Pattern Analogies: Learning to Perform Programmatic Image Edits by Analogy

Anonymous CVPR submission

Paper ID 1268

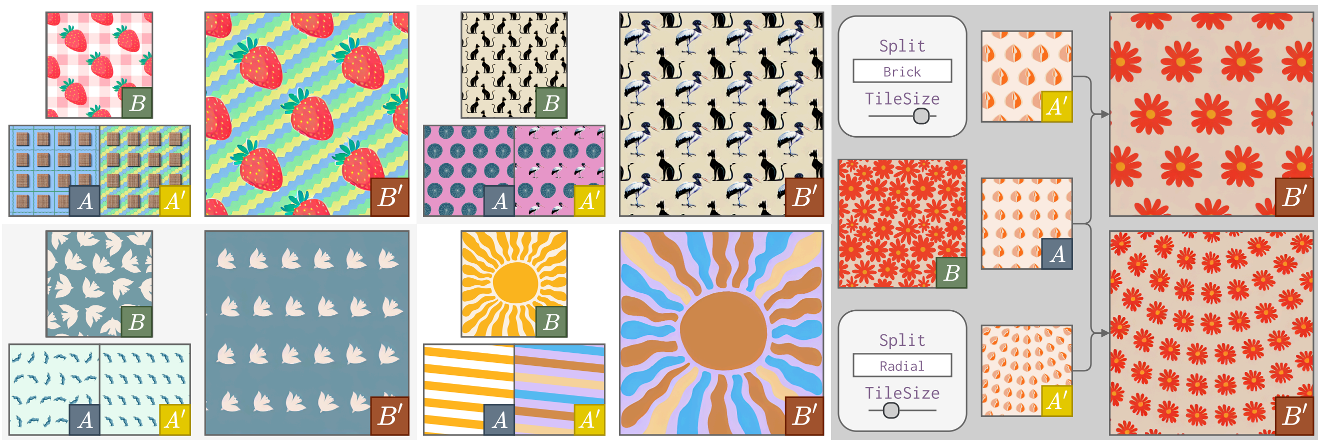

Figure 1. Our system performs *programmatic* edits on pattern images without inferring their underlying programs. (Left) Desired edits, expressed with a pair of patterns (A, A'), are executed on a target pattern B by a generative model to produce B'. (Right) Parametric changes $A \rightarrow A'$ enabled by our domain-specific pattern language induce corresponding changes to the more complex pattern B.

Abstract

Pattern images are everywhere in the digital and physical worlds, and tools to edit them are valuable. But editing pattern images is tricky: desired edits are often programmatic*: structure-aware edits that alter the underlying program which generates the pattern. One could attempt to infer this underlying program, but current methods for doing so struggle with complex images and produce unorganized programs that make editing tedious. In this work, we introduce a novel approach to perform programmatic edits on pattern images. By using a* pattern analogy*—a pair of simple patterns to demonstrate the intended edit—and a learning-based generative model to execute these edits, our method allows users to intuitively edit patterns. To enable this paradigm, we introduce* SplitWeave*, a domain-specific language that, combined with a framework for sampling synthetic pattern analogies, enables the creation of a large, high-quality synthetic training dataset. We also present* TriFuser*, a Latent Diffusion Model (LDM) designed to overcome critical issues that arise when naively deploying LDMs to this task. Extensive experiments on real-world, artist-sourced patterns reveals that our method faithfully performs the demonstrated edit while also generalizing to related pattern styles beyond its training distribution.*

1. Introduction

Visual pattern designs enhance digital media such as presentations, website themes, and user interfaces, and they are woven into the physical world through textiles, wallpapers, and product designs like hardware covers. Given the ubiquity of patterns, methods for editing them are essential: designers should be able to quickly experiment with variations, customize designs to meet specific needs, and adapt existing patterns to align with evolving trends.

Editing pattern images is not straightforward, as patterns are inherently structured, defined by rules that govern their layout and composition: tiling patterns adhere to principles of alignment and repetition (see Figure 1: top left), while retro-style designs rely on spatial divisions and fills (see Figure 1: bottom left). The edits that designers desire often aim to adjust these underlying organizational rules rather than make superficial, pixel-level changes. We refer to such edits as *programmatic* edits, requiring manipulation of the underlying program that defines a pattern's structure.

One strategy for enabling such programmatic edits is visual program inference (VPI) [5, 32, 46], where a program that replicates an image is automatically inferred, allowing users to modify the image by adjusting program parameters. However, applying VPI to patterns presents two obstacles. First, VPI attempts to infer a program that fully replicates a pattern, which can be challenging as patterns are often *semi-parametric*, blending rule-based logic with non-parametric components. For instance, the layout of elements in a tiling pattern may be rule-based, but the elements themselves may not be. Second, editing with an inferred program can be cumbersome, as they are often poorly-structured, with many unlabeled parameters, making them difficult to interpret. Consequently, VPI not only solves a more complex problem than necessary but also makes editing more challenging.

Can we perform programmatic edits without inferring the underlying program? Doing so requires the ability to *express* and *execute* the edit—both without direct access to the program's parameters. To express a programmatic edit, it's crucial to specify both *which* underlying parameter(s) to change and *how* to modify them. We draw inspiration from how humans communicate transformations: through analogies. By providing a pair of simple example patterns (A, A') that illustrate the desired change, users can intuitively convey both aspects of the edit. To execute these edits, we employ a learning-based conditional generative model. Given a pair of simple patterns (A, A') and a complex target pattern B, our system generates B', an edited version of B which performs the transformation demonstrated between A and A' while preserving B's other structural features. Crucially, A does not need to replicate or even be similar to B—it only needs to demonstrate *which* property to edit and how. Thus, specifying A is a much easier task than solving VPI. While prior works [1, 47, 51] have applied analogical editing to image manipulation, they focus primarily on appearance modifications. In contrast, our approach is the first to use analogies for *programmatic*, structure-aware edits. Figure 1 (left) shows examples of analogical editing on complex, real-world patterns.

To make our approach possible, we introduce SPLITWEAVE: a domain-specific language (DSL) for crafting visual patterns. SPLITWEAVE serves two purposes in our method. First, it enables parametric definition of input pairs (A, A'), allowing users to guide transformations in (B, B') as if the underlying program for B were accessible. In Figure 1 (right), modifying the SPLITWEAVE program for A' produces corresponding changes in B'. Second, SPLITWEAVE supports the creation of large-scale synthetic training data. We develop program samplers that generate high-quality patterns in two common styles: tiling-based designs with repeating elements and color field patterns characterized by splitting the canvas into intricate colored regions. Training a model for analogical editing requires a dataset of quartets (A, A', B, B'). By applying identical programmatic edits to the SPLITWEAVE programs for both A and B to produce A' and B', we ensure that the transformation from A to A' mirrors that from B to B'. This approach allows us to generate a diverse dataset of analogical quartets. Models trained on this dataset can generalize effectively to real-world patterns within these styles and can extend to related styles.

We use this synthetic dataset to train a novel diffusion-based conditional generative model for executing analogical edits. Our model directly generates edited patterns B' by conditioning on visual features extracted from input patterns (A, A', B). Existing image-conditioned diffusion models [43, 54] prove ineffective, as they fail to interpret the input analogies accurately and neglect fine details. To address these issues, we incorporate architectural enhancements that enable our model, TRIFUSER, to effectively perform analogical edits. With these improvements, TRIFUSER surpasses prior architectures for analogical editing when applied to pattern images.

To evaluate our method, we curated a test set of 50 patterns from Adobe Stock spanning 7 distinct styles. A perceptual study on this dataset shows that participants prefer edits by TRIFUSER over recent training-free and training-based methods. Although our training data covers only two of these styles, our model demonstrates effective generalization to the other, out-of-distribution styles. On a synthetic validation set with ground-truth analogical edits, our model produces outputs more similar to the ground truth than other methods. Finally, we showcase two applications of our approach: mixing attributes of different patterns and transferring pattern animations.

In summary, our contributions are as follows:

1. A novel framework for performing *programmatic* edits to pattern images without requiring program inference, leveraging analogies to specify and apply edits.
2. SPLITWEAVE, a DSL for crafting a diverse range of visual patterns, designed to support both parametric control and synthetic dataset generation.
3. A procedure for generating synthetic analogical quartets, enabling editing of *in-the-wild* patterns.
4. TRIFUSER, a diffusion-based conditional generative model that achieves high fidelity in analogical edits, surpassing prior techniques in both analogical fidelity and generation quality.

2. Related Work

We review three key areas: (1) Visual Program Inference (VPI) for programmatic editing of structured visual data and its limitations, (2) DSLs and synthetic data generation, specifically for visual patterns, and (3) analogical reasoning in computing, particularly for editing images.

Visual Program Inference for Editing: Visual Program Inference (VPI) enables programmatic edits of visual data by inferring executable programs from visual inputs. Prior works have achieved promising results in inferring material graphs [19, 27, 31, 46] and CAD programs for 2D [11, 28] and 3D [42, 53] inputs, using large annotated datasets [46, 52], differentiable program approximations [19, 41], or bootstrapped learning [9, 23, 25]. VPI is challenging to adapt to pattern editing due to the scarcity of high-quality annotated pattern data and the non-differentiability of most pattern programs. Also, VPI approaches often yield complex programs that are difficult to edit and interpret, making them impractical for editing. To address these challenges, recent work has aimed to simplify programmatic editing by inferring edit-specific controls [3, 10, 15] or a limited set of semantically meaningful parameters [22, 24, 26, 56]. Our approach shares this goal of enabling accessible control but extends it further: we transfer control from simple parametric objects to complex in-the-wild images via analogy, bypassing the need for VPI.

DSL and Synthetic data Domain-Specific Languages (DSLs) enable concise descriptions of structured objects, facilitating their creation. Prior works have developed DSLs for Zentangle patterns [45], material graphs [46], semi-parametric textures [14], and 3D models [21, 38]. Our DSL focuses on visual patterns constructed through partitioning and merging of canvas fragments. Closest to our work is ETD [30], which also uses canvas partitioning and merging operators, though it is limited to stationary patterns.

Analogical Reasoning Analogical reasoning is a foundational AI task: early work includes Evans' ANALOGY program [6], CopyCat [18], and Structure-Mapping Engine [7]. In visual computing, Image Analogies [16] pioneered the concept of analogy-driven editing. Recently, diffusion models have been adapted for analogical editing. DIA [51] introduced a training-free approach to analogical editing using pretrained diffusion models. Analogist [13] offers a complementary method, leveraging inpainting models alongside multimodal reasoning from large language models [36]. These training-free approaches are limited to images within the diffusion model's training domain, limiting their applicability to patterns. Other methods attempt to learn analogical editors by finetuning diffusion models on analogical pairs [1, 34, 47]. However, the focus of all these works remains largely on stylistic, appearance edits, often failing to perform *programmatic* edits. This limitation arises both from the models' architectures and from the lack of training pairs with *programmatic* edits. Our work addresses both these gaps, enabling structured, programmatic analogical edits for visual patterns.

3. Method

Our objective is to enable programmatic edits of 2D visual patterns without inferring their underlying programs. Instead, we propose an alternative that uses analogies to *express* desired edits and a conditional generative model to *execute* them. Formally, given two source patterns A and A' that demonstrate a desired edit, along with a target pattern B, our goal is to generate an edited target pattern B' that applies this edit to B. This task is defined as learning a mapping $f(A, A', B) \rightarrow B'$, where A, A', B, and B' are 2D RGB images ($\in \mathbb{R}^{H \times W \times 3}$). To learn this mapping, we generate a large synthetic dataset of analogical pattern quartets (A, A', B, B').

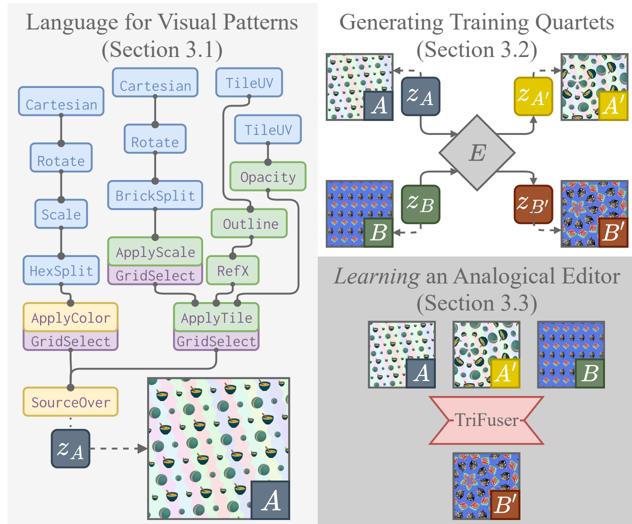

Figure 2. **Overview**: To create high-quality visual patterns, we introduce a custom DSL called SPLITWEAVE. Pairs of SPLITWEAVE programs (A, B) are then jointly edited to create analogical quartets. This synthetic data is then used to train TRIFUSER, a neural network for analogical pattern editing.

Figure 2 provides a schematic overview of our approach. First, in Section 3.1 we introduce SPLITWEAVE, a Domain-Specific Language (DSL) that enables the creation and manipulation of various kinds of patterns. Section 3.2 describes our approach for sampling analogical quartets in SPLITWEAVE to create the synthetic training data. Finally, in Section 3.3, we present TRIFUSER, a conditional generative model that learns to *execute* analogical edits.

3.1. A Language for Visual Patterns

To enable programmatic edits without program inference, our approach requires two core capabilities: (a) generating a large, high-quality synthetic dataset essential for training models to reliably *execute* analogical edits, and (b) the ability to create and parametrically control analogy inputs at test time to effectively *express* desired edits. Existing pattern generation tools are insufficient for these needs, as they are either limited to narrow pattern domains [45] or demand intense coding effort to produce diverse, high-quality patterns [30, 33]. To address these limitations, we introduce SPLITWEAVE, a DSL designed specifically to support analogical transformations in visual patterns. SPLITWEAVE combines abstractions for pattern synthesis with a node-based visual programming interface (see Supplementary), enabling efficient generation of high-quality synthetic pat-

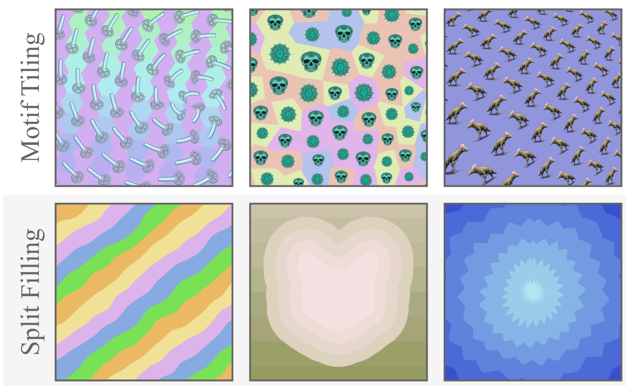

Figure 3. Custom program samplers for two pattern styles. Our samplers produce diverse and high-quality patterns, enabling generalization to real-world patterns.

terns for training while allowing flexible, precise pattern manipulation to define analogy inputs at test time.

SPLITWEAVE uses three types of operations for structured pattern creation: (1) *Canvas Fragmentation*, which allows structured divisions of the canvas, such as brick-like or voronoi splits; (2) *Fragment ID-Aware Operations*, enabling transformations that vary across fragments (e.g., scaling alternating rows or columns) to support spatial variability in non-stationary pattern designs; and (3) various *SVG Operators* for outlining, coloring, and compositing. Together, these operations enable efficient creation of patterns with complex structure and visual variety. Figure 2 (left) illustrates these capabilities in a SPLITWEAVE program for generating a tiling pattern design.

Our goal is to generate high-quality synthetic patterns using SPLITWEAVE that enable trained editing models to generalize well to real-world patterns. Naive sampling from the DSL grammar often leads to overly complex or incoherent patterns, limiting their effectiveness in model training. Instead, we draw inspiration from recent advances in fields such as geometric problem solving [50] and abstract reasoning [29], where tailored data generators have proven essential for tackling complex tasks. Following a similar approach, we design custom program samplers for two versatile and widely-used pattern styles. The first, *Motif Tiling Patterns (MTP)*, consists of compositions based on repeated *Tile* elements. These patterns exhibit controlled variations in tile properties across the canvas (e.g. orientation, color, and scale), creating visually cohesive yet richly diverse structures. The second, *Split-Filling Patterns (SFP)*, are generated by dividing the canvas into ordered fragments, applying region-specific coloring and transformations based on fragment IDs. Both pattern styles are common in digital design and support a wide range of programmatic variations, making them particularly suited for analogical editing tasks. Example patterns generated by our program samplers are shown in Figure 3; additional implementation details are in the supplementary materials.

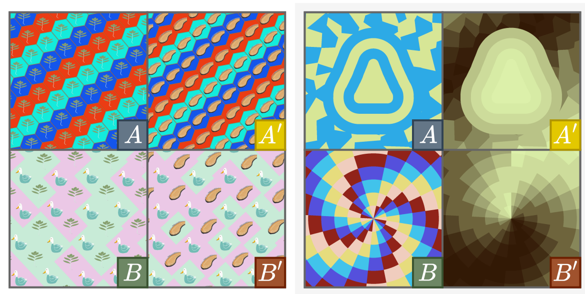

Figure 4. We create synthetic analogical quartets (A, A', B, B') with consistent edits between A and B pairs, providing data for training an analogical editing models.

3.2. Sampling Analogical Quartets

With the ability to generate diverse synthetic patterns using SPLITWEAVE (Section 3.1), our goal is now to construct analogical pattern quartets (A, A', B, B'). Each pattern image in a quartet is generated by a SPLITWEAVE program z. These quartets serve as structured training data for editing models, allowing them to learn consistent transformations that can generalize across different pattern domains.

Analogies in our framework are grounded in Structure Mapping Theory [12], which defines analogies as mappings of relational structure from a base to a target domain. We designate (A, A') as the base and (B, B') as the target, with the requirement that the relationship R between program pairs $(z_A, z_{A'})$ and $(z_B, z_{B'})$ remains consistent:

$$R(z_A, z_{A'}) = R(z_B, z_{B'}). \tag{1}$$

Rather than focusing on visual similarity between the patterns (A, A') themselves, this program-level analogy allows us to generate quartets with transformations that affect the underlying program, facilitating *programmatic* edits.

To construct these analogical quartets, we use a program sampler along with a predefined set of editing operators E. For each quartet, we begin by sampling an edit $e \in E$, followed by sampling initial programs z_A and z_B that are compatible with e. Applying e to both z_A and z_B yields transformed programs $z_{A'}$ and $z_{B'}$. By using identical transformations across domains, we ensure a consistent "edit relation" across the quartet, satisfying Equation 1 by construction. In Figure 4, we illustrate examples of synthetic analogical quartets generated using this method, demonstrating consistent transformations between (A, A') and (B, B').

Edit Operators E. We focus on edits targeting specific sub-parts of the program. Specifically, we consider three types of edits: *insertion*, *removal*, and *replacement* of sub-programs. For example, an edit operator might *replace* the sub-program responsible for splitting the canvas, while

CVPR #1268

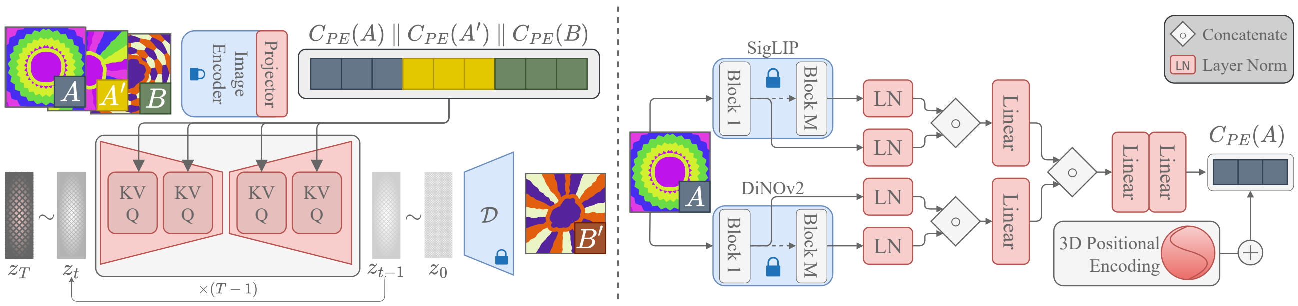

Figure 5. (Left) TRIFUSER is a latent diffusion model conditioned on patch-wise tokens of the input images (A, A', B) to generate the analogically edited pattern B'. (Right) To achieve high-quality edits, we enrich these tokens by fusing multi-level features from multiple encoders, followed by a 3D positional encoding: 2D to specify spatial locations and 1D to specify the token's source (A, A' or B).

other edits may *insert* or *remove* tiles within the pattern. Please refer to the supplementary for more details.

3.3. Learning an Analogical Editor

Our goal is to train a model on the synthetic data that is capable of performing analogical edits on real, *in-the-wild* patterns. Specifically, we aim to generate the target pattern B' from an input triplet (A, A', B). This approach allows users to demonstrate desired edits with a simple pattern pairs (A, A'), which the model then applies to a complex patterns B to produce B'. Given the success of Latent Diffusion Models (LDMs) in various generative modeling tasks [43], we chose to adapt an LDM for our task as well. We propose TRIFUSER, a latent diffusion model (LDM) for analogical editing (Figure 5). We provide a brief overview of LDMs to provide context before detailing TRIFUSER 's modifications for analogical editing.

Preliminaries: Denoising Diffusion Probabilistic Models (DDPMs) [17] transform random noise into structured data via reverse diffusion steps guided with a conditioning embedding $c(y)$ (often derived from text). Latent Diffusion Models (LDMs) extend DDPMs by mapping data to a lower-dimensional latent space via an encoder. During training, a UNet model [44] learns to remove noise introduced into the latents. During inference, a latent sampled from a normal distribution is iteratively denoised by the model to yield a clean latent. Finally, the clean latent is decoded to generate the output image. Please refer to [55] for a more thorough overview. For analogical editing we adapt an Image Variation (IM) model [54], which uses patch-wise image tokens extracted using a text-image encoder [39] as the conditioning embedding $c(y)$.

The simplest adaptation of an IM model to our task is to generate B' conditioned on image tokens from all three input images, concatenated as $C = c(A) \| c(A') \| c(B)$, where $\|$ denotes token-wise concatenation. This approach, however, suffers from three drawbacks: *Token Entanglement*, *Semantic Bias*, and *Detail Erosion*. We discuss each of these issues briefly, along with our solutions.

Detail Erosion: Despite using patch-wise tokens, the extracted features lack the fine-grained information needed to retain key aspects of B in the generated pattern B'. Consequently, the model often struggles to preserve elements like tile textures. To address this problem, we combine features from both the first and last layers of the feature encoder:

$$C_{hl}(P) = \mathit{Linear}(\mathit{LN}(c_{high}(P)) \cdot \mathit{LN}(c_{low}(P)), \quad (2)$$

where LN is layer normalization, $\cdot$ denotes channel-wise concatenation, P is an input pattern, and Linear is a linear projection layer that fuses low- and high-level features.

Semantic Bias: Image variation models typically use feature extractors such as CLIP [39], which are trained to align image embeddings with corresponding text embeddings. Such embeddings emphasize high-level semantics but lack spatial and fine-grained visual details. Combining these embeddings with features from text-free, self-supervised extractors, such as DiNO [2], has been shown to improve performance in downstream tasks [20, 49]. For our task, a similar approach—combining features from both text-image (m_1) and self-supervised (m_2) feature extractors—significantly enhances generation quality. The extracted features are fused as follows:

$$C_{mix}(P) = \mathit{Mixer}(C_{hl}^{m_1}(P) \cdot C_{hl}^{m_2}(P)), \quad (3)$$

where Mixer is a two-layer MLP that integrates features from the two extractors.

Token Entanglement: To successfully perform an analogical edit, for each patch-level feature token, the model must be able to identify to which source image (A, A', or B) that patch belongs as well as the 2D position of the patch within that image. Without these distinctions, the model often fails to identify the pattern to edit (i.e., B) and to recognize the desired edit from (A, A'). To address this problem, we introduce 3D positional encodings: two dimensions for spatial location within each pattern and one dimension for the source image. These encodings are applied to the extracted embeddings, yielding:

$$C^{\Omega} = C_{PE}(A) \parallel C_{PE}(A') \parallel C_{PE}(B), \quad (4)$$

$$C_{PE}(P)^{xy} = C_{mix}(P)^{xy} + PE(t_P, x, y), \quad (5)$$

where t_P is a one-hot vector encoding which input image a token comes from and $PE(t_P, x, y)$ positionally encodes both spatial and source information for each token.

As we demonstrate in Section 4.4, conditioning on C^{Ω} instead of C significantly enhances the quality of patterns generated by TriFuser. Our adapted architecture, shown in Figure 5, integrates the modifications described above to effectively address the described drawbacks. To enhance generalizability to real-world patterns, we initialize TriFuser with an existing pretrained IM model [54], and fine-tune only the denoising UNet and the projection layers in our feature extractor.

4. Experiment

In this section, we evaluate our approach along three directions: (1) the effectiveness of TriFuser at performing analogical edits on complex, real-world patterns, emphasizing how our synthetic data enables editing of in-the-wild pattern images; (2) the ability of TriFuser to support *programmatic*, structure-preserving edits without explicit program inference; and (3) the impact of architectural modifications introduced in TriFuser on the quality of generated patterns. We conduct a human perceptual study, quantitative assessments, and qualitative comparisons to demonstrate our system's ability to perform high-quality analogical edits across a range of pattern types.

4.1. Experiment Design

Datasets: We generate a large synthetic dataset of analogical quartets, i.e., pairs of analogical patterns (A, A', B, B'), using the SplitWeave program samplers introduced in Section 3.1. This synthetic dataset contains approximately 1 million samples covering two pattern styles, namely Split Filling Patterns (SFP) and Motif Tiling Patterns (MTP) (cf. Section 3.1). For MTP patterns, we synthesize 100k distinct tiles using the LayerDiffuse [58] model, guided by text prompts derived from WordNet [35] noun synsets. Additionally, we construct a synthetic test set with 1000 analogical quartets to evaluate model performance on unseen synthetic data. Further details on dataset construction are provided in the supplementary material.

To assess TriFuser on real-world patterns, we curate a test dataset of 50 patterns created by professional artists and sourced from Adobe Stock. This dataset spans seven distinct sub-domains of 2D patterns, representing a range of pattern styles. These styles include MTP and SFP patterns as well as previously unseen pattern styles such as Memphis-style, geometric, and digital textile patterns. Each pattern is annotated with a desired edit, and we use SplitWeave to generate a pair of simpler patterns (A, A') demonstrating this edit. This test set provides a challenging benchmark to evaluate TriFuser's generalization to diverse, real-world editing tasks.

Training details: We fine-tune a pre-trained diffusion model using our synthetic dataset of analogical quartets, as described in the previous section. We initialize our model with Versatile-Diffusion's Image Variation model [54]. We use SigLIP [57] as our text-image feature encoder and DiNOv2 [37] for self-supervised features. We fine-tune the model on 8 A100 GPUs using a batch size of 224 for ~ 65 epochs over 7 days. During inference, we generate each edited pattern B' with typical diffusion parameter settings such as a classifier-free guidance weight of 7.5 and 50 denoising steps.

4.2. Analogical Editing Baselines

To evaluate the analogical editing capability of TriFuser, we compare it to three baseline methods, each representing a leading approach for analogical image editing.

First, we consider *training-free editors* and *latent arithmetic editors*. Training-free editors repurpose pre-trained diffusion models to perform analogical edits without additional training [13, 51], leveraging the rich representations learned by diffusion models for editing. In this category, we compare against *Analogist* [13], the current state-of-the-art method. Latent arithmetic editors, on the other hand, rely on transformations in a learned latent space to infer analogical modifications [40, 48]. Note that these approaches only require samples from the target domain, not analogical training pairs. We implement a baseline for this method by fine-tuning a naive Image Variation model [54] on our synthetic dataset to learn a generative latent embedding space of patterns. At inference, analogical edits are generated using latent arithmetic: given patterns A, A', and B, we condition the generation of B' on $E(B) + E(A') - E(A)$. We refer to this baseline as *LatentMod*.

Finally, we consider *analogy-conditioned generative editors*, where models are explicitly trained on analogical data to learn analogical transformations [47]. This category includes our proposed TriFuser as well. Image Brush, the state-of-the-art method, fine-tunes a diffusion inpainting model for analogical editing with multi-modal conditioning. Since code for Image Brush is unavailable, we implement a similar baseline by fine-tuning a Stable Diffusion inpainting model. This model, which we term *Inpainter*, performs analogical editing by inpainting the lower-left quadrant of a 2x2 analogy grid containing (A, A', B) and conditioned on a fixed text template.

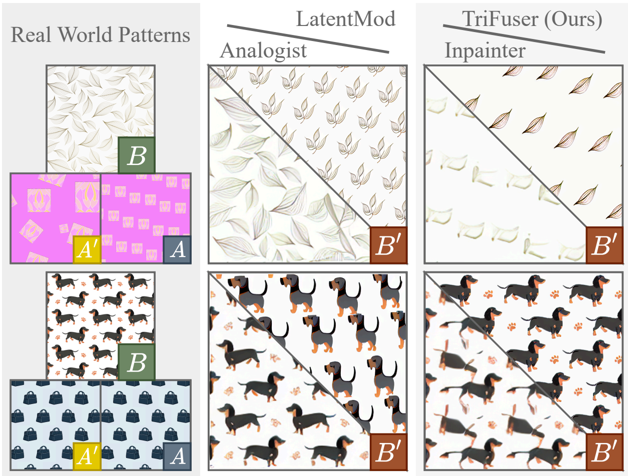

Figure 6. Qualitative comparison between patterns generated by our model, TRIFUSER, and the baselines. TRIFUSER generates higher quality patterns with greater fidelity to the input analogy.

	Preference Rate
TRIFUSER vs. *Analogist*	87.74%
TRIFUSER vs. *LatentMod*	80.78%
TRIFUSER vs. *Inpainter*	72.21%

Table 1. Results of a two-alternative forced-choice perceptual study comparing our model (TRIFUSER) against three baselines. Ours is preferred in the overwhelming majority of judgments.

4.3. Editing Real-World Patterns

To evaluate TRIFUSER's real-world analogical editing capabilities, we conducted a human preference study on the curated test set of Adobe Stock patterns.

We performed a two-alternative forced-choice perceptual study comparing TRIFUSER with baseline methods on all 50 entries in the test set. Each method generates $k = 9$ outputs for each input tuple, and we select the best one based on visual inspection. Participants were shown edited patterns generated by two different methods along with the input patterns (A, A', B) and instructed to select the edit that best preserved the analogical relationship and exhibited higher image quality. We recruited 32 participants for the study, resulting in a total of 1550 total judgments.

Table 1 presents the results, showing that TRIFUSER was preferred over both *Analogist* and *LatentMod*. Due to the domain gap between the training data of the underlying model [43] and pattern images, *Analogist* fails to interpret and edit pattern images. Meanwhile, *LatentMod* fails to perform reasonable edits as the embedding space lacks the low-level details necessary for *programmatic* edits While these baselines perform adequately on stylistic edits, they are unsuitable for *programmatic* editing. When compared to *Inpainter*, TRIFUSER was favored in 72.2% of comparisons.

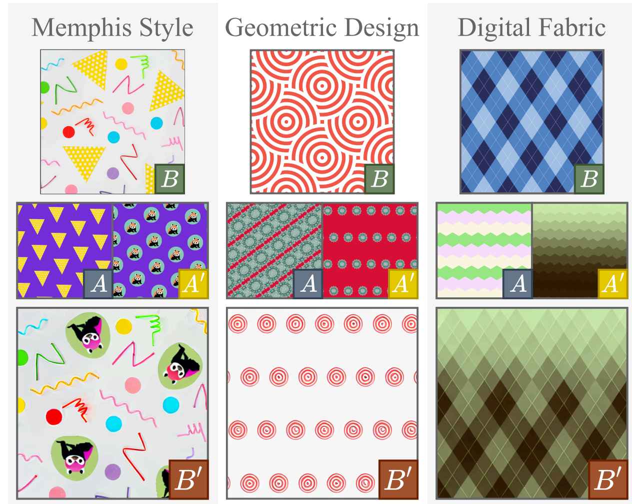

Figure 7. TRIFUSER effectively edits patterns from novel pattern styles not present in the training dataset. TRIFUSER shows a noteworthy ability to generalize beyond its training distribution.

Both methods benefit from training on analogical quartets, yet *Inpainter* sacrifices pattern quality as it generates the edited pattern in only a quarter of the full canvas resolution.

Figure 6 shows examples of pattern edits generated by TRIFUSER and the baselines, with our model consistently delivering superior results. In Figure 7, we show examples of TRIFUSER 's edits on out-of-distribution pattern styles not present in the training set. These results suggest that our synthetic training data enables manipulation of real-world patterns, even extending to certain untrained pattern styles.

4.4. Editing Synthetic Patterns

Next, we evaluate TRIFUSER's ability to perform *programmatic* edits on the synthetic validation set, which contains ground truth patterns B'. Ideally, this would involve verifying that the underlying program $z_{\hat{B}'}$ of the generated pattern reflects the same transformation from z_B as that between z_A and z'_A. However, this would require visual program inference on $\hat{B}'$, which is infeasible. Instead, we approximate this criterion by comparing the program outputs $\hat{B}'$ and B' to see if the visual results align with the intended transformation. To quantify this alignment, we use perceptual metrics—DSim [8], DIST [4] and LPIPS [59]—along with SSIM to capture pixel-level structural similarity.

Note that analogies can have multiple valid interpretations, and even a single interpretation may yield several visually-related variations. To account for this multiplicity, we generate $k = 5$ output patterns for each input set (A, A', B) and select the one that maximizes each metric. In other words, we evaluate whether at least one generated output aligns with the intended target.

Table 2 shows the results of this experiment. First, we note that TRIFUSER outperforms all baselines across

CVPR
#1268

	DSIM (↓)	DISTS (↓)	LPIPS (↓)	SSIM (↑)
Analogist	0.496	0.432	0.small	0.494
LatentMod	0.242	0.320	0.613	0.502
Inpainter	0.092	0.256	0.371	**0.713**
TRIFUSER	**0.074**	**0.184**	**0.304**	0.704

Table 2. Quantitative evaluation on the synthetic validation set shows that TRIFUSER generates patterns with higher perceptual similarity to the ground truth than the baselines.

	DSIM (↓)	DISTS (↓)	LPIPS (↓)	SSIM (↑)
TRIFUSER	**0.074**	**0.184**	**0.304**	**0.704**
- Pos. Enc.	0.147	0.239	0.383	0.659
- Lower	0.087	0.196	0.335	0.652
- Mix	0.098	0.210	0.345	0.682
Base [54]	0.585	0.460	0.815	0.435

Table 3. Subtractive ablation study on TRIFUSER shows that removing any component (see Section 3.3) degrades performance, and that removing all components (Base) results in a sharp decline.

all perceptual metrics. These metrics capture different aspects of perceptual similarity [8], and superior performance across all of them suggests a comprehensive improvement. Second, we observe that the analogy-conditioned generative editors (*Inpainter* & TRIFUSER) surpass both the training-free and latent modification editors. Interestingly, *Inpainter* achieves slightly higher SSIM scores than TRIFUSER, suggesting that future methods combining elements of both models might be fruitful.

4.5. TRIFUSER Ablation

To evaluate the contributions of each model component introduced in Section 3.3, we conduct a subtractive analysis on the synthetic validation set, using the same perceptual and structural metrics as above. For this ablation study, we remove each component one at a time and measure the resulting performance, as reported in Table 3. The results demonstrate that removing any single modification leads to a performance drop, with the removal of 3D positional encoding causing the most severe degradation. This is understandable: without 3D positional encoding, the network often fails to accurately identify which pattern to edit. For comparison, we also include results from the original Image Variation model [54] trained without any modifications (Base). As expected, this model performs poorly, underscoring the importance of our modifications in achieving high-quality analogical edits.

5. Application

The ability to edit patterns without requiring program inference unlocks new creative possibilities. We demonstrate two practical applications of analogical pattern editing:

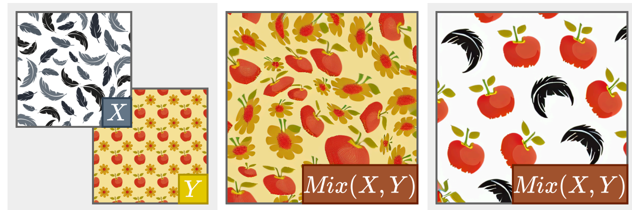

Figure 8. Our model helps users mix elements of different real-world patterns together, accelerating design exploration.

Pattern Mixing: Figure 8 shows example of using our method to *mix* elements of two real-world patterns X and Y, allowing the user to create unique, hybrid designs. The *Mix* operator is implemented by using a synthetic pair (A, A') to create a variant X' of X and then using the pair (X, X') to specify an edit to Y: *Mix*$(X, Y) = f(X, X', Y)$, where $X' = f(A, A', X)$. See the supplementary material for more details.

Animation Transfer: TRIFUSER can also be used to create animated sequences of edited patterns. By leveraging parametric SPLITWEAVE programs, users can generate animations for simple patterns and then apply these animations to complex patterns with no additional effort. See the video in the supplementary material for examples.

6. Conclusion

In this paper, we introduced a novel approach for *programmatic* editing of visual patterns without inferring the underlying program. By using analogies to *express* desired edits and a learned conditional generative model to *execute* them, our method provides an intuitive solution for pattern manipulation. A key component of our approach is SPLITWEAVE, a domain-specific language for generating diverse, structured pattern data. Paired with our procedure for sampling analogical quartets, SPLITWEAVE enables the creation of a large, high-quality dataset for training. We also presented TRIFUSER, a Latent Diffusion Model (LDM) designed to overcome critical issues that emerge when LDMs are naively deployed for analogical pattern editing, enabling high-fidelity edits that capture user intentions. Our experiments demonstrate that TRIFUSER successfully edits real-world patterns and surpasses baseline methods, while also generalizing to novel pattern styles beyond its training distribution. We believe that our DSL, dataset, and model will help drive further research on in-the-wild pattern image editing. Looking forward, we aim to extend this analogical editing framework to other domains such as semi-parametric 3D modeling while continuing to improve synthetic data quality and scalability.

References

[1] Amir Bar, Yossi Gandelsman, Trevor Darrell, Amir Globerson, and Alexei A. Efros. Visual prompting via image inpainting. *arXiv preprint arXiv:2209.00647*, 2022. 2, 3

[2] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 5

[3] Ta-Ying Cheng, Matheus Gadelha, Thibault Groueix, Matthew Fisher, Radomir Mech, Andrew Markham, and Niki Trigoni. Learning continuous 3d words for text-to-image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6753–6762, 2024. 3

[4] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P. Simoncelli. Image quality assessment: Unifying structure and texture similarity. *CoRR*, abs/2004.07728, 2020. 7

[5] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018. 1

[6] Thomas G. Evans. A heuristic program to solve geometric-analogy problems. In *Proceedings of the April 21-23, 1964, Spring Joint Computer Conference*, page 327–338, New York, NY, USA, 1964. Association for Computing Machinery. 3

[7] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The structure-mapping engine. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, page 272–277. AAAI Press, 1986. 3

[8] Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. *Advances in Neural Information Processing Systems*, 36, 2024. 7, 8

[9] Aditya Ganeshan, R. Kenny Jones, and Daniel Ritchie. Improving unsupervised visual program inference with code rewriting families. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 3

[10] Aditya Ganeshan, Ryan Y. Huang, Xianghao Xu, R. Kenny Jones, and Daniel Ritchie. Parsel: Parameterized shape editing with language, 2024. 3

[11] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. In *Advances in Neural Information Processing Systems*, pages 5885–5897. Curran Associates, Inc., 2021. 2

[12] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983. 4

[13] Zheng GU, Shiyuan Yang, Jing Liao, Jing Huo, and Yang Gao. Analogist: Out-of-the-box visual in-context learning with image diffusion model. *ACM Transactions on Graphics (TOG)*, 2024. 3, 6

[14] P. Guehl, R. Allègre, J.-M. Dischler, B. Benes, and E. Galin. Semi-procedural textures using point process texture basis functions. *Computer Graphics Forum*, 39(4):159–171, 2020. 3

[15] Julia Guerrero-Viu, Milos Hasan, Arthur Roullier, Midhun Harikumar, Yiwei Hu, Paul Guerrero, Diego Gutiérrez, Belen Masia, and Valentin Deschaintre. Texsliders: Diffusion-based texture editing in clip space. In *ACM SIGGRAPH 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery. 3

[16] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. *Image Analogies*. Association for Computing Machinery, New York, NY, USA, 1 edition, 2023. 3

[17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2020. Curran Associates Inc. 5

[18] Douglas Hofstadter and Melanie Mitchell. *The Copycat project: a model of mental fluidity and analogy-making*, page 205–267. Basic Books, Inc., USA, 1995. 3

[19] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Node graph optimization using differentiable proxies. In *ACM SIGGRAPH Conference Proceedings*, 2022. 2, 3

[20] Dongsheng Jiang, Yuchen Liu, Songlin Liu, XIAOPENG ZHANG, Jin Li, Hongkai Xiong, and Qi Tian. From CLIP to DINO: Visual encoders shout in multi-modal large language models, 2024. 5

[21] R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics (TOG), Siggraph Asia 2020*, 2020. 3

[22] R. Kenny Jones, David Charatan, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapemod: Macro operation discovery for 3d shape programs. *ACM Transactions on Graphics (TOG), Siggraph 2021*, 2021. 3

[23] R. Kenny Jones, Homer Walke, and Daniel Ritchie. Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[24] R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics (TOG), Siggraph 2023*, 42(4), 2023. 3

[25] R. Kenny Jones, Renhao Zhang, Aditya Ganeshan, and Daniel Ritchie. Learning to edit visual programs with self-supervision. In *Advances in Neural Information Processing Systems*, 2024. 3

[26] Milin Kodnongbua, Benjamin Jones, Maaz Bin Safeer Ahmad, Vladimir Kim, and Adriana Schulz. Reparamcad: Zero-shot cad re-parameterization for interactive manipulation. In *SIGGRAPH Asia 2023 Conference Papers*, New York, NY, USA, 2023. Association for Computing Machinery. 3

[27] Beichen Li, Liang Shi, and Wojciech Matusik. End-to-end procedural material capture with proxy-free mixed-integer optimization. *ACM Trans. Graph.*, 42(4), 2023. 2

[28] Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J. Mitra. Sketch2cad: Sequential cad modeling by sketching in

context. *ACM Trans. Graph. (Proceedings of SIGGRAPH Asia 2020)*, 39(6):164:1–164:14, 2020. 2

[29] Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M. Dunn, Hao Tang, Michelangelo Naim, Dat Nguyen, Wei-Long Zheng, Zenna Tavares, Yewen Pu, and Kevin Ellis. Combining induction and transduction for abstract reasoning, 2024. 4

[30] Hugo Loi, Thomas Hurtut, Romain Vergne, and Joelle Thollot. Programmable 2d arrangements for element texture design. *ACM Trans. Graph.*, 36(4), 2017. 3

[31] Arman Maesumi, Dylan Hu, Krishi Saripalli, Vladimir Kim, Matthew Fisher, Soeren Pirk, and Daniel Ritchie. One noise to rule them all: Learning a unified model of spatially-varying noise patterns. *ACM Trans. Graph.*, 43(4), 2024. 2

[32] Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *International Conference on Computer Vision*, 2019. 1

[33] L. McCarthy, C. Reas, and B. Fry. *Getting Started with P5.js: Making Interactive Graphics in JavaScript and Processing*. Maker Media, Incorporated, 2015. 3

[34] Zichong Meng, Changdi Yang, Jun Liu, Hao Tang, Pu Zhao, and Yanzhi Wang. Instructgie: Towards generalizable image editing. *arXiv preprint arXiv:2403.05018*, 2024. 3

[35] George A. Miller. WordNet: A lexical database for English. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994. 6

[36] OpenAI. Gpt-4 technical report, 2024. 3

[37] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *Trans. Mach. Learn. Res.*, 2024, 2024. 6

[38] Ofek Pearl, Itai Lang, Yuhua Hu, Raymond A. Yeh, and Rana Hanocka. Geocode: Interpretable shape programs. 2022. 3

[39] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 5

[40] Scott Reed, Yi Zhang, Yuting Zhang, and Honglak Lee. Deep visual analogy-making. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, page 1252–1260, Cambridge, MA, USA, 2015. MIT Press. 6

[41] Daxuan Ren, Jianmin Zheng, Jianfei Cai, Jiatong Li, Haiyong Jiang, Zhongang Cai, Junzhe Zhang, Liang Pan, Mingyuan Zhang, Haiyu Zhao, and Shuai Yi. Csg-stump: A learning friendly csg-like representation for interpretable shape parsing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 3

[42] Daxuan Ren, Jianmin Zheng, Jianfei Cai, Jiatong Li, and Junzhe Zhang. Extrudenet: Unsupervised inverse sketch-and-extrude fornbsp;shape parsing. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part II*, page 482–498, Berlin, Heidelberg, 2022. Springer-Verlag. 2

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 2022. 2, 5, 7

[44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. 5

[45] Christian Santoni and Fabio Pellacini. gtangle: a grammar for the procedural generation of tangle patterns. *ACM Trans. Graph.*, 35(6), 2016. 3

[46] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match: differentiable material graphs for procedural material capture. *ACM Trans. Graph.*, 39(6), 2020. 1, 2, 3

[47] Yasheng Sun, Yifan Yang, Houwen Peng, Yifei Shen, Yuqing Yang, Han Hu, Lili Qiu, and Hideki Koike. Imagebrush: learning visual in-context instructions for exemplar-based image manipulation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates Inc. 2, 3, 6

[48] Yoad Tewel, Yoav Shalev, Idan Schwartz, and Lior Wolf. Zero-shot image-to-text generation for visual-semantic arithmetic. *arXiv preprint arXiv:2111.14447*, 2021. 6

[49] Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. Eyes wide shut? exploring the visual shortcomings of multimodal llms. 5

[50] Trieu Trinh, Yuhuai Tony Wu, Quoc Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. 4

[51] Adéla Šubrtová, Michal Lukáč, Jan Čech, David Futschik, Eli Shechtman, and Daniel Sýkora. Diffusion image analogies. In *ACM SIGGRAPH 2023 Conference Proceedings*, New York, NY, USA, 2023. Association for Computing Machinery. 2, 3, 6

[52] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6772–6782, 2021. 3

[53] Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. Inferring cad modeling sequences using zone graphs. In *CVPR*, 2021. 2

[54] Xingqian Xu, Zhangyang Wang, Gong Zhang, Kai Wang, and Humphrey Shi. Versatile diffusion: Text, images and variations all in one diffusion model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7754–7765, 2023. 2, 5, 6, 8

[55] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4): 1–39, 2023. 5

[56] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K. Hodgins, and Levent Burak Kara. Semantic shape editing using deformation handles. *ACM Trans. Graph.*, 34(4), 2015. 3

[57] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11975–11986, 2023. 6

[58] Lvmin Zhang and Maneesh Agrawala. Transparent image layer diffusion using latent transparency. *ACM Trans. Graph.*, 43(4), 2024. 6

[59] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 7