intensity-ordered set $\mathcal{C}$, producing a tone-based ASCII art grid $G \in \mathcal{C}^{H \times W}$. Character tones and cell positions represent the foreground intensity distribution and spatial composition. Appendix B.1 describes background filling and intensity normalization.

We group each non-overlapping $s \times s$ block of characters into one patch token, where s is the patch side length in characters. We use $s = 2$ in our experiments. A character set of size $|\mathcal{C}|$ admits $|\mathcal{C}|^{s^2}$ possible patches, each representing an arrangement of s^2 character tones. Relative to assigning one token per character, this reduces the ASCII art sequence length to $1/s^2$ of its original length.

To limit the patch vocabulary, we form a codebook $\mathcal{B} = \{b_1, \ldots, b_K\}$ from the K most frequent nonblank patches in the training data. Each patch p is represented by a vector $\mathbf{v}(p) \in \mathbb{R}^{s^2}$ that concatenates its character tones. A nonblank patch outside the codebook is mapped to the entry with

the smallest L_2 distance in tone space:

$$q(p) = \arg\min_{b \in \mathcal{B}} \|\mathbf{v}(p) - \mathbf{v}(b)\|_2. \tag{1}$$

Patches already in the codebook are retained, and an all-blank patch is assigned a separate token. The full patch vocabulary therefore contains $K + 1$ tokens.

Let $\widehat{G}$ denote the grid obtained by mapping the patches of G to the patch vocabulary. We serialize its patches in row-major order as $A = \mathcal{S}(\widehat{G}) = (a_1, \ldots, a_L)$, where $L = HW/s^2$. Here, H and W are multiples of s, and $\mathcal{S}$ maps each patch to its token in the vocabulary, including the blank token, and serializes the grid. Blank patches remain in the sequence to preserve grid positions. To obtain $R(A_t)$, we map tokens back to character patches, arrange them in row-major order, and render the grid with a fixed font. This image-to-sequence conversion is used to prepare training data and construct the initial state A_0 from I_0. The subscript in A_t denotes the editing turn; subsequent states are generated directly by the LLM rather than reconstructed from intermediate RGB images.

3.3 ASCII Art Generation and Editing

We train the LLM in two stages: ASCII art generation and instruction-guided editing. We use Qwen2.5 (Yang et al., 2024a) and train LoRA (Hu et al., 2022) parameters together with the embedding and output weights for the added tokens. Both stages share the patch vocabulary, and Stage 2 starts from the Stage 1 checkpoint. The original backbone weights remain frozen. Table 9 lists the full LLM identifiers and their roles. Appendix B.2 lists the LoRA modules, learning rates, batch sizes, epochs, and input and output formats.

Let $\mathcal{V}_{\text{patch}} = \mathcal{B} \cup \{b_{\text{blank}}\}$ denote the full patch vocabulary, including the blank patch. We add these tokens to the LLM vocabulary and initialize their embeddings from their tone patterns. For each b, we standardize $\mathbf{v}(b)$ using the per-dimension mean and standard deviation over $\mathcal{V}_{\text{patch}}$, obtaining $\mathbf{z}(b) \in \mathbb{R}^{s^2}$. We project this vector to the hidden dimension h and match the statistics of existing embeddings:

$$\mathbf{u}(b) = \mathbf{z}(b) W_{\text{proj}}, \qquad \mathbf{e}(b) = \boldsymbol{\mu}_{\text{ref}} + \alpha\big(\mathbf{u}(b) - \bar{\mathbf{u}}\big). \tag{2}$$

The projection matrix $W_{\text{proj}} \in \mathbb{R}^{s^2 \times h}$ has orthonormal rows obtained by QR decomposition of a random matrix, and $\bar{\mathbf{u}}$ is the mean projected vector over $\mathcal{V}_{\text{patch}}$. The reference mean $\boldsymbol{\mu}_{\text{ref}}$ is computed from existing embeddings selected at evenly spaced vocabulary indices. The scalar α matches the RMS norms of the projected and reference embeddings after centering each set at its mean. For $s^2 \leq h$, the projection preserves distances between standardized tone vectors; the common scalar then rescales these distances. This initialization reflects patch tone patterns in the relative distances between initial embeddings.

In Stage 1 (Figure 2(a)), the model generates A_t from a scene prompt P_t. Each training example uses a scene prompt as context and the patch sequence converted from its paired image, followed by EOS, as the target. Appendix A.1 describes the scene–image pairs, and Table 11 specifies the training message format. We use the EOS token from the existing vocabulary.

In Stage 2 (Figure 2(b)), the inputs are the current prompt P_t, instruction E_t, and ASCII art sequence A_t. The model outputs the updated prompt P_{t+1}, ASCII art sequence A_{t+1}, and EOS in that order:

$$(P_{t+1}, A_{t+1}) = F_\psi(P_t, E_t, A_t), \tag{3}$$

Here, F_ψ is the editing LLM, and ψ denotes the trainable LoRA and added-token parameters. Both stages use teacher forcing with autoregressive cross-entropy loss on output tokens only; input context tokens are excluded from the loss. For a stage-specific dataset $\mathcal{D}$ with input x and target sequence y, we minimize the negative log-likelihood of completion tokens:

$$\mathcal{L}_{\text{SFT}}(\psi) = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \sum_{j=1}^{|y|} \log p_\psi(y_j \mid x, y_{<j}). \tag{4}$$

Stage 2 uses adjacent state pairs stored in the dataset, with target ASCII art converted from the corresponding edited image. Stage 1 targets contain ASCII art and EOS, whereas Stage 2 targets contain the updated prompt, ASCII art, and EOS. The prompt and ASCII art form a single output

sequence, so ASCII art tokens are conditioned on the preceding scene prompt tokens. Both parts use the same next-token prediction objective.

At inference, we initialize the state with A_0 from the source image and a prompt P_0 describing the initial scene. We keep the trained LLM parameters fixed and apply instructions sequentially, starting with E_0. The generated pair (P_{t+1}, A_{t+1}) becomes the input state for the next instruction. After N updates, we obtain (P_N, A_N) and use a separate LLM to aggregate the instruction history $E_0, \ldots, E_{N-1}$ for final synthesis. Intermediate ASCII art renderings allow state inspection, while RGB synthesis is performed on the final state.

3.4 Image Synthesis

Figure 2(c) shows how the final editing state conditions RGB image synthesis. We use GPT-4o (OpenAI, 2024) to aggregate the instruction history into a single edit instruction, $\tilde{E}_N = \mathcal{H}(E_0, \ldots, E_{N-1})$. Here, $\mathcal{H}$ denotes instruction aggregation. The prompt P_N describes the final scene, while $\tilde{E}_N$ describes the changes to apply to the source image. This aggregated instruction is constructed separately from the Stage 2 output. The Qwen-Edit-based image editing model (Wu et al., 2025a) receives I_0, P_N, and $\tilde{E}_N$, while the ASCII art rendering is supplied to ControlNet (Zhang et al., 2023b) as a spatial condition:

$$c_N = R(A_N), \qquad I_N = \mathcal{G}_{\theta,\phi}(I_0, P_N, \tilde{E}_N; c_N). \tag{5}$$

The synthesis operation $\mathcal{G}_{\theta,\phi}$ combines the image editing model θ and ControlNet ϕ, including encoding and sampling. ControlNet extracts spatial conditioning features from $R(A_N)$. The source image I_0 provides the appearance reference, and c_N guides the edited object layout and shape. Appendix B.4 describes instruction aggregation and the final text inputs.

During synthesis training, we freeze the image editing model parameters θ and optimize only the ControlNet parameters ϕ using the base model's training objective. At inference, both θ and ϕ remain fixed, and the model synthesizes one image from the final state produced by successive LLM edits.

4 Experiments

4.1 Experimental Setup

Data. Table 6 summarizes the training and multi-turn evaluation data. Stage 1 uses scene–ASCII art pairs, and Stage 2 uses editing transitions. Evaluation applies four instructions to each of 25 source images. Appendix A describes data generation, filtering, augmentation, and evaluation sequence construction. The separate representation selection experiments are detailed in Appendices C.2 and D.1.

Baselines. We compare PEAT with Qwen-Edit, Nano Banana, FLUX (FLUX.1-Kontext-dev), MTC, and OmniGen2. Each baseline passes the image generated at one turn to the next, whereas PEAT passes the ASCII art state and scene prompt while retaining the source image as the appearance reference. All methods use the same source images and instruction sequences.

Evaluation Metrics. We evaluate instruction following, source content preservation, and perceptual quality at each turn. CLIP-T measures semantic alignment between the generated image and the accumulated instructions, while CLIP-I measures visual similarity to the source image. Aug-CLIP (Kim et al., 2025) evaluates modification and preservation using the source image and target text. Without reference target images, we assess perceptual quality with MANIQA (Yang et al., 2022) and MUSIQ (Ke et al., 2021). CLIP-I measures whole-image similarity rather than isolating instruction following or preservation of unedited regions. Appendix C provides the evaluation details. Section 5.1 presents human ratings of the intermediate ASCII art states and their consistency with the final image.

Implementation Details. We use Qwen2.5 as the base LLM and train LoRA parameters together with the embedding and output weights for added patch tokens. ASCII art uses a 64×64 grid of eight characters, with each 2×2 patch represented by one token. The vocabulary contains $3{,}000$ nonblank

Table 1: Multi-turn editing performance. Bold and underline indicate the best and second-best unrounded scores.

Method	Round 1					Round 2				
	CLIP-I↑	CLIP-T↑	AugCLIP↑	MANIQA↑	MUSIQ↑	CLIP-I↑	CLIP-T↑	AugCLIP↑	MANIQA↑	MUSIQ↑
PEAT	0.922	**0.332**	0.924	<u>0.524</u>	<u>73.523</u>	0.880	**0.340**	<u>0.906</u>	<u>0.536</u>	73.503
Qwen-Edit	<u>0.935</u>	0.329	<u>0.933</u>	0.515	72.945	<u>0.881</u>	0.332	0.900	0.535	73.299
Nano Banana	**0.946**	<u>0.331</u>	**0.939**	0.473	71.255	**0.898**	0.335	**0.912**	0.454	70.186
FLUX	0.929	0.327	0.929	**0.530**	73.225	0.874	<u>0.336</u>	0.901	**0.542**	<u>73.758</u>
MTC	0.912	0.317	0.922	0.466	71.003	0.864	0.313	0.901	0.481	71.021
OmniGen2	0.899	0.328	0.906	0.503	**73.778**	0.833	0.323	0.871	0.493	**74.217**

Method	Round 3					Round 4				
	CLIP-I↑	CLIP-T↑	AugCLIP↑	MANIQA↑	MUSIQ↑	CLIP-I↑	CLIP-T↑	AugCLIP↑	MANIQA↑	MUSIQ↑
PEAT	<u>0.862</u>	0.344	<u>0.891</u>	**0.534**	73.482	<u>0.846</u>	**0.342**	**0.881**	**0.534**	**73.522**
Qwen-Edit	0.841	**0.345**	0.877	<u>0.530</u>	<u>73.676</u>	0.797	<u>0.342</u>	0.848	<u>0.516</u>	<u>73.324</u>
Nano Banana	**0.869**	0.341	**0.894**	0.433	69.001	**0.850**	0.338	<u>0.878</u>	0.416	67.936
FLUX	0.818	<u>0.344</u>	0.865	0.509	72.383	0.794	0.335	0.837	0.497	72.394
MTC	0.827	0.312	0.875	0.483	70.638	0.800	0.312	0.858	0.482	70.293
OmniGen2	0.778	0.333	0.835	0.471	**74.021**	0.747	0.327	0.795	0.443	73.235

patches and one blank token, giving $1{,}024$ patch tokens per grid. Final synthesis uses a Qwen-Edit-based image editing model with ControlNet for ASCII art conditioning. Appendix Table 9 lists the full LLM identifiers and roles. Representation selection is described in Appendix D.1, and initialization and training settings in Appendix B.

4.2 Multi-Turn Editing Performance

Table 1 compares editing performance across four turns. Nano Banana leads in CLIP-I and AugCLIP at the first turn, whereas PEAT achieves the highest AugCLIP, MANIQA, and MUSIQ at the final turn. Its final-turn CLIP-T is also highest before rounding. The change in rankings distinguishes initial editing performance from performance after successive edits.

CLIP-I and AugCLIP decrease across turns for PEAT, but their first-to-last-turn declines are the smallest among the compared methods. CLIP-T remains above its first-turn value, and MANIQA and MUSIQ show no sustained decline. These trends are consistent with deferring synthesis to the final state; Section 5.2 separates the contribution of this deferral from that of the ASCII art state. Nano Banana has the highest CLIP-I at every turn, but its MANIQA and MUSIQ decrease. Source similarity alone therefore does not capture the full editing outcome; instruction alignment and perceptual quality also matter. PEAT combines high final-turn AugCLIP with high reference-free quality scores. Section 5.2 compares sequential RGB editing, prompt-only synthesis, and ASCII art state updates.

5 Analysis

5.1 Human Evaluation

Fourteen evaluators rated the ASCII art generation and editing results of PEAT on 25 four-turn sequences. At each turn, they assessed whether the grayscale ASCII art reflected the current instruction and preserved the previous state outside the requested changes. They also rated how well the final image followed the structure, intensity, and shape of the last ASCII art state. All criteria used a five-point scale; Appendix C.3 provides the rubric and aggregation procedure.

Table 2 shows similar mean ratings for current-edit success and previous-state preservation across the four turns, with no sustained decline. Final-image consistency averages 4.33, indicating that

Table 2: Human evaluation of PEAT: mean ratings on a five-point scale. Final-image consistency is rated once, after T4.

Criterion	T1	T2	T3	T4
Current-edit success	4.47	4.41	4.50	4.48
Previous-state preservation	4.40	4.40	4.49	4.46
Final-image consistency	—	—	—	4.33

Table 3: Final-turn performance by editing configuration. Bold marks the best scores.

Metric	Sequential RGB	Prompt-only	PEAT
CLIP-I↑	0.797	**0.847**	0.846
CLIP-T↑	0.342	0.337	**0.342**
AugCLIP↑	0.848	**0.882**	0.881
MANIQA↑	0.516	0.533	**0.534**
MUSIQ↑	73.324	73.324	**73.522**

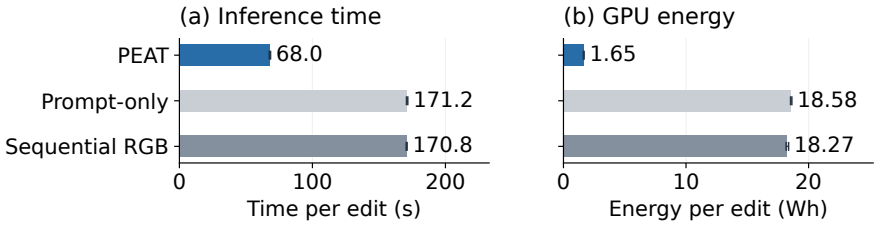


Figure 4: Per-edit time and GPU energy. Bars show means; error bars show standard deviations over 100 edits for Sequential RGB and PEAT, and 25 calls for Prompt-only.

evaluators generally found the final image consistent with the visual features of the last ASCII art state. These ratings complement the automatic RGB evaluation by assessing edits and preservation within the intermediate state, as well as its visual consistency with final synthesis.

5.2 Intermediate State Updates and Image Synthesis

Table 3 compares final-turn performance for three configurations that differ in their intermediate state and synthesis schedule. Sequential RGB applies Qwen-Edit four times, passing each output to the next turn. Prompt-only aggregates the instructions and calls Qwen-Edit once, without updating an intermediate visual state. PEAT updates the ASCII art and scene prompt four times, then synthesizes once using the final state and aggregated instruction. Prompt-only scores higher than sequential RGB in CLIP-I, AugCLIP, and MANIQA, showing that synthesis after all instructions can also yield strong preservation-related scores. PEAT and prompt-only have similar CLIP-I and AugCLIP, with prompt-only slightly higher in the displayed values. PEAT has higher CLIP-T and MUSIQ, while MANIQA is similar. Alongside this final-image performance, PEAT provides an intermediate spatial state that can be inspected and passed to the next edit. The comparison thus examines final quality together with the ability to maintain an explicit state throughout editing. Section 5.3 distinguishes per-edit cost from total cost through final image generation.

5.3 Inference Efficiency

Measurement scope. We distinguish per-edit cost from total cost through final image generation. Figure 4 shows mean time and GPU energy per edit. Figure 5 compares final-turn AugCLIP with mean total cost for processing four instructions and producing the final image. Appendix Table 14 gives the corresponding cost means and standard deviations. We compare methods within each measurement scope. GPU energy is measured with the NVML counter and excludes remote API energy.

Per-edit cost. In Figure 4, PEAT uses 60.2% less time and 90.9% less GPU energy per edit than sequential RGB editing with Qwen-Edit; because both configurations run four edits per sequence, the same ratios hold per sequence. A single prompt-only call costs about as much as one sequential Qwen-Edit edit (171.2 s, 18.6 Wh), and the measured per-edit time and energy of PEAT (68.0 s, 1.65 Wh) are below both.

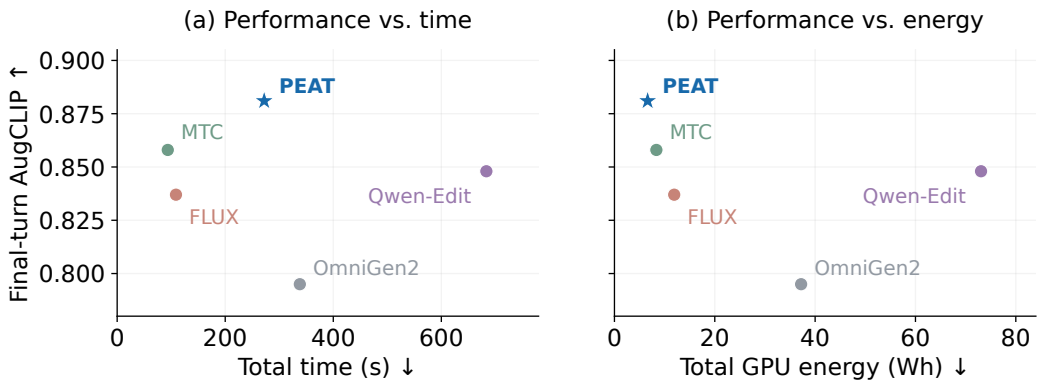


Figure 5: Final-turn AugCLIP versus total cost through final image generation, averaged over 25 four-turn sequences. Detailed cost statistics are in Appendix Table 14.

Total cost and final quality. In Figure 5(a), FLUX and MTC take less time than PEAT but have lower final-turn AugCLIP. PEAT uses less time and GPU energy than Qwen-Edit and OmniGen2. Among the measured editors, it combines the highest final-turn AugCLIP with the lowest GPU energy (Figure 5(b)). Its energy advantage is shared across these comparisons, while the time trade-off depends on the editor.

6 Conclusion

We propose PEAT, a framework that uses ASCII art as a visual bottleneck to separate successive scene edits from final image synthesis. The key is to distinguish the editing state updated by instructions from the source appearance reference. Our codebook-based ASCII art patch representation connects the character grid users inspect with the token sequence the LLM edits. The LLM jointly updates the scene prompt to pass spatial and semantic information across turns. Over four editing turns, PEAT loses the least CLIP-I and AugCLIP among six editors and ends with the best AugCLIP, MANIQA, and MUSIQ, even though other editors lead at the first turn. Human raters score current-edit success and preservation of the previous state between 4.4 and 4.5 out of 5 at every turn, and rate the final image consistent with the last ASCII art state at 4.33. Our ablation shows that synthesizing once after aggregating the instructions accounts for much of the preservation gain; maintaining the ASCII art state on top of it keeps CLIP-I and AugCLIP within 0.001 of prompt-only synthesis while raising CLIP-T and exposing an inspectable state. Per edit, PEAT uses 60.2% less time and 90.9% less GPU energy than sequential Qwen-Edit, and its measured time and energy are also below those of a single prompt-only synthesis call; lighter editors such as FLUX.1-Kontext and MTC remain faster in wall-clock time. Multi-turn editing therefore benefits from a discrete, inspectable editing state that is separate from the appearance reference, not only from fewer synthesis steps.

Limitations and Future Work

Tone-based ASCII art encodes foreground shape and intensity, leaving color, texture, background, and style to the scene prompt and final image editing model. Foreground extraction errors affect the initial state, while the fixed 64×64 grid and top-K codebook quantization limit fine shape detail. Our evaluation focuses on object arrangement and shape edits and does not establish performance on complex multi-object scenes with occlusion. Each turn still generates $1{,}024$ patch tokens and a scene prompt, which is why PEAT takes more time than prompt-only synthesis; the cost balance depends on sequence length, the number of turns, instruction aggregation, and RGB synthesis cost. Adaptive grids and different patch sizes could balance spatial detail and sequence length. Future work will examine this separation of editing state and appearance reference in other symbolic representations, video keyframes, and 3D scene composition.

AI USE STATEMENT

We use generative AI tools to assist with writing and polishing the manuscript, retrieve and discover relevant literature, and generate synthetic datasets.

ETHICS STATEMENT

All training and evaluation images are synthesized from text prompts with publicly available generation models; no photographs of real people or private data are used, and foreground masks are extracted from these synthetic images only. The human evaluation collected ratings of edited outputs from fourteen volunteer evaluators; no personal information was recorded, and the rubric is given in Appendix C.3. As with any image editing system, PEAT could be used to alter images in misleading ways; the method edits only object layout, shape, and intensity of a synthetic foreground and inherits the safety filters of the underlying editing model. We release no model that generates faces or identities beyond the capabilities of the public base models we build on.

REPRODUCIBILITY STATEMENT

We describe the ASCII art representation, codebook construction, and serialization in Section 3.2 and Appendix B.1, and the token initialization and two-stage training in Section 3.3 and Appendix B.2. Dataset construction, filtering, and split statistics are given in Appendix A; the synthesis module, instruction aggregation, and final text inputs in Appendix B.4; and all model identifiers in Table 9. Evaluation protocols, baseline settings, and the efficiency measurement setup are specified in Appendix C, including hardware, seeds, and the GPU energy measurement method. We will release the data pipeline, training and inference code, the ASCII art codebook, and trained adapter and ControlNet weights upon publication.

REFERENCES

Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f7641940c7dd9e5de58c20e39586eb64-Paper-Conference.pdf.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. InstructPix2Pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18392–18402, 2023. URL https://openaccess.thecvf.com/content/CVPR2023/html/Brooks_InstructPix2Pix_Learning_To_Follow_Image_Editing_Instructions_CVPR_2023_paper.html.

Mingdeng Cao, Xintao Wang, Zhongang Qi, Ying Shan, Xiaohu Qie, and Yinqiang Zheng. MasaCtrl: Tuning-free mutual self-attention control for consistent image synthesis and editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22560–22570, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Cao_MasaCtrl_Tuning-Free_Mutual_Self-Attention_Control_for_Consistent_Image_Synthesis_and_ICCV_2023_paper.html.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. Sam 3: Segment anything with concepts, 2025. URL https://arxiv.org/abs/2511.16719.

Huiwen Chang, Han Zhang, Jarred Barber, Aaron Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Patrick Murphy, William T. Freeman, Michael Rubinstein, Yuanzhen Li, and Dilip

Krishnan. Muse: Text-to-image generation via masked generative transformers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 4055–4075. PMLR, 2023. URL https://proceedings.mlr.press/v202/chang23b.html.

Guillaume Couairon, Jakob Verbeek, Holger Schwenk, and Matthieu Cord. DiffEdit: Diffusion-based semantic image editing with mask guidance. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=3lge0p5o-M-.

Helisa Dhamo, Azade Farshad, Iro Laina, Nassir Navab, Gregory D. Hager, Federico Tombari, and Christian Rupprecht. Semantic image manipulation using scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12873–12883, 2021.

Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Xuehai He, S. Basu, Xin Eric Wang, and William Yang Wang. LayoutGPT: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3a7f9e485845dac27423375c934cb4db-Abstract.html.

Tsu-Jui Fu, Wenze Hu, Xianzhi Du, William Wang, Yinfei Yang, and Zhe Gan. Guiding instruction-based image editing via multimodal large language models. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f0e91b1314fa5eabf1d7ef6d1561ecec-Paper-Conference.pdf.

Yuying Ge, Sijie Zhao, Chen Li, Yixiao Ge, and Ying Shan. SEED-Data-Edit technical report: A hybrid dataset for instructional image editing, 2024. URL https://arxiv.org/abs/2405.04007.

Zigang Geng, Binxin Yang, Tiankai Hang, Chen Li, Shuyang Gu, Ting Zhang, Jianmin Bao, Zheng Zhang, Houqiang Li, Han Hu, Dong Chen, and Baining Guo. Instructdiffusion: A generalist modeling interface for vision tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12709–12720, 2024.

Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross-attention control. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=_CDixzkzeyb.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Shiyuan Huang, Li Liu, Jincheng He, and Leilani H. Gilpin. Learning to draw ASCII improves spatial reasoning in language models, 2026. URL https://arxiv.org/abs/2604.14641.

Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, and Ying Shan. Smartedit: Exploring complex instruction-based image editing with multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8362–8371, 2024.

Mude Hui, Siwei Yang, Bingchen Zhao, Yichun Shi, Heng Wang, Peng Wang, Cihang Xie, and Yuyin Zhou. Hq-edit: A high-quality dataset for instruction-based image editing. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/b639c526a942618f1dba2b7a5fa0bac0-Paper-Conference.pdf.

Achin Jain, Jie An, Siddharth Chaudhary, and Davide Modolo. Mural: Transferring LLM knowledge to image generation via Mixture-of-Transformers, 2026. URL https://arxiv.org/abs/2606.29013.

Qi Jia, Xiang Yue, Shanshan Huang, Ziheng Qin, Yizhu Liu, Bill Yuchen Lin, Yang You, and Guangtao Zhai. ASCIIEval: Benchmarking models' visual perception in text strings via ASCII art. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/63f5c95b1e6364c42075f913d84ccb73-Abstract-Conference.html.

Fengqing Jiang, Zhangchen Xu, Luyao Niu, Zhen Xiang, Bhaskar Ramasubramanian, Bo Li, and Radha Poovendran. ArtPrompt: ASCII art-based jailbreak attacks against aligned LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15157–15173, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.809. URL https://aclanthology.org/2024.acl-long.809/.

Junjie Ke, Qifei Wang, Yilin Wang, Peyman Milanfar, and Feng Yang. MUSIQ: Multi-scale image quality transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5148–5157, 2021. URL https://openaccess.thecvf.com/content/ICCV2021/html/Ke_MUSIQ_Multi-Scale_Image_Quality_Transformer_ICCV_2021_paper.html.

Yoonjeon Kim, Soohyun Ryu, Yeonsung Jung, Hyunkoo Lee, Joowon Kim, June Yong Yang, Jaeryong Hwang, and Eunho Yang. Preserve or modify? context-aware evaluation for balancing preservation and modification in text-guided image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23474–23483, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Kim_Preserve_or_Modify_Context-Aware_Evaluation_for_Balancing_Preservation_and_Modification_CVPR_2025_paper.html.

Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. Gligen: Open-set grounded text-to-image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22511–22521, 2023.

Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. LLM-grounded Diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2305.13655.

Kerry Luo, Michael Fu, Joshua Peguero, Husnain Malik, Anvay Patil, Joyce Lin, Megan Van Overborg, Ryan Sarmiento, and Kevin Zhu. ASCIIBench: Evaluating language-model-based understanding of visually-oriented text, 2025. URL https://arxiv.org/abs/2512.04125. NeurIPS 2025 LLM Evaluation Workshop and Multimodal Algorithmic Reasoning Workshop.

Ron Mokady, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. NULL-Text inversion for editing real images using guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6038–6047, 2023. URL https://openaccess.thecvf.com/content/CVPR2023/html/Mokady_NULL-Text_Inversion_for_Editing_Real_Images_Using_Guided_Diffusion_Models_CVPR_2023_paper.html.

Chong Mou, Xintao Wang, Liangbin Xie, Yanze Wu, Jian Zhang, Zhongang Qi, and Ying Shan. T2i-adapter: Learning adapters to dig out more controllable ability for text-to-image diffusion models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(5):4296–4304, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i5.28226. URL http://dx.doi.org/10.1609/aaai.v38i5.28226.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8821–8831. PMLR, 2021. URL https://proceedings.mlr.press/v139/ramesh21a.html.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16175–16186, 2025.

Shelly Sheynin, Adam Polyak, Uriel Singer, Yuval Kirstain, Amit Zohar, Oron Ashual, Devi Parikh, and Yaniv Taigman. Emu Edit: Precise image editing via recognition and generation tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8871–8879, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Sheynin_Emu_Edit_Precise_Image_Editing_via_Recognition_and_Generation_Tasks_CVPR_2024_paper.html.

Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1921–1930, 2023. URL https://openaccess.thecvf.com/content/CVPR2023/html/Tumanyan_Plug-and-Play_Diffusion_Features_for_Text-Driven_Image-to-Image_Translation_CVPR_2023_paper.html.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

Yael Vinker, Tamar Rott Shaham, Kristine Zheng, Alex Zhao, Judith E. Fan, and Antonio Torralba. Sketchagent: Language-driven sequential sketch generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23355–23368, 2025.

Cong Wei, Zheyang Xiong, Weiming Ren, Xinrun Du, Ge Zhang, and Wenhu Chen. Omniedit: Building image editing generalist models through specialist supervision. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/0113ef4642264adc2e6924a3cbbdf532-Paper-Conference.pdf.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, et al. Qwen-Image technical report, 2025a. URL https://arxiv.org/abs/2508.02324.

Ronghuan Wu, Wanchao Su, and Jing Liao. Chat2svg: Vector graphics generation with large language models and image diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23690–23700, 2025b.

Jinheng Xie, Yuexiang Li, Yawen Huang, Haozhe Liu, Wentian Zhang, Yefeng Zheng, and Mike Zheng Shou. Boxdiff: Text-to-image synthesis with training-free box-constrained diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7452–7461, 2023.

Xuemiao Xu, Linling Zhang, and Tien-Tsin Wong. Structure-based ASCII art. *ACM Transactions on Graphics*, 29(4):52:1–52:9, 2010. doi: 10.1145/1778765.1778789. URL https://ttwong12.github.io/papers/asciiart/asciiart.html.

Xuemiao Xu, Linyuan Zhong, Minshan Xie, Xueting Liu, Jing Qin, and Tien-Tsin Wong. Ascii art synthesis from natural photographs. *IEEE Transactions on Visualization and Computer Graphics*, 23(8):1910–1923, August 2017. ISSN 1077-2626. doi: 10.1109/tvcg.2016.2569084. URL http://dx.doi.org/10.1109/TVCG.2016.2569084.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report, 2024a. URL https://arxiv.org/abs/2412.15115.

Ling Yang, Zhaochen Yu, Chenlin Meng, Minkai Xu, Stefano Ermon, and Bin Cui. Mastering text-to-image diffusion: Recaptioning, planning, and generating with multimodal LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56704–56721. PMLR, 2024b. URL https://proceedings.mlr.press/v235/yang24ai.html.

Sidi Yang, Tianhe Wu, Shuwei Shi, Shanshan Lao, Yuan Gong, Mingdeng Cao, Jiahao Wang, and Yujiu Yang. MANIQA: Multi-dimension attention network for no-reference image quality assessment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1191–1200, 2022. URL https://openaccess.thecvf.com/content/CVPR2022W/NTIRE/html/Yang_MANIQA_Multi-Dimension_Attention_Network_for_No-Reference_Image_Quality_Assessment_CVPRW_2022_paper.html.

Qifan Yu, Wei Chow, Zhongqi Yue, Kaihang Pan, Yang Wu, Xiaoyang Wan, Juncheng Li, Siliang Tang, Hanwang Zhang, and Yueting Zhuang. Anyedit: Mastering unified high-quality image editing for any idea. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26125–26135, 2025.

Kai Zhang, Lingbo Mo, Wenhu Chen, Huan Sun, and Yu Su. MagicBrush: A manually annotated dataset for instruction-guided image editing. In *Advances in Neural Information Processing Systems*, volume 36, 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/64008fa30cba9b4d1ab1bd3bd3d57d61-Abstract-Datasets_and_Benchmarks.html.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3836–3847, 2023b. URL https://openaccess.thecvf.com/content/ICCV2023/html/Zhang_Adding_Conditional_Control_to_Text-to-Image_Diffusion_Models_ICCV_2023_paper.html.

Haozhe Zhao, Xiaojian Ma, Liang Chen, Shuzheng Si, Rujie Wu, Kaikai An, Peiyu Yu, Minjia Zhang, Qing Li, and Baobao Chang. UltraEdit: Instruction-based fine-grained image editing at scale. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/05a30a0fc9e6bacdd3abd4ca8508a9e6-Abstract-Datasets_and_Benchmarks_Track.html.

Shihao Zhao, Dongdong Chen, Yen-Chun Chen, Jianmin Bao, Shaozhe Hao, Lu Yuan, and Kwan-Yee K. Wong. Uni-controlnet: All-in-one control to text-to-image diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11127–11150. Curran Associates, Inc., 2023. doi: 10.52202/075280-0491. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/2468f84a13ff8bb6767a67518fb596eb-Paper-Conference.pdf.

Zijun Zhou, Yingying Deng, Xiangyu He, Weiming Dong, and Fan Tang. Multi-turn consistent image editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15792–15801, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Zhou_Multi-turn_Consistent_Image_Editing_ICCV_2025_paper.html.

APPENDIX INDEX

Appendix A: Dataset Construction 15
Appendix B: Implementation Details 17
Appendix C: Evaluation Protocols 20
Appendix D: Additional Representation Analyses 23
Appendix E: Qualitative Results and Failure Cases 25

A DATASET CONSTRUCTION

A.1 INITIAL SCENE GENERATION

We construct training data for ASCII art generation from scene descriptions paired with corresponding images. The initial scenes consist of single-subject scenes (a human, an animal, or an object) as well as scenes combining two such subjects; Table 5 summarizes the resulting seven domains. Each scene prompt P_0 is fed to an image generation model to produce an image I_0; we use Qwen-Image-Lightning for this initial image generation. Table 4 lists the role of each model along with its checkpoint filename. We then extract the foreground from each generated image and apply the conversion described in Section B.1 to obtain an ASCII art sequence A_0. Stage 1 learns the spatial layout corresponding to each scene prompt from (P_0, A_0) pairs, and the RGB image itself is never used as input to the LLM.

Table 4: Models and Lightning checkpoints used for dataset construction.

Role	Model and checkpoint
Initial scene generation	Qwen-Image-Lightning `Qwen-Image-Lightning-4steps-V1.0.safetensors`
Subsequent image editing	Qwen-Image-Edit-Lightning `Qwen-Image-Edit-Lightning-4steps-V1.0.safetensors`

Table 5: Scene domains used for generation and editing.

Domain	Initial generation	Editing sequences
Human	Yes	Yes
Animal	Yes	No
Object	Yes	No
Human–object	Yes	Yes
Animal–object	Yes	No
Human–animal	Yes	Yes
Object–object	Yes	No

A.2 EDITING SEQUENCE CONSTRUCTION

We construct editing data from the human, human–object, and human–animal domains. We define the state at step t as $S_t = (P_t, I_t, A_t)$ and use a VLM to inspect the current image and compose the next instruction E_t. Instructions span five types: object addition, object removal, object replacement, pose change, and attribute change. An image editing model produces the edited image I_{t+1}, from which we obtain a scene description P_{t+1} and an ASCII art sequence A_{t+1} converted from the image. We use Qwen-Image-Edit-Lightning for these subsequent image generations. Starting from each initial scene, we apply up to three edits, yielding a sequence of state transitions:

$$S_0 \xrightarrow{E_0} S_1 \xrightarrow{E_1} S_2 \xrightarrow{E_2} S_3, \tag{6}$$

where E_t denotes the instruction applied to S_t, following the notation in the main text.

Each pair of consecutive states yields a Stage 2 training example $((P_t, E_t, A_t), (P_{t+1}, A_{t+1}))$. The target ASCII art is always converted from the edited image; ASCII art previously generated by the

Table 6: Training and evaluation data. Stage 2 includes reverse-edit augmentation.

Data	Size	Coverage	Editing turns
Stage 1 training	50,483 scene–ASCII art pairs	7 domains	—
Stage 2 training	59,344 transitions	3 domains	Up to 3
Multi-turn evaluation	25 sequences / 25 source images	5 styles	4

LLM is never used as a target. All states in Stage 2 are represented with the codebook constructed in Stage 1. Note that the RGB editing sequences serve only to provide supervision for state transitions during data construction. This differs from the inference-time execution of PEAT, which updates the ASCII art state directly and synthesizes an RGB image only at the final step.

A.3 Quality Control and Filtering

We manually inspect all generated and edited images and remove samples in which hallucinations are identified. Foregrounds are extracted with SAM3, and the extraction results are converted into ASCII art grids, on which we check patch occupancy. This filter is not intended to measure the semantic accuracy of the foreground; rather, it screens out results in which the foreground is either too small or occupies most of the canvas.

Let L denote the total number of patches and n_{blank} the number of blank patches, where a patch is considered blank only if all of its characters are blank; a patch containing at least one nonblank character is classified as nonblank. Defining τ_{nonblank} and τ_{blank} as the minimum occupancy ratios of nonblank and blank patches, respectively, a sample is retained if

$$\frac{L - n_{\mathrm{blank}}}{L} \geq \tau_{\mathrm{nonblank}}, \qquad \frac{n_{\mathrm{blank}}}{L} \geq \tau_{\mathrm{blank}}. \tag{7}$$

We set $\tau_{\mathrm{nonblank}} = 0.25$ and $\tau_{\mathrm{blank}} = 0.0625$ in all experiments. With a 64×64 grid and 2×2 patches, $L = 1{,}024$, so we retain samples with at least 256 nonblank patches and at least 64 blank patches. Filtering is applied to the grids before quantization with the top-K codebook, and the data splits and the codebook are constructed from the samples that pass the filter.

A.4 Reverse-Edit Augmentation

We augment Stage 2 with reverse-edit examples, in which the order of the pre- and post-edit states is swapped. Given a forward transition with input (P_t, E_t, A_t) and output (P_{t+1}, A_{t+1}), the corresponding reverse example is constructed as

$$(P_{t+1}, E_t^{\mathrm{rev}}, A_{t+1}) \longmapsto (P_t, A_t), \tag{8}$$

where E_t^{rev} describes the change that restores the post-edit state to the pre-edit state. Since the reverse target is the original state from the forward example, both directions share the same pair of states and differ only in the direction of the instruction. Notably, reversing a removal requires re-adding the removed object, so the reverse instruction must carry information about that object from the previous state. Constructing reverse instructions therefore goes beyond simply inverting the verb of the forward instruction. The total number of transitions, including the augmented examples, is reported in Section A.5.

A.5 Dataset Statistics and Splits

Table 7 summarizes the size of the training data for each stage. Stage 1 comprises $50{,}483$ scene–ASCII art pairs, and Stage 2 comprises $59{,}344$ transitions, including reverse-edit augmentation. The per-turn counts for Stage 2 indicate the number of transitions originating from each editing depth; they do not correspond to the number of independent source images or complete sequences. Since transitions from the first turn are by far the most numerous, overall performance should be interpreted separately from performance at each editing depth.

Multi-turn evaluation sequences. Source images for evaluation are drawn from held-out Stage 2 images that are not used for training. We randomly select five images from each of five styles (anime,

Table 7: Training dataset statistics, including reverse-edit augmentation for Stage 2.

Training stage	Unit / originating turn	Count
Stage 1	Scene–ASCII art pairs	50,483
Stage 2	Transitions from turn 1	45,351
	Transitions from turn 2	13,424
	Transitions from turn 3	569
	Total transitions	59,344

Table 8: ASCII art conversion and codebook settings.

Setting	Value
Conversion image size	$1{,}024 \times 1{,}024$ pixels
ASCII art grid size	64×64 cells
Cell size before conversion	16×16 pixels
Character set size	8
Patch size	2×2 cells
Possible patch patterns	$8^4 = 4{,}096$
Retained nonblank patterns	$K = 3{,}000$
Patch vocabulary size	3,001, including one blank token
Patch sequence length	$L = 1{,}024$
Serialization order	Row-major within and across patches
Out-of-codebook mapping	Nearest nonblank pattern in tone space

realistic, 3D render, pixel art, and watercolor), yielding 25 images in total. For each image, we use an LLM to compose four instructions within the range of editing types covered by the training data, and apply them sequentially. The evaluation set thus consists of 25 four-turn sequences, resulting in 100 per-turn outputs evaluated for each method. Table 6 summarizes the scale of the training and evaluation data.

The distinction between the 500 evaluation images used in the representation selection experiments and the final multi-turn editing evaluation set is described in Section C.2.

B Implementation Details

B.1 ASCII Art Conversion and Codebook

We map the brightness of the input foreground to character indices and group adjacent characters into patch tokens. Table 8 summarizes the settings for the conversion and the codebook. The character set consists of, from darkest to lightest, full block, dark shade, medium shade, light shade, #, =, −, and _. The first four characters correspond to Unicode U+2588, U+2593, U+2592, and U+2591, respectively, and the last character, _, denotes blank. The tone value of each character is defined as its index $j \in \{0, \ldots, 7\}$ in this ordering.

We resize the foreground extraction result to a $1{,}024 \times 1{,}024$ RGBA image and treat pixels with a positive alpha value as foreground. The background is filled with the average color of the three foreground pixels with the highest mean RGB brightness, and the image is then converted to grayscale. If there are fewer than three foreground pixels, the average of all foreground pixels is used instead, and if there is no foreground, the background is filled with white. We divide the grayscale image into 16×16-pixel cells and compute the mean brightness m_{ij} of each cell. Using the minimum and maximum brightness over the entire image, $m_{\min}$ and $m_{\max}$, the character index is obtained as

$$g_{ij} = \mathrm{clip}_{[0,7]} \left(\mathrm{round} \left[7 \frac{m_{ij} - m_{\min}}{m_{\max} - m_{\min} + 10^{-8}} \right] \right). \tag{9}$$

In our implementation, we use NumPy's round-half-to-even rounding.

The codebook is constructed by counting the occurrence frequency of nonblank patches in the training split. Ties in frequency are broken by selecting first the patch whose row-major character indices,

Table 9: LLM identifiers and roles.

Short name	Full model / checkpoint	Role and adaptation
Qwen2.5	Qwen2.5-3B-Instruct (Yang et al., 2024a) `Qwen/Qwen2.5-3B-Instruct`	Stage 1 generation and Stage 2 editing. LoRA and added-token weights are trained; original backbone weights are frozen.
GPT-4o	GPT-4o (OpenAI, 2024) API snapshot: to be verified	Aggregates the instruction history into $\tilde{E}_N$ for final image synthesis.

read as a base-8 integer, yield the smaller value. The top $3{,}000$ patches are kept as is, and each remaining nonblank patch is mapped to the codebook entry with the smallest squared L_2 distance between character index vectors. Blank patches, in which all four characters are _, are excluded from this mapping and preserved as a separate token. Hence, Top-3000 refers to the size of the nonblank codebook rather than the full vocabulary size. The same codebook and conversion rules are applied to the validation and test splits. For each patch, we concatenate its four characters in row-major order, prepend a single space, and register the resulting string as a single token in the LLM tokenizer. Since patches in the grid are also arranged in row-major order, every position, including blank regions, corresponds to one of the $1{,}024$ patch tokens.

B.2 Token Initialization and LLM Training

In the main text, we refer to the base LLM as Qwen2.5 and the instruction consolidation model as GPT-4o. Table 9 lists the full model names and their roles. Stage 2 does not use a separate base model; instead, training continues from the Stage 1 checkpoint.

We add the $3{,}001$ patch tokens described above to the vocabulary of Qwen2.5-3B-Instruct (Yang et al., 2024a). The reference embeddings used for initialization consist of up to $4{,}096$ rows selected at uniform intervals over the index range of the original vocabulary. Denoting this set by $\mathcal{R}$, we compute the reference mean and centered RMS norm as

$$\boldsymbol{\mu}_{\mathrm{ref}} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \mathbf{e}_r, \qquad \rho_{\mathrm{ref}} = \sqrt{\frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \|\mathbf{e}_r - \boldsymbol{\mu}_{\mathrm{ref}}\|_2^2}. \tag{10}$$

The reference rows are selected deterministically rather than sampled at random, and are thus independent of the random seed used for initialization.

We divide the character indices of each patch by 7 and standardize each dimension over the entire patch vocabulary, including the blank token. We use the population standard deviation, clamped to a minimum of 10^{-6}. We obtain $Q \in \mathbb{R}^{h \times 4}$ with orthonormal columns by QR decomposition of a Gaussian random matrix and set $W_{\mathrm{proj}} = Q^\top$. The standardized vectors are then mapped into the embedding space using W_{proj}. We center the projected vectors, rescale them so that their centered RMS norm equals ρ_{ref}, and add $\boldsymbol{\mu}_{\mathrm{ref}}$. This corresponds to setting α in Section 3.3 to the ratio of the two centered RMS norms. The default initialization seed is $1{,}729$. If the input embedding and output head weights are tied, they are initialized once; otherwise, each weight is initialized using its own reference statistics. For the matched random initialization baseline, we replace the tone vectors with Gaussian random vectors and apply the same mean and RMS matching.

In Stage 1, the model is trained to generate ASCII art sequences from scene prompts, and Stage 2 continues training from the Stage 1 checkpoint. In both stages, we update only the LoRA (Hu et al., 2022) parameters and the weights corresponding to the added tokens. LoRA is applied to the query, key, value, and output projections of the attention layers, while the original vocabulary and the remaining backbone parameters are kept frozen. Stage 2 reuses the patch vocabulary and adapter configuration from Stage 1.

Table 10 summarizes the default settings for both training stages. Both stages use the Qwen chat template and are trained with a completion-only autoregressive cross-entropy loss, in which the input context is excluded from the loss. The Stage 1 output consists of the ASCII art sequence followed by EOS, while the Stage 2 output consists of the scene prompt following `Updated prompt:`, the ASCII art sequence following `Updated ASCII:`, and EOS. We use the existing EOS token of the

Table 10: Default LLM training settings for a single device.

Setting	Stage 1	Stage 2
Initial checkpoint	Qwen2.5-3B-Instruct	Stage 1 checkpoint
LoRA rank / alpha	128 / 128	Inherited
LoRA dropout	0	Inherited
Optimizer	AdamW	AdamW
Learning rate	3×10^{-4}	10^{-4}
Weight decay	0.001	0.001
Learning rate schedule	Constant	Constant
Warmup steps	0	0
Epochs	3	2
Batch size per device	1	1
Gradient accumulation steps	8	8
Effective batch size	8	8
Maximum sequence length	3,072	4,096
Gradient clipping norm	1.0	1.0
Precision	BF16	BF16
Attention implementation	SDPA	SDPA
Gradient checkpointing	Enabled	Disabled
Training seed	42	42

original tokenizer. The maximum sequence length applies to the full input context and output combined, and is distinct from the length of the ASCII art patch tokens alone. We do not use sequence packing, and checkpoints are saved at every epoch.

We apply the chat template of the Qwen tokenizer to the training inputs. The formats below show the content of the user message and the assistant completion, where items in angle brackets are replaced with the values of each example. Each patch token in the ASCII art sequence is preceded by a single space, and the existing EOS token is appended at the end of the completion.

Table 11: Supervised training templates. Angle brackets indicate substituted values.

Stage	User message	Assistant completion

Stage 1	`Generate the ASCII representation of the following scene:`
	
	`<scene prompt>`
		`<ASCII art sequence><EOS>`

Stage 2	`Current prompt:`
	`<scene prompt>`
	
	`Edit instruction:`
	`<instruction>`
	
	`Current ASCII:`
	`<ASCII art sequence>`
		`Updated prompt:`
		`<target description>`
		
		`Updated ASCII:`
		`<target sequence><EOS>`

Stage 2 takes the stored `prompt_t`, `edit_instruction`, and `ascii_t` as input and uses `prompt_t1` and `ascii_t1` as targets. Under teacher forcing, the model learns to predict the next token conditioned on the target prefix, and free-running predictions are not fed into subsequent training examples. The generation of scene prompts and instructions for data construction is independent of this training template and is described in Appendix A.2.

B.3 Decoding and State Updates

At each editing turn, the LLM takes the previous scene prompt P_t, the ASCII art sequence A_t, and the current editing instruction E_t as input, and generates the updated pair (P_{t+1}, A_{t+1}). We use nucleus sampling ($p = 0.95$) for decoding and set the maximum generation length to 1,400 tokens.

The sum of the input length and the maximum generation length is limited to 4,096 tokens, and the tokenizer's EOS token serves as the stop token.

After removing trailing EOS and padding tokens from the generated output, we split it into the scene prompt and the ASCII art sequence using `Updated prompt:` and `Updated ASCII:` as delimiters. We then check the output format, whether the scene prompt is empty, and whether the ASCII sequence follows the fixed-patch format and contains only valid codebook entries. Outputs that reach the maximum generation length without an EOS token are also treated as failures.

Only when validation succeeds do we normalize the ASCII sequence and use (P_t, A_t) as the input state for the next turn. If validation fails, the previous state is retained and the remaining turns of that editing sequence are terminated. As a result, invalid outputs are never passed to subsequent turns.

B.4 Final Image Synthesis

The final editing instruction $\tilde{E}_N$ is constructed by providing GPT-4o with the ordered instruction history $(E_0, \ldots, E_{N-1})$. This consolidation step is separate from Stage 2's prediction of the scene prompt and ASCII art. The final synthesis, based on Qwen-Edit, takes as input the source image I_0, the scene prompt P_N, and the consolidated instruction $\tilde{E}_N$. The rendering $R(A_N)$ is provided to ControlNet as a spatial condition. The scene description specifies the final state, whereas the consolidated instruction specifies the changes to be applied to the source image.

C Evaluation Protocols

C.1 Baselines and Comparison Settings

MagicBrush (Zhang et al., 2023a) is a benchmark that pairs instructions with before- and after-edit images, and it distinguishes between single-turn and multi-turn evaluation. Single-turn evaluation uses the reference source of each edit as input, whereas multi-turn evaluation starts from the initial source and applies each subsequent instruction to the previous prediction. The multi-turn protocol, which compares the final result with the last reference target, also applies to PEAT, which synthesizes RGB from the final state.

The mask-free evaluation in MagicBrush uses L1, L2, CLIP-I, and DINO computed against the target image, and CLIP-T computed against the local target description. Here, the target image is not the original image but a reference that reflects the requested edit. These metrics measure agreement with the reference; we do not interpret them as fully separating the preservation of non-target regions from instruction following. Methods that take a mask as input are evaluated under a condition separate from that of mask-free methods.

Metrics for the main comparison. Tables 1 and 3 report CLIP-I, CLIP-T, AugCLIP, MANIQA, and MUSIQ. AugCLIP (Kim et al., 2025) jointly evaluates modification and preservation in the context of the source image and the target text. MANIQA (Yang et al., 2022) and MUSIQ (Ke et al., 2021) are reference-free image quality metrics and do not directly assess whether a specific instruction has been followed. This set of metrics is distinct from the SSIM, PSNR, and MSE used in the SDXL reconstruction experiments described below.

C.2 Representation Evaluation Protocol

The representation evaluation extracts a spatial condition from an image before any edit is applied and reconstructs the original image from that condition. Let P denote the scene prompt of the original image I and T the representation transform. We provide P and $T(I)$ to SDXL and compare the synthesized image $\widehat{I}$ with I. This comparison measures the reconstruction information conveyed by each representation under a given synthesis model, and it is separate from the LLM's editing performance in reflecting successive instructions. Conditions other than ASCII art are also extracted from the same original images.

Table 12 summarizes the settings of the two comparison experiments. For the comparison of character sets and codebook sizes, we train all nine combinations of $C \in \{8, 9, 10\}$ and $K \in$

Table 12: Protocol for representation selection and spatial condition comparison.

Setting	Representation selection	Condition comparison
Training images	5,000	10,000
Evaluation images	500	500
Training epochs	3	3
Synthesis backbone	SDXL	SDXL
Output resolution	$1{,}024 \times 1{,}024$	$1{,}024 \times 1{,}024$
Sampling steps	30	30
CFG scale	7.5	7.5
ASCII art grid size	64×64	64×64
ASCII art patch size	2×2	2×2
Character count C	$8, 9, 10$	8
Nonblank codebook size K	$1{,}000, 2{,}000, 3{,}000$	3,000

$\{1{,}000, 2{,}000, 3{,}000\}$. Because the grid and patch sizes are fixed, every combination represents an image with $1{,}024$ patch tokens. For the comparison of structural conditions, we compare the selected setting ($C = 8$, $K = 3{,}000$) with prompt-only, bounding box, mask, and edge conditions. Prompt-only serves as a baseline that provides no spatial condition. For each setting that uses a spatial condition, we freeze SDXL and train the corresponding T2I-Adapter (Mou et al., 2024). Within each comparison experiment, we use the same training and evaluation images and fix the number of training epochs and the sampling settings.

CLIP-I measures semantic consistency through the similarity between the image embeddings of the original and synthesized images. SSIM measures the structural similarity between the two images, while PSNR and MSE measure pixel-level reconstruction error. CLIP-T serves as an auxiliary metric that checks the alignment between the synthesized image and the scene prompt. Higher values of CLIP-I, CLIP-T, SSIM, and PSNR, and lower values of MSE, indicate better results under the corresponding criterion. When selecting the representation, we prioritize reconstruction metrics with respect to the original image over text alignment alone. Among the compared settings, $C = 8$ and $K = 3{,}000$ achieves the best CLIP-I, SSIM, PSNR, and MSE, and we therefore adopt it as the representation setting for subsequent ASCII art editing. Detailed results are presented in Tables 15 and 16.

In this experiment, the original image serves as the reconstruction target. In final editing, by contrast, parts of the original image must change according to the request, so SSIM or MSE between the edited image and the entire original image alone cannot determine whether an edit succeeded. The evaluation of final editing therefore distinguishes between reflecting the requested changes and preserving content outside the edit targets. Editing evaluation without per-turn target images is addressed through the human evaluation in Section C.3.

C.3 Human Evaluation

This evaluation targets the ASCII art generation and editing results of PEAT, and measures how well each instruction is reflected in the intermediate grayscale images and how closely the final image follows the visual characteristics of the last grayscale image. Evaluators reviewed the 4-turn editing sequences of 25 samples on the provided evaluation page and responded according to the sample and turn numbers. At each turn, they separately rated how well the current request was reflected and how well the previous state was maintained, and they rated the consistency of the final image once per sequence. All items use a scale from 1 to 5, where higher scores indicate that the corresponding criterion is more faithfully satisfied.

The evaluation instructions asked evaluators to judge according to the given criteria, without being influenced by their relationship with the researchers or by any expectations. The instructions explicitly stated that assigning scores higher than warranted in order to help the researchers could hinder objective interpretation of the results. The items below and Table 13 summarize the evaluation criteria used. The English wording in the table is a translation of the criteria from the Korean survey.

Table 13: Human evaluation rubric (English translation of the Korean questionnaire).

Score	Current-edit success	Previous-state preservation	Final-image consistency
1	The requested edit is not reflected at all.	Almost none of the content that should remain is preserved.	The final image barely matches the last grayscale image.
2	The edit is only partly reflected; important changes are missing.	Only some content is preserved, with major changes.	The images match only partly, with many important differences.
3	The edit is apparent, but clear errors remain.	Content is preserved to some extent, but clear changes remain.	The images match to some extent, but clear differences remain.
4	Most of the edit is reflected, with minor errors.	Most content is preserved, with minor changes.	The images mostly match, with minor differences.
5	The requested edit is faithfully reflected.	The content that should remain is faithfully preserved.	The final image faithfully follows the grayscale image's visual features.

Table 14: Total cost through final image generation: mean $\pm$ standard deviation over 25 four-turn sequences. Bold marks the lowest mean.

Method	Time (s)↓	GPU energy (Wh)↓
PEAT	272.1 $\pm$ 1.5	**6.61** $\pm$ 0.13
Qwen-Edit	683.3 $\pm$ 0.8	73.07 $\pm$ 0.34
FLUX	108.9 $\pm$ 0.3	11.91 $\pm$ 0.02
MTC	**93.7** $\pm$ 1.5	8.36 $\pm$ 0.03
OmniGen2	338.1 $\pm$ 1.1	37.21 $\pm$ 0.03

Current-edit success. Evaluators answered: "Looking at the grayscale image, how well is this editing instruction reflected?" This item measures the degree to which the instruction of the current turn is reflected.

Previous-state preservation. Evaluators answered: "Excluding the parts requested to change in this turn, how well is the content of the previous grayscale image maintained?" The reference for comparison is the immediately preceding grayscale image, and the targets of the current instruction are excluded from the preservation assessment. Accordingly, this item does not directly assess the preservation of the original RGB image's appearance or the fulfillment of all past instructions.

Final-image consistency. Evaluators answered: "How closely does the final image follow the structure, brightness, and shape of the last grayscale image?" This item measures the visual consistency between the last intermediate state and the final image.

Response aggregation. The response file contains ratings from 14 evaluators on 25 samples, with each evaluator responding to nine items per sample. Current-edit success and previous-state preservation each comprise $14 \times 25 \times 4 = 1{,}400$ responses, and final-image consistency comprises $14 \times 25 = 350$ responses. The fields `edit_success`, `edit_accumulation`, and `ascii_fidelity_final` in the response file correspond to the three evaluation items above, respectively. Per-turn averages are computed from the 350 responses at each turn, and final-image consistency, which is rated once per sequence, is aggregated separately from the per-turn scores.

C.4 Efficiency Measurement

Table 14 reports the mean and standard deviation of the total cost up to final image generation.

Table 15: ASCII art representation selection with 5,000 training images. C: character count; K: nonblank codebook size.

K	C	CLIP-I↑	CLIP-T↑	SSIM↑	PSNR↑	MSE↓
1000	8	0.878	**0.383**	0.517	10.528	0.096
	9	0.876	0.383	0.514	10.469	0.097
	10	0.875	0.382	0.513	10.468	0.097
2000	8	0.879	0.383	0.518	10.537	0.096
	9	0.877	0.383	0.516	10.510	0.096
	10	0.876	0.383	0.515	10.469	0.097
3000	8	**0.880**	0.383	**0.520**	**10.570**	**0.095**
	9	0.877	0.382	0.517	10.512	0.096
	10	0.878	0.383	0.518	10.538	0.096



Table 16: Spatial condition comparison with 10,000 training images. ASCII art uses $C = 8$ and $K = 3000$.

Condition	CLIP-I↑	CLIP-T↑	SSIM↑	PSNR↑	MSE↓
Prompt-only	0.820	0.381	0.455	9.094	0.128
BBox	0.836	0.379	0.451	8.817	0.136
Mask	0.866	0.384	0.497	9.608	0.116
Edge	0.877	0.382	0.523	9.953	0.107
ASCII Art	**0.887**	**0.384**	**0.530**	**10.759**	**0.091**

Both experiments use 500 evaluation images. All ASCII art settings use 64×64 grids with 2×2 patches. Bold marks the best unrounded score.

D Additional Representation Analyses

D.1 Evaluation of ASCII Art Representations

We select the character set and codebook size of the ASCII art representation, and evaluate the selected representation as a spatial condition. The selection criterion is reconstruction performance: an image is synthesized using a condition extracted from the original image and compared with that original.

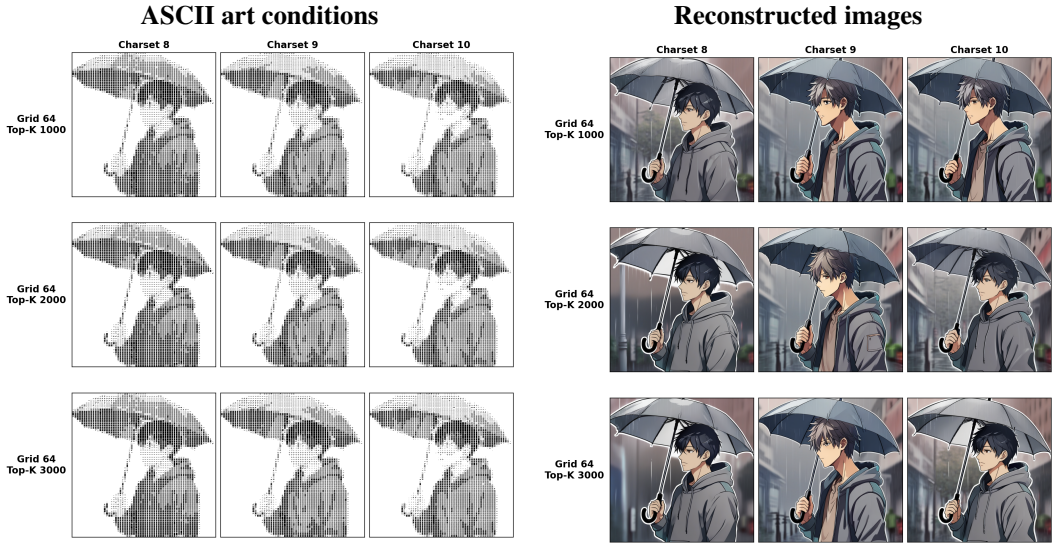
Evaluation setup. We fix a 64×64 grid and 2×2 patches, and compare the character set size C and the nonblank codebook size K. SDXL is frozen, and a T2I-Adapter is trained for 3 epochs for each spatial condition. Within each comparison experiment, the same training and evaluation images are used. Reconstruction is evaluated with CLIP-I, SSIM, PSNR, and MSE, and CLIP-T is used as an auxiliary metric that measures alignment with the scene prompt. The full evaluation setup is summarized in Appendix C.2.

Character set and codebook size. Table 15 compares $C \in \{8, 9, 10\}$ and $K \in \{1000, 2000, 3000\}$ with 2×2 patches fixed. The 8-character setting achieves the highest CLIP-I and SSIM for every K. In particular, $C = 8$, $K = 3000$ achieves a CLIP-I of 0.87976 and an SSIM of 0.51964, as well as the best PSNR and MSE among the compared settings. CLIP-T is highest at $C = 8$, $K = 1000$, but we select $C = 8$, $K = 3000$ because reconstruction of the original image is the selection criterion of this experiment. These results indicate that, within the compared range, enlarging the character set does not necessarily improve reconstruction.

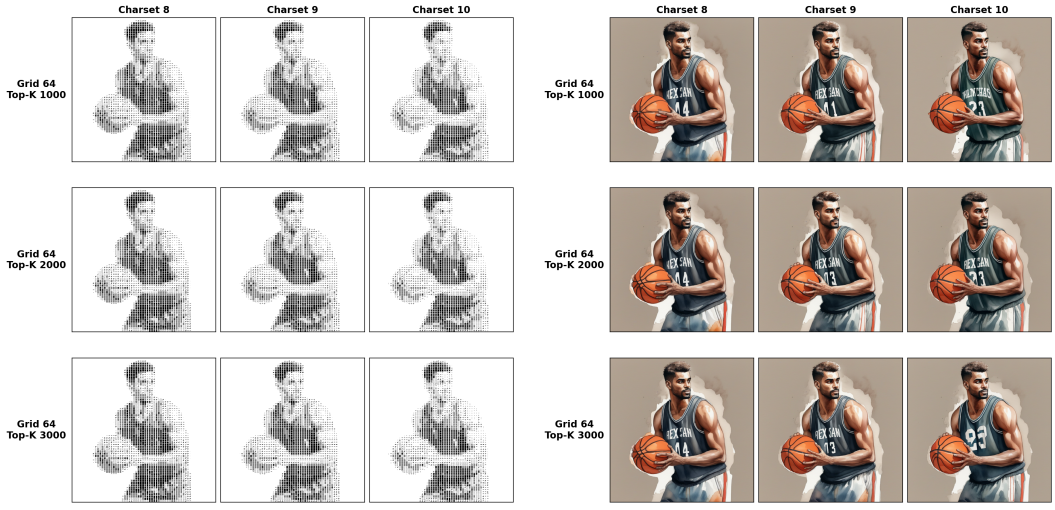
Figure 6 shows the ASCII art conditions and corresponding reconstructions for two examples across the nine settings. The examples illustrate visual differences across settings, while the representation is selected based on the aggregate reconstruction metrics in Table 15.

Effect of the ASCII art condition. We compare the selected ASCII art representation with prompt-only, BBox, Mask, and Edge conditions. The top of Table 16 shows the original image and an example of each spatial condition. Compared with prompt-only, the ASCII art condition with 2×2 patches raises CLIP-I from 0.8199 to 0.8870 and SSIM from 0.4553 to 0.5302. Compared with Edge, it also raises CLIP-I from 0.8773 to 0.8870 and SSIM from 0.5231 to 0.5302, and achieves the best PSNR and MSE. Under this evaluation setup, ASCII art yields better reconstruction scores than the other spatial conditions compared, supporting its utility as a spatial condition for image synthesis.

Representation cost and final setting. When the 64×64 grid is grouped into 2×2 patches and serialized in row-major order, every ASCII art setting is represented by $L = (64/2)^2 = 1024$



(a) Man holding an umbrella.



(b) Man holding a basketball.

Figure 6: Qualitative comparison of ASCII art representations for two examples. For each example, the left panel shows the ASCII art conditions, and the right panel shows the corresponding reconstructions produced by SDXL with a T2I-Adapter. Within each panel, rows correspond to nonblank codebook sizes $K = 1000, 2000, 3000$ from top to bottom, and columns correspond to character set sizes $C = 8, 9, 10$ from left to right. All settings use a 64×64 grid and 2×2 patches.

patch tokens. This count excludes the prompt and special tokens. The comparison of character sets and codebook sizes is therefore performed at the same number of patch tokens, and changing K affects the codebook vocabulary size and the substitution of out-of-codebook patterns rather than the sequence length. Under these conditions, we select the setting with the highest reconstruction performance and fix the 64×64 grid, 8 characters, 2×2 patches, a nonblank codebook size of $K = 3000$, and fixed-length row-major serialization for all subsequent multi-turn editing experiments.

E Qualitative Results and Failure Cases

E.1 Complete Editing Sequences



Figure 7: A four-turn editing sequence. Rows show successive editing turns from top to bottom. Columns show the edited ASCII art, the outputs of PEAT, and the outputs of five baseline editors.

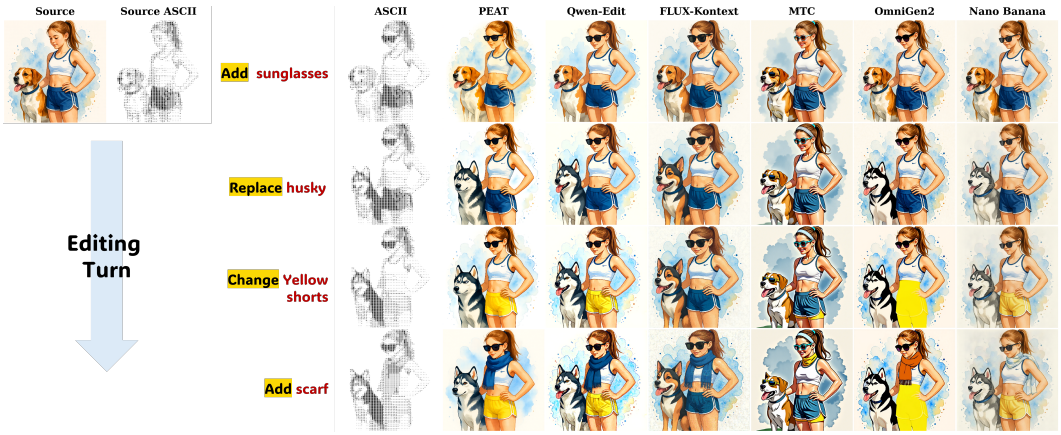


Figure 8: An additional four-turn editing sequence, following the same layout as Figure 7. Rows show successive editing turns from top to bottom. Columns show the edited ASCII art, the outputs of PEAT, and the outputs of five baseline editors.