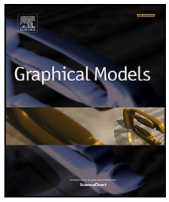
Graphical Models 132 (2024) 101213

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod



Pixel art character generation as an image-to-image translation problem using GANs☆

Flávio Coutinho [a,b,*], Luiz Chaimowicz [a]

[a] Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[b] Departamento de Computação, Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, Brazil

ARTICLE INFO

Keywords:
Generative adversarial networks
Pixel art
Image-to-image translation
Procedural content generation
Mixed-initiative system

ABSTRACT

Asset creation in game development usually requires multiple iterations until a final version is achieved. This iterative process becomes more significant when the content is pixel art, in which the artist carefully places each pixel. We hypothesize that the problem of generating character sprites in a target pose (e.g., facing right) given a source (e.g., facing front) can be framed as an image-to-image translation task. Then, we present an architecture of deep generative models that takes as input an image of a character in one domain (pose) and transfers it to another. We approach the problem using generative adversarial networks (GANs) and build on Pix2Pix's architecture while leveraging some specific characteristics of the pixel art style. We evaluated the trained models using four small datasets (less than 1k) and a more extensive and diverse one (12k). The models yielded promising results, and their generalization capacity varies according to the dataset size and variability. After training models to generate images among four domains (i.e., front, right, back, left), we present an early version of a mixed-initiative sprite editor that allows users to interact with them and iterate in creating character sprites.

1. Introduction

The game development process involves the creation of different types of content, which can be related to its game design, such as rules, narrative, and levels; or cosmetic, such as images, sound effects, and music. In the latter case, creating artistic assets usually requires multiple iterations until the final version is reached [1]. In the particular case of pixel art, designers carefully select the color of each pixel from a palette, and small changes in a sprite can incur a lot of rework in others [2]. To illustrate, when creating character sprites with different animations spanning multiple frames, modifying a character's part (e.g., the color or shape of his shoes) might require updating many other instances of the character in different poses. Such workflows are common and can take up valuable time from the often tight project schedules.

Such tasks might be laborious and repetitive, but designers could partially automate some work if they leveraged procedural content generation (PCG) in their creative processes. When the content is procedurally generated while embedded in game design tools, they can *"augment the creativity of individual human creators"* (p. 3) [3]. For example, some PCG systems let human creators and algorithms work together to produce game content. Indeed, automating only part of the content creation pipeline is usually designated as mixed-initiative PCG when both the algorithm and the human creator can interact with the content being generated in a co-creation process [4,5].

This paper is an extended version of our previous work [6], where we presented an architecture for generating pixel art character sprites in a target pose given its image in a source direction: for example, our approach can create a picture of a character facing right from another one of it facing front. We build on Pix2Pix's architecture [7], tweaking the networks' capacity, image representation, and the discriminator's patch size to the pixel art style and the addressed task.

We trained models with four small paired datasets ranging from 184 to 776 training examples, each featuring a different art style. In one dataset in which characters were modularly built by assembling different previously created body parts, the model generated images with high quality, with almost no perceptual divergence from the ground truth. In the other datasets, the results had different quality levels. Characters with shapes similar to others seen during training generated better sprites, sometimes suffering some level of high-frequency noise, while sprites with unique shapes yielded average to bad results, producing images of low utility. Such low numbers of examples hindered the models' generalization capacity. Hence, beyond the four small datasets,

☆ This work was partially supported by CAPES, CNPq and Fapemig.
* Corresponding author at: Departamento de Computação, Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, Brazil.
E-mail addresses: flavioro@dcc.ufmg.br, fegemo@cefetmg.br (F. Coutinho), chaimo@dcc.ufmg.br (L. Chaimowicz).

https://doi.org/10.1016/j.gmod.2024.101213
Received 8 April 2023; Received in revised form 7 January 2024; Accepted 21 January 2024
Available online 2 February 2024

in this extended version, we investigated training a model with a larger and more varied one (12,074 examples).

We had experimented using RGBA vs. RGB representation for the images and found that the information from the additional alpha channel improved the quality of the generated sprites. In this extended version, we also evaluated different data augmentation techniques (hue rotation and translation) [8,9], and a post-processing step to enforce the use of the same palette of the source image. The results were evaluated through visual inspection and using the L_1 distance and FID [10] metrics.

After the architecture definition and evaluation, we trained individual models to translate images among the front, right, back, and left poses, evaluating the generalization capacity of the models for each translation direction. Then, we built a sprite editor that allows users to interface with the models. Finally, through a worked example, we showed some possible outcomes that can arise from the interaction between the human and computational agents in a mixed-initiative setup.

2. Related work

This section presents other works that used deep generative models to create procedural images, using various architectures and tackling different tasks. Then, we show how some particularities of the pixel art style introduce challenges to image generation and how some works faced similar problems. Lastly, we present some concepts and related works of mixed-initiative content creation systems, comparing to how we approached it by building a prototype of a sprite editor that uses the generative models we proposed.

2.1. Image-to-image translation

Pang et al. [11] define image-to-image translation as the process of converting an input image x_A in a source domain A to a target domain B while keeping some intrinsic content from A and transferring it to the extrinsic style of B. In this work, we frame the approached problem as an image-to-image translation task, with the domains representing different character poses: front, right, back, and left. The intrinsic content of the source domain is the character phenotype features, accessories, and palette, while the extrinsic style of the target domain comprises the arrangement of pixels in the new pose.

An influential work on image-to-image translation is Pix2Pix [7]. It is a general architecture of deep generative models that proposes to condition the generation process on an image in the source domain while doing supervised training for the generator to create its version in a target domain (e.g., generating an aerial photo from a map view of the same place). Training is conducted as generative adversarial networks (GANs), using a generator and a discriminator networks. The generator has a U-net shape [12], receiving and producing images with the same resolution, but in a different domain. The discriminator follows a PatchGAN format [7], as instead of providing a single real/fake logit, it yields one classification per rectangular patch (of some size). The architecture has been used in different tasks, such as translating photographs from day to night, grayscale to colored images, and labeled to realistic facades. Next, we present related work involving image-to-image translation of 2D game content.

Serpa and Rodrigues [13] proposed an architecture based on Pix2Pix to translate line art sketches of characters into a grayscale and a colored sprite, encoding shading information and body-part segmentation of characters. The generated grayscale sprites were very close to the ground truth. Still, the colored ones contained a lot of high-frequency noise, especially for unseen poses, indicating that the model could not generalize well. The number of training examples was 85 and 530 for the two characters in their experiments.

Similarly, Jiang and Sweetser [14] also adapted Pix2Pix [7] but to translate grayscale to colored images. Experiments showed that representing images in YUV inputs yielded better-colored images than when represented with RGB. Such improvement is because when working in YUV, the network had to learn only the U and V channels, using Y as a redundancy of the condition.

Fig. 1. One pixel can change a character's expression. *Source:* [17].

Gonzalez et al. [15] proposed a Convolutional Variational Autoencoder (CVAE) to transfer the domain of a Pokémon from a source type (e.g., fire) to a target type (e.g., grass). However, the reconstruction of the original images yielded blurry results with shapes faintly resembling the original but with noisy colors. In addition, the domain transfer task (type swap) had considerable shape degradation, usually with high-frequency noise and dangling pixels outside the intended character shape. Due to the lack of examples in the dataset (974 before augmentation), Gonzalez et al. experimented with pretraining in another dataset and then fine-tuning the model, which yielded better results.

Hong et al. [16] proposed a multiple discriminators GAN (MDGAN) that translates images from a source pose into a target. The generator receives two images, one representing a shape and color sprite and another comprising a target pose (bone-graph sprite). The system is trained with two discriminators, one responsible for determining whether two images have the same color and shape, and the other deciding if the pose of a character sprite is correct compared to some target bone-graph. Albeit successful in the proposed experiments, the system requires a bone graph dataset matching characters' positions in the shape and color dataset. For that reason, Hong et al. artificially tailored datasets with non-pixel art images, which, unfortunately, may hinder its applicability with real pixel art character sprites. In addition, all characters must have the same shape, varying only its texture, and that dramatically restricts the artistic variability.

Our work also adapts Pix2Pix to generate pixel art character sprites. But differently from the aforementioned works, we want to translate characters drawn on a source to a target pose. MDGAN [16] is the most similar work regarding the particular task being tackled. Still, it imposes unnecessary restrictions on the problem. Regarding the image representation, we consider RGBA channels versus RGB used in [13,16], YUV in [14], and HSV in [15]. Our main contribution is the proposition of an architecture that can translate character sprite images from one side into another with differing levels of generalization capacity, depending on the dataset size and variance, exempting from an artificial bone-graph dataset and not restricting character shapes.

2.2. Generating pixel art

Even though generating pixel art may seem a more straightforward task than generating photorealistic images at first, it encompasses specific challenges. In comparison to photos, pixel art images frequently have a lower resolution, so each pixel encodes more information. For instance, a one-pixel difference in a character's face can change its expression (Fig. 1). The distribution of frequencies along the image is also different than in photographs: low-frequency regions (e.g., of the same or similar colors) interleave more quickly with high-frequency parts (e.g., edges) due to the smaller resolution but also because the borders are usually emphasized.

Pixel artists compose images by selecting colors from a small palette, frequently with less than thirty colors. Such constraint led to unique

coloring techniques such as color ramps (for discrete gradients) and dithering (mixing colors in a cross-stitch pattern). Lastly, pixel art datasets are much more scarce than photography-based ones. Next, we present different works that generate pixel art through deep generative models.

In [17], Coutinho and Chaimowicz analyzed two hypotheses to improve the quality of translating pixel art character sprites between domains: representing images as indices of colors in their palette and adding a histogram loss term to the generator of a GAN. The palette-based representation led to an overfitting scenario, with bad results for unseen data, and the histogram loss marginally improved the quality of the generation for unseen data.

Investigating a different hypothesis, Saravanan and Guzdial [18] proposed an adaptation of the Vector-Quantized Variational Autoencoder (VQ-VAE) to the task of learning a representation of Pokémon sprites. Rather than producing embeddings in a continuous space like a VAE, the Vector-Quantized architecture learns it in a discrete space [19]. According to Saranavan and Guzdial, the approach synergizes well with pixel art as individual encodings correspond to patches of pixels in the final image. They also suggested using 1×1 convolutions at the beginning and end of the encoder and decoder networks to improve the quality of feature map reduction across the depth channel.

In this work, we investigate the effectiveness of the proposed model with four small datasets, with varying examples (less than 1k) and art styles. We also use a larger dataset (12k samples) comprising different styles and much more variability. Later, we experiment with varying augmentation techniques to reduce model overfitting.

2.3. Mixed-initiative content generation

While some procedural content generation methods are fully automated, others involve more human interaction as they automate only part of the process. According to Liapis et al. [5], in such systems, a user and a computational agent can both take the initiative, whether to define what the available tasks are (*task initiative)*, to determine when to act (*speaker initiative*) or to decide how some problem should be solved (*outcome initiative*). Liapis et al. deem it optional for the participants to take part equally in the creation process, with the systems sometimes giving more control to either the user or the computational agent. In fact, they propose a spectrum of user and computational agent participation. On one end lie computer-aided generation systems that allow users to create content but do not take the initiative. On the other end lies "fire and forget" systems that require only some initial setup from the user, and the computational agent carries on with the bulk of the generation process.

Still on the definition of mixed-initiative systems, Lai et al. [20] suggest considering the continuum of participation as an open interval, excluding systems that lie at the extremes. Hence, systems require some initiative from both participants to be considered mixed-initiative. They proceed with a formal definition: *"[...] an iterative loop where at least one agent (human or artificial) can take the initiative by starting the content creation process and all involved agents [...], having memory of the current state or past states of the system, can contribute to the content, and respond at least once [...]"* (p. 3). Next, we present some works that propose mixed-initiative systems lying in different parts of the spectrum.

Zhang et al. [21] propose DrawCompileEvolve, a mixed-initiative drawing tool that gives human agents direct control over the initial specification of drawings and indirect over their interactive evolution conducted by a computational agent.

Karimi et al. [22] study how co-creativity can be fostered in a sketch design task's interaction between a human and a computational agent. The proposed system asks its users to sketch some object, and the computational agent offers a sketch that should inspire them. Then, users decide to consider the inspiration or not when drawing a new version of the graphic. Karimi et al. implemented the computational agent to use a conceptual shift when making its proposition: it picks an object conceptually different from what the user sketched (e.g., a plane as inspiration for a chair) and found that more distant objects yielded suggestions that improved the creativity of the human users.

Ibarrola et al. [23] also investigate co-creativity in a mixed-initiative drawing system but design the computational agent to help the users in their tasks. The tool allows human agents to receive completion suggestions using a text prompt.

In this work, after proposing an architecture for image-to-image translation of pixel art character sprites, we created a sprite editor that uses the trained models to allow a human agent to generate images procedurally. Like in [24], the resulting mixed-initiative system comprises an interaction loop with short iterations and may automate part of the creation process. Differently from [21], our system allows the human agent to interact very quickly and without any restriction with the content generated by the computational agent, both initially and in later iterations. In comparison to [22], although our system does not use conceptual shifts, we expect that by interacting with the images generated by the computational agent, the human agent might also receive some creative influence. Finally, we note that neither of the presented systems nor ours have proactive computational agents – that initiate actions by themselves.

3. Architecture overview

Similar to some related work [13,14], our architecture is based on Pix2Pix. Each model we train can translate images of a dataset from a source into a target pose. Next, we describe our generator and discriminator networks. As presented in Fig. 2, the input to the generator is a $64 \times 64 \times 4$ image with RGBA channels in the $[-1, 1]$ domain.

3.1. Generator

The generator is a U-net with $64 \times 64 \times 4$ input and output. The layers in the first half downscale the image to 1×1, and those in the second upscale the data to its original resolution.

Each downsampling step comprises a convolution (4×4 kernels), instance normalization (except for the first), and leaky ReLU layers, halving the resolution. Considering the input has 64×64 pixels, there are 6 downsampling blocks.

The decoder is a reflection analogy of the encoder, with upscaling steps instead. Each block comprises a transpose convolution (4×4 kernels), instance normalization, an optional dropout (50%), and ReLU activation layers. The first three blocks have dropout regularization, and the number of filters is the same as in the encoder but in reverse order. The last upsampling block has a $\tanh$ activation, so it outputs pixel intensities in the range of $[-1, 1]$. To allow the network to learn custom downsampling/upsampling mechanisms, it changes the resolution only through fractionally/strided convolutions [25]. There are skip connections from the output of the ith encoder layer to the $n^{th} - i$ decoder one, with $n = 6$, to preserve spatial information. The generator has 29,307,844 parameters and, after training, performs inference in 47 ms when run on a GeForce GTX 1050 GPU.

The loss function for the generator is similar to the one in [7] but uses a non-saturating adversarial part. We also use the L_1 distance between the real and generated images, with the hyperparameter λ set to 100. The generator's loss is:

$$\mathcal{L}_G = -\mathbb{E}_x[\log D(G(x))] + \lambda \mathbb{E}_{x,y}[\|y - G(x)\|_1] \quad (1)$$

where x represents images in the source pose, and y in the target; and $\|\cdot\|_1$ is the L_1 distance (absolute value of the differences of the pixels from generated to target images).

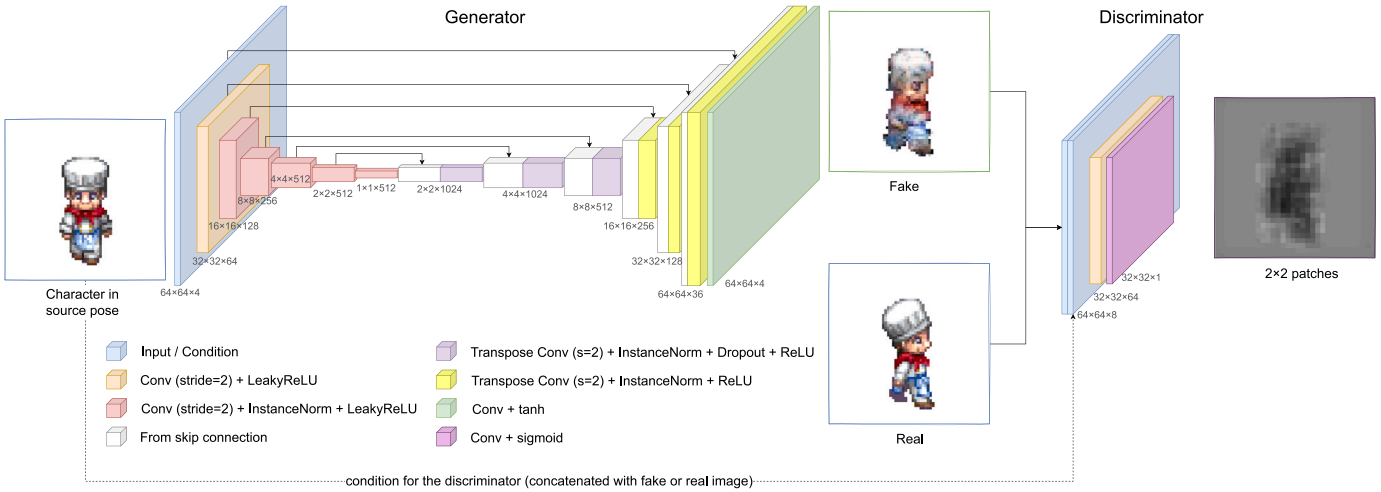


Fig. 2. Generator (left) and discriminator (right) network architecture.

3.2. Discriminator

Our discriminator is a PatchGAN [7]: a conditional classifier that takes a source image (e.g., character facing front) as a condition, and either a real or a generated sprite in the target direction (e.g., facing right), splits it into same-size square patches and discriminates each region as being real or fake.

The rationale behind splitting the output in patches is to enable local discrimination instead of providing a single global value of a sprite being either real or fake. Doing so allows penalizing more parts of the images that require more work from the generator. While the generator's L_1 loss term steers the generation towards the target images (but leads to blurry results), the patch discrimination works as a texture loss.

We experimented with different patch sizes: 2×2, 5×5, 11×11, and 64×64 (single patch), and Fig. 3 shows a comparison. The patch size of 2×2 achieved the closest result to the ground truth. The other images presented some color and shape noise, with the models using larger patches yielding worse results. In particular, the model with a single patch suffered from dangling pixels outside the sprite shape.

We attribute the better results with smaller patches to the discriminator being penalized by misclassifying 2×2 regions individually. As pixel art sprites typically have low resolution, each pixel carries a lot of information and should consider mostly its local vicinity. In such a setting, there are very small regions of low frequency (same color or just a slight variation). Hence, smaller patches evaluate not only the texture of the area but also the shape edges. Furthermore, it may be due to that double responsibility that the images present high color variation, even in parts that should have a single or a few colors. For such reasons, we chose to use 2×2 patches. Ultimately, the resulting shape and colors in all patch sizes somewhat resemble the ground truth.

Regarding its layers, the discriminator comprises the same downsampling blocks used by the generator (Fig. 2). Moreover, its loss function is the same as for conditional GANs [26], which can be calculated as a binary cross-entropy between the real images it discriminated as fake and the fake discriminated as real. The only change is that because the discriminator's output is not a single number per image but one for each patch, it is first reduced to the mean value of all patches. In total, the discriminator contains 9,217 parameters.

Fig. 3. Outputs of models with varying patch sizes for the discriminator.

4. Experiments

We conducted different experiments to evaluate the model architecture. First, we investigated how well the model performed on each dataset. Then, to improve the generalization for unseen data in this extended work, we trained a model with a larger dataset, experimented with different types of data augmentation, and evaluated a post-process step to ensure the sprites had no colors from outside its intended palette. Lastly, we trained models using the larger dataset and the selected hyper-parameters for all translation directions and evaluated them. All tests represent images using RGBA channels.

We analyzed the results qualitatively through visual inspection and quantitatively using the L_1 distance to the target images and the Fréchet Inception Distance (FID) [10]. While the L_1 distance measures the pixel-wise absolute difference of colors in two images, FID measures the distance between the feature vectors of the generated and the real images, using the Inception v3 network (proposed for image classification) [27]. Both metrics are zero for identical images, so lower numbers indicate more similar generated and target images.

Fig. 4. Sample images from the dataset showing four different sizes/art styles (columns) in four directions (rows).

Table 1
Description of the datasets used.

Source	Size	Examples	Per pose
Tiny Hero	64 × 64 × 4	912	1
RPG Maker 2000	32 × 24 × 3	216	3
RPG Maker XP	48 × 32 × 4	294	3
RPG Maker VX Ace	32 × 32 × 4	408	3

4.1. Datasets

We assembled datasets of pixel art character sprites from different sources. They consist of characters in four directions – front, right, back, and left – and comprise four different sizes and art styles (Fig. 4).

The dataset images had different character sizes, so the smaller ones were transparency-padded to the largest size, 64 × 64. Also, we created an alpha channel with the character shape for the images that lacked one. Table 1 presents the sizes and number of images per character pose. Regarding the latter, some sources (all except Tiny Hero) had three frames for each character side, as they were extracted from sprite sheets of walking animations. Each frame was treated as a different example such that the samples {1, 2, 3} depict the first character, {4, 5, 6} depict the second and so on. During train/test split (of 85%), the last character from the training set might have 0, 1 or 2 examples present in the test set. Even though a character might have frames spread between the train and test partitions, we do not consider it to be a harmful data leakage, as it happens with a single character per dataset.

The Tiny Hero sprites were modularly created by assembling previously-drawn body pieces to form each character. For this reason, it is usual for different characters to present the same or similar shapes but with different colors or a distinct combination of parts.

4.2. Training procedure

The networks' weights were optimized with Adam ($\beta_1 = 0.5$ and $\beta_2 = 0.999$) using a fixed learning rate of 0.0002. All models were trained in batches of 4 for 240 epochs on a GeForce GTX 1060 with 6 GB GPU, which took about 02:30 h.

Table 2
Training size, FID, and L_1 per dataset (train and test).

Dataset	Train size	Best FID		Best L_1	
		Train	Test	Train	Test
Tiny Hero	776	0.124	0.136	0.00761	0.00754
RM2K	184	0.522	2.908	0.00395	0.01168
RMXP	250	3.579	9.623	0.02642	0.06068
RMVX	347	0.631	4.990	0.01373	0.04704

5. Results

We present the generated images in different settings, analyze the results through visual inspection, and then quantitatively compare them using FID[1] and the L_1 distance. The values of the metrics were picked as the best ones from the test set throughout the training procedure (early stopping).

In all experiments, the models were trained to translate a front-facing character into the pose facing right. Fig. 5 shows the results of the models trained with the individual datasets. All examples were hand-picked from the test data and organized vertically with better results at the top.

5.1. Individual datasets

As some models started overfitting to the training data after some epochs, we selected the model weights that had the best scores in the test set. Table 2 presents each dataset's best (lowest) FID and L_1 distances. Nevertheless, at the end of the training, after the last epoch, the examples from most training sets (except for RMXP) could be reproduced with high fidelity by the models. Next, we analyze the models and show images only from the test sets.

5.1.1. Tiny hero

The images generated for the Tiny Hero sprites (Fig. 5) were perceptually identical to the ground truth, corroborated by the FID score of 0.136 and L_1 of 0.00754. Although it looks rather impressive initially, such a nice result could only be achieved due to how the Tiny Hero dataset was built: the characters were created by assembling body parts and accessories. So although the model had not seen the test images nor their ground truth, it learned how to translate each body part while training. This test does not show that the proposed model can generalize the translation but instead that it can memorize segments of the sprites it sees during training and use them to assemble characters in new combinations.

5.1.2. RPG maker 2000 (RM2K)

The first row of Fig. 5 shows a child girl translated with only minor color differences to the ground truth. Although that sprite was not on the training set, another one had the exact shape but different hair and dress colors. In this case, the model could understand their shapes and translate only the colors of parts of a sprite. Something similar happened to the granny and gramps in rows 3 and 4 – the model saw a sprite with the same shape but wearing an orange dress during training. In this case, it could not fully translate the colors, but it preserved the shape for the old lady and partially preserved it for the graybeard man.

The man in the second row of Fig. 5 was an exciting result suggesting a more profound generalization, as its translation uses information of a sprite with similar, but not equal shape, seen during training, but using different colors and slightly different shape. Fig. 6 shows the test image input, ground truth, and generated image, with the closest match from the training set to the side for comparison. We can note that the

[1] Note on FID: the distances should be calculated with thousands of images [10]. However, the metric still applies even with our tiny datasets.

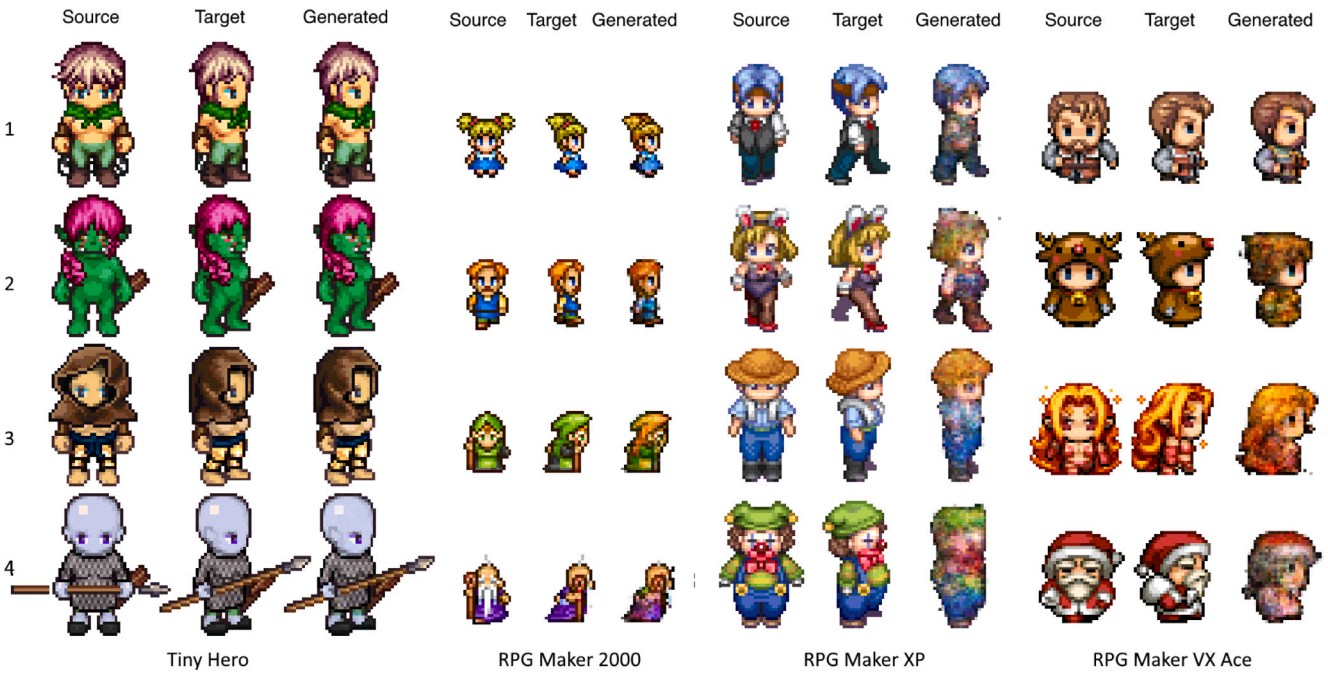


Fig. 5. Test examples from *Tiny Hero*, *RPG Maker 2000*, *RPG Maker XP*, and *RPG Maker VX Ace*.

Fig. 6. Front and right view of a test sprite (blue clothes) with the closest example (orange) in the training set from *RPG Maker 2000*. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

front-facing characters look similar, with differences in colors, the size and position of the hair, and the design of the mustache. The generated image contains some noise, and the edges inside the shape are not crisp, but it did not reuse the same hair and mustache shape as the other character.

The FID value for the generated images in the dataset was 2.908 and L_1 0.01168 (Table 2) and, after Tiny Hero, those were the lowest values, indicating that the generated images were close to the ground truth. Having a test set with sprites that are mostly color variations from the training set contributed to the good results with this dataset.

5.1.3. RPG maker XP (RMXP)

This model had the highest FID (9.623) and L_1 distance (0.06068), indicating lower translation quality. Fig. 5 shows examples of the best quality translations (first rows) to the worst ones (last rows). Such sprites are larger than those from RPG Maker 2000 (32×24 vs. 48×32) and not modular as in Tiny Hero. Furthermore, there are no sprites with the same shape with just some color variation, making it a reasonable lower bound for the translation quality of the proposed model.

The image generated in the first row of Fig. 5 had higher quality than the others. That happened because this dataset has three images per character pose, which are frames of a walking cycle. In this case, the frame of the character standing still stayed in the training set, with the other two as part of the test. Although the generated image contains some noise, it is possible to see the faint color of the ribbon in his head, and the overall shape is correct. This suggests that small changes in the pose are at least partially generalizable.

The characters in rows 2 and 3 (dealer girl and farmboy) have distinct features – a bunny tiara and a straw hat. Unfortunately, the model could not correctly translate such features' shapes, but we can notice that some traits, such as the most prominent colors and prominences, exist.

Lastly, the generated image for the clown in row 4 has a lot of high-frequency noise. Again, the colors appear in meaningful positions, but there is little information to separate them visually.

5.1.4. RPG maker VX Ace (RMVX)

The first row of Fig. 5 presents the best result, and the character is a color variation of another sprite seen during training. Except for the color of the left hand and highlights in his hair, there is very little perceptual difference with the ground truth.

The characters in rows 2 and 3 have distinctive features, such as in RPG Maker XP. They differ from the ground truth, but the sprites have overall colors and approximate shape. In addition, it had high-frequency color noise and blurry edges. In the fourth row, Santa Claus was generated implausibly and, even worse, without its sack of gifts.

The comparison of FID and L_1 distance between the model output and the ground truth images was 4.990 and 0.04704 — between RPG Maker 2000 and RPG Maker XP.

Except for Tiny Hero, the results for the individual datasets indicate overfitting, as the metrics of FID and L_1 are lower for the training set (Table 2), and they keep decreasing for the training set only when run for more epochs. Next, to improve the models' generalization, we experiment training using a larger dataset.

5.2. Larger dataset

In addition to the small individual datasets, we tested training a model with all of them together plus a collection of 10,517 assorted pixel art sprites. These comprise primarily humanoid characters in different sizes and art styles but also a few sprites of animals, vehicles, and monsters. In total, this eclectic dataset contains 12,074 paired training examples. Fig. 11 shows some examples in the first column of each quadrant.

Table 3
Training on the larger dataset but validating on the individual ones (values for the test set).

Validating with test set from	Best FID		Best L_1	
	Indiv.	Larger	Indiv.	Larger
Tiny Hero	**0.136**	0.553	**0.00754**	0.01724
RM2K	**2.908**	4.713	**0.01168**	0.01468
RMXP	9.623	**8.582**	0.06068	**0.05971**
RMVX	4.990	**4.804**	0.04704	**0.04490**
Larger	–	3.319	–	0.05375

We first evaluated this model with a test set with the same distribution as its training set, and it had a FID value of 3.319 and L_1 of 0.05375. We also calculated both validation metrics but used the four individual datasets' test sets to assess if the models trained only with the individual datasets would improve if trained on the larger one. Table 3 shows the FID and L_1 values of the model trained with the larger dataset but validated on the individual ones in the first four rows. The last one presents the results with its own test set.

Training the model on the larger dataset yields FID and L_1 values that are worse for Tiny Hero and RPG Maker 2000 but better for RPG Maker XP and RPG Maker VX Ace, which represent more complex tasks. In the case of Tiny Hero, having contact with many diverse art styles from the large set hindered the model's ability to memorize individual character parts. Still, both FID and L_1 are the lowest values among the other datasets. Regarding RPG Maker 2000, using the larger dataset to train also degraded the results. That can be probably due to its test data being mostly comprised of color variations of the training set, and seeing much variance during training made focusing on color transfer more difficult.

Regarding RPG Maker XP and VX Ace, training with the larger dataset yielded better results. As their test set has a lot more variability than the other datasets, when trained with increased shape variation, the model might have been able to understand better how it can "turn" characters from one pose to another.

Fig. 7 shows some images generated by the model trained with the larger dataset (4th column) and by the corresponding original dataset (3rd column). We can observe that some results improved with the additional training data (such as in rows 2, 5, 6), some might have as well (1, 7), but others just changed without necessarily enhancing (3, 4, 8).

To summarize, having a larger dataset with more variability hinders the output quality for tasks where color transfer or memorization of parts is necessary. But introducing more variation in the training data can enhance the results especially for the more challenging tasks. That leads us to hypothesize that having an even larger dataset could improve the quality even more.

Fig. 7. Comparison of images generated when training using the original individual dataset (3rd column) and the larger one (4th column). Examples come originally from RPG Maker 2000 (row 1), RPG Maker XP (rows 2-4, 8), and RPG Maker VX Ace (rows 5-7).

5.3. Augmentation

Especially useful for small datasets, data augmentation can help reduce the model overfitting to the training data, yielding better results for unseen data [8,9]. However, due to some characteristics of pixel art and from the task approached in this work, some augmentation options cannot be used. For instance, rotation by oblique angles and scaling transformations produce degraded images that do not conform to the art style. Moreover, flipping and rotation by non-oblique angles could change the domain of the images (e.g., character facing right, when horizontally flipped, starts facing left), or it could put characters in a non-existent pose (e.g., upside down).

We implemented two transformations: hue rotation and translation of the character inside the 64 × 64 region, and studied their impact when applied together and separately. There is an 80% probability for an input to undergo augmentation. The sprite hue is rotated by a random angle between −180 and 180 degrees, while the translation moves the character inside the sprite area by some random offset.

We trained models on each dataset and compared the effects of both transformations together, only hue rotation, only translation, and no augmentation. Table 4 presents the values of FID and L_1 for each experiment.

Quantitatively, none of the experimented scenarios (no technique, only hue rotation, only translation, both) scored the best values for all validation sets, indicating that its use must be carefully considered. However, we observed some repeating behaviors:

- Hue rotation improved the quality of the generated sprites that had their shape seen during training but in a different color palette (color transferring). RPG Maker 2000 had various color variation sprites in its test split — hence, its more significant gain with this augmentation technique. Fig. 8 shows some examples.
- Translation acted as a strong regularizer for the model, hampering its ability to memorize (and hence, overfit to) the training data. This led the generated images to have less noise at the expense of some detail, even on the training set. Fig. 9 shows examples.

Table 4
Metrics for the test set per augmentation technique.

Dataset	Augmentation	Best FID	Best L_1
Tiny Hero	None	**0.135**	**0.00754**
	Hue rotation	0.174	0.00973
	Translation	0.917	0.02056
	Both	0.895	0.02086
RPG Maker 2000	None	2.908	0.01168
	Hue rotation	**2.056**	**0.00888**
	Translation	7.874	0.01826
	Both	7.325	0.01630
RPG Maker XP	None	9.623	0.06068
	Hue rotation	9.687	0.05934
	Translation	9.814	0.06002
	Both	**9.400**	**0.05887**
RPG Maker VX Ace	None	**4.990**	0.04704
	Hue rotation	5.237	**0.04497**
	Translation	6.537	0.04640
	Both	5.808	0.04617
Larger	None	**3.319**	0.05375
	Hue rotation	3.623	**0.05343**
	Translation	3.862	0.05593
	Both	3.948	0.05624

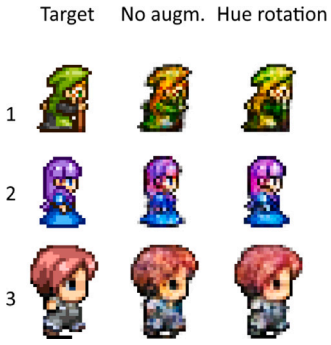


Fig. 8. Improvement from hue rotation on color variation sprites from *RPG Maker 2000* (rows 1 and 2) and *RPG Maker VX Ace* (row 3).

- Using both augmentation techniques together generated better images than with translation alone but worse than with hue rotation only (as in Tiny Hero, RPG Maker 2000, and RPG Maker VX Ace).
- In Tiny Hero, all augmentation techniques produced worse results, probably due to the dataset's modular nature. We conclude that any regularization added to the model (in the form of augmentation) hindered the generator from being able to memorize individual character parts.

5.4. Post-processing of palette-quantization

To avoid having small variations of the same color in a sprite, we created an experiment that adds a post-processing step that enforces the generated images to only have colors from the source images' palettes. As it typically reduces the number of colors in the final image, we called it palette-quantization. It works by replacing each pixel of the generated image by its closest color (in some color space) in the palette of the source image.

We executed different experiments to determine which color space would yield better results when finding the closest colors, comparing: RGB, YUV, and CIELAB. YUV is a color model that represents luminance information separately from the chroma of the color, with the Y term encoding a grayscale value, and U and V being a blue–green and yellow-red projections [14]. CIELAB, in turn, is a color space intended

Fig. 9. Difference from the translation on the noise vs. detail from *RPG Maker XP* (rows 1 and 2) and *Larger dataset* (row 3).

Table 5
Results of FID and L_1 for images after post-processing.

Post-process	With Augm.		Without Augm.	
	B. FID	Best L_1	B. FID	Best L_1
None	9.400	**0.05887**	9.623	**0.06068**
Using RGB	8.778	0.05922	9.713	0.06324
Using YUV	8.892	0.05921	9.526	0.06322
Using CIELAB	**8.176**	0.05973	**8.306**	0.06403

to represent colors in a perceptually uniform space, with the L component encoding the lightness, and A and B encoding green-magenta and blue-yellow spectra [28].

For the experiment, we used models trained with the RPG Maker XP dataset, as it represents a more difficult task than the others due to its consistently higher FID and L_1 values observed in previous experiments. To evaluate the palette-quantization in different scenarios, we experimented training without and with data augmentation (hue rotation and translation). Table 5 summarizes the results, and Fig. 10 shows some images generated by the models before and after the palette-quantization step.

Considering FID, using CIELAB yielded the best results: 8.176 (with augmentation) and 8.306 (without augmentation). We hypothesize that as it is a more perceptual color space, the pixels are matched to the palette in a more meaningful way. An example of this improved mapping can be seen on the cook character in row 2 of Fig. 10: his red scarf appears more continuous if the post-processing uses the CIELAB color space than if using RGB or YUV.

Qualitatively, the three representations yield similar results: they mostly do not change the character shape (except for a few pixels), and they reduce the model uncertainty by spreading the same color in larger regions (e.g., note the sailor's bandana in row 1, and the dancer's skirt in row 3, especially using CIELAB). Such benefits are more expressive with the model trained with data augmentation (rows 1–2). Another benefit is that the sprites after the palette-quantization contain only colors from the intended palette, making them more useful to pixel art designers.

6. Evaluation with all sides

Having selected the configuration and hyper-parameters that yielded the best model for the front to right pose task, we trained one model for each pair of source and target directions (16 in total).

Fig. 10. Resulting images after the palette-quantization post-processing step for test images from the RPG Maker XP dataset. The columns show the input, target, and generated image from left to right, followed by the post-processed versions using RGB representation, YUV, and CIELAB. Rows 1–2 are from the model trained with augmentation, and 3–4 are from the model without augmentation.

Table 6
Results of FID and L_1 for the evaluation of models trained with all sides as source and target.

Target →	Back		Left		Front		Right	
↓ Source	Best FID	Best L1	Best FID	Best L1	Best FID	Best L1	Best FID	Best L1
Back	0.014	0.00096	3.637	0.05905	**5.018**	**0.04878**	3.828	0.05890
Left	7.684	0.05443	0.004	0.00090	7.205	0.05821	**1.722**	**0.02318**
Front	**5.118**	**0.04076**	3.685	0.05225	0.009	0.00111	3.623	0.05343
Right	7.702	0.05405	**1.860**	**0.02357**	6.534	0.05743	0.008	0.00098

The training procedure used the larger dataset as it spans different art styles and the learned models were used in the sprite editor described in the next section. They were trained using hue rotation as augmentation. We used $\lambda_{L_1} = 100$ and optimized the weights using Adam ($\beta_1 = 0.5$ and $\beta_2 = 0.999$) with learning rate of 0.0002, and the training loop used batches of size 4 executed for 240 epochs. Each model took about 06:45 h on a GeForce 1080 Ti with 11 GB GPU.

Table 6 shows the values of FID and L_1 for all models at the iteration in which they were the lowest (best). Both metrics converged in most measurements, and the numbers in bold indicate the source domain (on rows) that yielded the best results for the target pose (on columns).

The models that had the same pose as source and target (e.g., back to back) optimized to become an identity function, which is indicated by the very low values of FID and L_1 in the main diagonal of Table 6 (indicated in gray). As expected, those models learned a set of weights that reproduces the input image.

Analyzing the table by columns indicates which source side yielded the best model for the target pose in that column. For back, front-facing characters yielded the best input (FID 5.118, L_1 0.04076). Left and right yielded the best models to each other (left to right had FID 1.860 and L_1 0.02357, right to left had FID 1.722 and L_1 0.02318). And back-facing characters were the best input for the front target pose (FID 5.018 and L_1 0.04878).

Regarding the difficulty of the tasks, we can note that for back and front as targets, the metrics had the highest values, indicating them as the hardest, followed by left and right. The easiest tasks were the translation of left to right and vice-versa, which usually consist of a reflection over the Y axis.

Fig. 11 shows some example characters generated by the models, hand-picked to illustrate good and bad results. Each quadrant depicts a character from the larger dataset. The first column shows the source image in the poses of back, left, front, and right (from top to bottom). The remaining four columns contain the generated image for some particular target pose.

The models that had the same pose as source and target (e.g., back to back) learned an identity function that reproduces the input image (see the main diagonals Fig. 11), as also indicated by the much lower FID and L_1 values for such models.

We note that some translation directions are more ambiguous than others. For instance, the back pose of a character might not encode enough information to generate all details of the front, or even left or right. Such is the case of the zombie (quadrant 2) and the semi-bald (quadrant 4) characters. Such translations yielded almost plausible results, but that were unexpected if we consider the target image (as they lack zombie traits and the frontal lock of hair). Other directions are less ambiguous, such as left to right and vice-versa, as most sprites

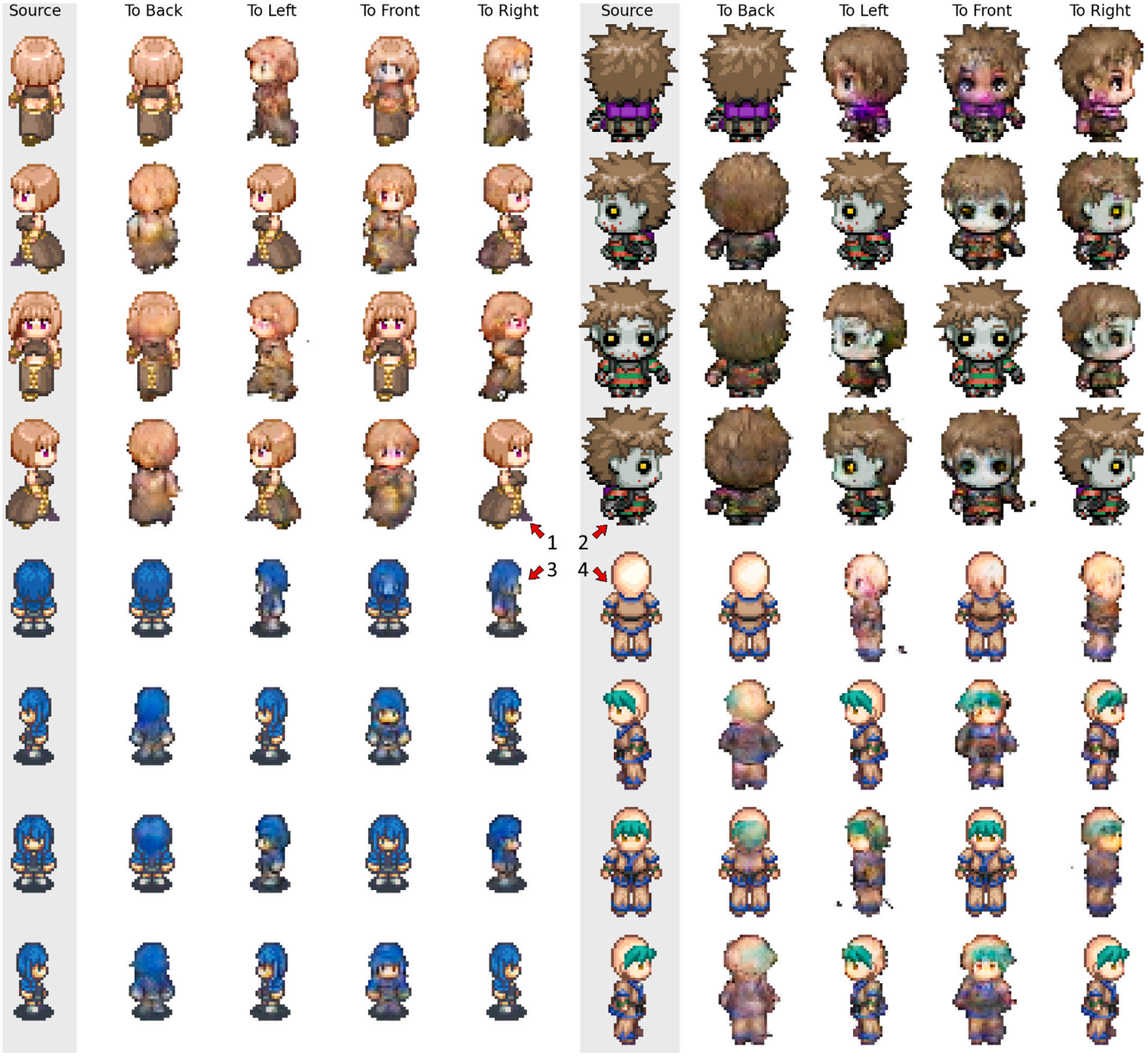


Fig. 11. Four examples generated from the 16 models trained with the larger dataset in each quadrant. The first column shows the source image, with each row representing a different pose from: back, left, front, and right. The second to fifth columns show the image generated by the model with the source side defined by the row and the target side by the column.

are laterally symmetric. In those cases, the generated images are close to the target.

Another observation is the varying degree of uncertainty shown by some models, especially in more ambiguous tasks. All images generated from the back-facing blue haired guy (quadrant 3) have an incorrectly colored bluish face. A similar effect happened to the semi-bald character (quadrant 4), within the shape of his hair and his eyes. The generation of front-facing sprites for many examples also presented some misalignment of the eyes, such as the dancer lady (quadrant 1) generated from back and left poses.

We hypothesize that the impacts of the ambiguity and the uncertainty can be alleviated by models with architectures that account for multiple inputs, at least to some extent. For instance, if more than one pose of a character is available, such a model could potentially use the additional information to better generate the image in the target pose.

7. Sprite editor

Although the proposed image-to-image translation models still require improvements regarding their generalization, we believe they could already support artists in an interactive tool. Hence, we created a prototype of a mixed-initiative system that could potentially aid users in creating pixel art character sprites. We first describe the system features and how users can interact with it. Then, we depict how it can currently be used in a creative workflow through a worked example. And we conclude the section with possible lines of further investigation.

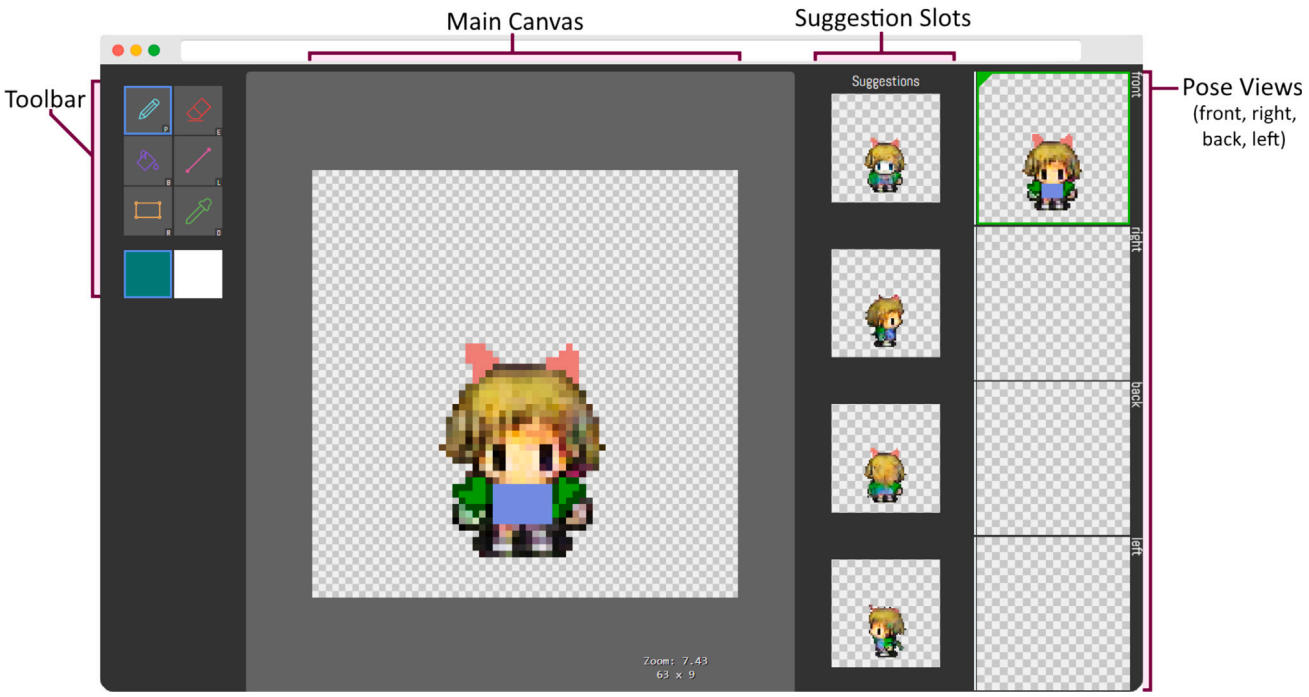


Fig. 12. The sprite editor's user interface, with one view per pose (on the right) and slots for the images generated by the trained models (left of it). A human agent first drew a character facing front, then generated it facing right, back, and left. Last, they generated the character facing front from the suggestion of it facing back.

7.1. User interface and interaction

The sprite editor[2] was developed as a client-side web app, and it can be seen in Fig. 12: in the middle, it contains the main canvas. The panel on the right side includes four "pose views" that depict a character in each pose (front, right, back, and left). Clicking on such a view activates it and brings it to the main canvas so the user can edit it. To the left of the pose views, another panel shows the "suggestion slots", where the images generated by the trained models appear.

As there are four different poses, each one can be a source and a target (except that the target cannot be equal to the source), the editor requires twelve models. They were picked from the ones trained for the evaluation with all sides from the previous section.

An image-to-image translation process is triggered when the user drags a pose view into a suggestion slot, which receives the generated output. Then a corresponding model is selected by leveraging the pose of the dragged view as a source and the pose of the suggestion slot as the target. The dragged image is then provided as input to the model, and the output is recorded on the dropped suggestion slot. The user can edit any pose view, and the models can only change the suggestion slots, never overriding the users' work. The toolbar (at the left) contains a few basic image editing tools, such as a pencil and a bucket, and two color swatches representing foreground and background colors (used with left and right mouse buttons, respectively).

As a system that involves different entities interacting to generate content, we henceforth define the user of the sprite editor as the human agent and the collection of the deep generative models as the computational agent.

Considering the definition of mixed-initiative for content generation from [20] (which we quoted literally in Section 2), we frame our sprite editor as such a system because it contains an iterative loop where the human agent can take the initiative by starting the generation process (dragging "pose views" into the "suggestion slots"). Also, both the human and the computational agents have memory of the current state (what is in the "pose view" or has been dragged into a "suggestion slot") and can contribute to the creation.

7.2. Worked example

To illustrate some possible uses, we walk through an example run of a conversational interaction an artist can have with the sprite editor. This interaction happens in a series of iterative steps, depicted in Fig. 13.

After drawing an initial version of a character facing front (row 1), the human agent wants some variation to it and roughly paints some regions with other colors: pink for the left half of the hair and cyan for the pants (row 2). Then, the human agent asks the computational agent to generate that character facing right. The generator was not trained with examples of images with such large regions in flat color, so it tries to create a right-facing sprite with plausible colors, given the input (row 3).

The human agent liked the outcome of the right-facing character and asked the computational agent to generate it into the front pose again (row 4). However, although the left half of the hair is now colored as intended by the human agent, the right tip also became tinted. So the user manually paints the tip back in flat yellow (row 5) and asks the computational agent to generate it facing back (row 6). Interestingly, the output had the left half of the hair as pink, but as the character faces back, it should have been the right half. The human agent then manually selects the character's hair and flips it horizontally (row 7).

Although this workflow encompasses only a quick interaction scenario, it illustrates some of the abilities of the trained models but also some failure modes and shortcomings. Regarding the interactions that can be considered successful, we could observe the capability of color transfer (hair and pants), at least to some extent, as well as the skill to generate a more plausible sprite (with expected texture/shading) from a lesser one (with flat colors). Next, we present what needs to be addressed and how it can be improved.

[2] Sprite editor: demo at https://fegemo.github.io/sprite-editor-gm/. It downloads the models on the first access, and they have ~1.3 GB.

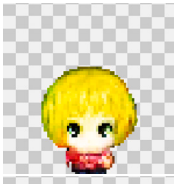
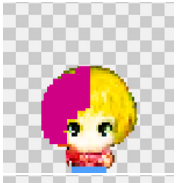
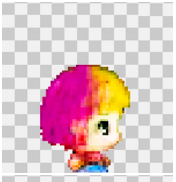
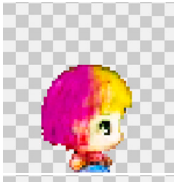
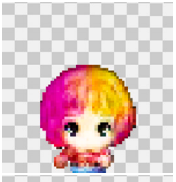
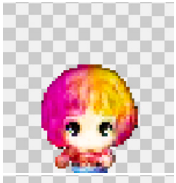
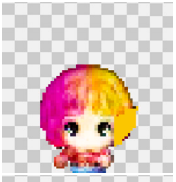
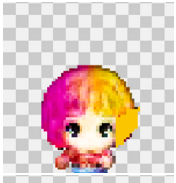
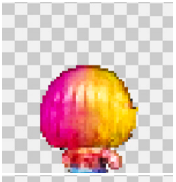
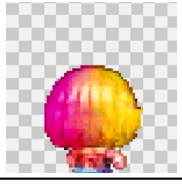
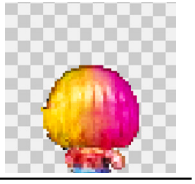
	Input Situation	Human User Thoughts	Action Taken	Result
1		I want my character's hair to have two colors, split in half. Also, their pants should be in a lighter blue.	The human agent roughly paints half of the hair with a hot pink tint and the pants region with a blue tone.	
2		I want the AI to show me how the new hair and pants would look.	The human agent asks the computational agent to generate a sprite facing right (given front).	
3		Interestingly, the hues were changed (hair and pants). Now I need to see it facing front.	The human agent asks the computational agent to generate a sprite facing front (given right).	
4		I like it, but some hot pink went into the right side… let me fix it and see how it looks in the other direction.	The human agent roughly paints the tip of the right half of the hair in yellow.	
5		Now let's see how this character can look from behind.	The human agent asks the computational agent to generate a sprite facing back (from the front).	
6		Hmm… from the back, the blond half should be on the left.	The human agent manually selects the hair and flips it horizontally.	

Fig. 13. A worked example of the mixed-initiative conversational process between the human and computational agents. Each row represents an iteration, with columns depicting (left to right): an input situation, the human agent's thoughts, the action taken, and its output.

7.3. Limitations and next steps

Here we present some limitations of the overall system comprising the editor and the underlying models and discuss some following lines of investigation.

7.3.1. Too many models

A technical aspect of the system architecture is that it requires training a different model for each translation direction (one for front to right, another for front to left, and so on), including in the opposite direction (12 models in total). Each model takes time to train and occupies space that might require download in the initial run (~1.3 GB), as it executes on the client side of the prototype editor.

Additionally, the user might want to generate an improved sprite version in a single step (e.g., front to front) instead of generating in a second direction and then generating it in the source one. However, training a model to translate images in the same pose revealed ineffective, as the model learned an identity function.

Both shortcomings can be addressed by multi-domain models, which can translate images from any source to any domain, such as StarGAN [29]. In such case, a single generator is trained with the whole training data (all poses), and it receives, in addition to the input image, a mask corresponding to the target domain. Besides the potentially reduced size of the trained model, it might also learn more reusable mappings, as a single model sees all poses from the sprites in the training set (four times the number of images in our case, as we have four domains). Having a model that can translate images in the same direction could also prevent the issue seen in the result of iteration 3 of the worked example (Fig. 13). If the human agent could generate an improved version of the character facing front directly, the tip of the right half of the hair would likely not have been tainted in pink.

7.3.2. Insufficient generalization

As previously seen in the results of the experiments, the trained models have some generalization capacity. Still, they might produce images that either contain high-frequency noise or are too smooth, especially when generating images whose shapes deviate from the ones in the training set. The sprite editor made such observations easier to spot, allowing us to interact quickly with the models.

One such problem is the presence of many similar colors in the generated image, which should follow a much smaller palette due to their pixel art nature. Another issue is that some shapes are not recognized by the models and might languish throughout the iterations. For example, the interaction shown in Fig. 12 involved the human agent creating a character with cat ears facing front, then asking the models

to generate it facing: right, back, and left. We can see that the front-to-back model kept the years, but the front-to-right and left generated images with only traces of it.

In the fifth iteration depicted in the worked example (Fig. 13, row 5), the model correctly generated the shape of the character (right arm raised) but did not accurately reflect the colors of each half of the hair. This indicates that color transferring works (e.g., the hair and pants in iteration 2) but might be geometrically wrong in some situations (e.g., iteration 5).

An attempt to improve the generalization capacity of the models is to use an even larger dataset. However, the reduced availability of data is one of the most challenging problems in the task approached in this work. Another possibility is integrating domain-specific knowledge from pixel art when training the models, such as done with a histogram loss term added to the generator [17]. Lastly, other network architectures might also improve the results, such as the VQ-VAE proposed in [18] specifically for pixel art.

7.3.3. Not enough control

One issue that happened while using the system, but that was not portrayed in the worked example, is that the computational agent can change parts of a character that the human agent wanted to keep — and there is currently no way to prevent it. For instance, the human agent might wish to decrease the hair volume of a character by manually erasing some pixels. When the computational agent generates new images from this new version, it might produce the desired result for the hair but also change other parts. For instance, the character's shirt becomes blue. That happens because the model might have been trained with examples that had short hair and a blue shirt. We noticed it is also difficult for the models to change only the eye colors – they usually ignore it. In summary, the models can change more than desired or less.

This undesired behavior might be alleviated by having the models receive, besides the input image, a mask that encodes which pixels should be changed. And the training procedure could randomly create masks for the examples. Plus, the objective function should be altered to account for the masks. A similar situation was proposed in [30] for image inpainting, which uses partial convolutions to accommodate the masked regions.

7.3.4. Loss of opportunity (unused information)

The suggestion of a front-facing character in Fig. 12 was generated by translating the rendered image of it facing back. In contrast, the others had been generated from the front-facing sprite drawn by the human agent. We note that the skin color, as desired by the user, should have a light pink tone. However, as the front-facing suggestion was generated from the back and had little to no visible skin, the model assigned a much lighter color to the character's face. This could be avoided if the model could use information from more than one input domain (e.g., back).

In fact, there have been proposed models that can take more than a single input. One example is CollaGAN [31] which, similarly to StarGAN [29], also uses a single generator/discriminator pair but works with multiple inputs simultaneously. In our scenario, such a model could potentially build even better mappings and hence improve the generalization capacity.

7.3.5. Low initiative

Instead of providing only a single suggestion upon request, the computational agent could generate more options for the user, given the same input. That involves multi-modal models, as they allow random sampling during inference, such as BicycleGAN [32]. Another option would be to generate a new suggestion for each pose already drawn by the human agent, allowing the user to pick the one that best fits their expectation.

Lastly, the proposed computational agent could also proactively suggest the generations without requiring explicit requests from the human agent. To illustrate, all domains could have new suggestions whenever the user changes something, so recommendations would continuously be updated with the user version.

8. Final remarks

In this work, we extended our previous findings from framing the task of generating a pixel art character sprite facing one side from another as an image-to-image translation problem [6]. We proposed an architecture that can perform such translation with varying degrees of generalization. Finally, we trained models with datasets of different sizes and variability and studied different image representations. This paper extends the original work by presenting new experiments with different data augmentation techniques, a larger and more diverse dataset, a post-process step to ensure the use of the intended color palette, and a preliminary study on the use of the trained models in a mixed-initiative sprite editor.

The trained models showed different capabilities depending on the dataset size and variability. From easier to harder tasks, the models could reproduce perceptually identical test images, translate color variations of sprites from the training data with reasonable quality, and generate poor images with either high-frequency noise or faint inner edges for the most unique shapes.

We conducted different experiments with the architecture. First, regarding data augmentation, we observed that hue rotation improves the performance of color transferring, and translation acts as a strong regularizer, preventing the model from overfitting. We also assembled a larger dataset and noticed that training with it enhanced the quality of the generated images for more challenging tasks — when the test set contains characters with unique shapes.

Still, without more training examples, the translations sometimes have insufficient quality. When the model does not generalize well, the generated images suffer from high-frequency color noise and very faint inner edges, resulting in unusable images.

After training models for translating among the poses of front, right, back, and left, we created a mixed-initiative sprite editor that acts as an interface between users and the collection of generative models. A worked example showcased some interactions that can arise between the human and the computational agents and helped uncover some limitations and possible new lines of work for both the editor and the underlying models.

As future work, one can experiment with new architectures that leverage more of the available information, e.g., having a single generator learn to map to and from all poses (such as in StarGAN [29]), or even more, having the generator receive all of the input images already available for a character and use that information to improve the generated image (like CollaGAN [31]). Another line of work is to evolve the sprite editor, allowing the computational agent more initiative and letting users choose different trained models. A user study involving the sprite editor can also bring fruitful observations on which interactions between human and computational agents can be created or refined.

CRediT authorship contribution statement

Flávio Coutinho: Conceptualization, Methodology, Software, Writing – original draft, Validation, Investigation. **Luiz Chaimowicz:** Writing – reviewing & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The text has a link to the mixed-initiative sprite editor proposed in the paper.

Acknowledgments

This work was partially supported by CAPES (finance code 001), CNPq (grant 311900/2020-8) and Fapemig (grant PPM-00563-18).

References

[1] J. Novak, Game Development Essentials: An Introduction, fourth ed., Novy Unlimited, Inc, Santa Monica, 2022.
[2] D. Silber, Pixel Art for Game Developers, CRC Press, 2015.
[3] N. Shaker, J. Togelius, M.J. Nelson, Procedural Content Generation in Games, Springer, 2016, [Online]. Available: http://pcgbook.com/.
[4] G. Smith, J. Whitehead, M. Mateas, Tanagra: Reactive planning and constraint solving for mixed-initiative level design, IEEE Trans. Comput. Intell. AI Games 3 (3) (2011) 201–215, [Online]. Available: http://ieeexplore.ieee.org/document/5887401/.
[5] A. Liapis, G. Smith, N. Shaker, Mixed-initiative content creation, in: Procedural Content Generation in Games: A Textbook and an Overview of Current Research, Springer, 2016, ch. 11.
[6] F. Coutinho, L. Chaimowicz, Generating pixel art character sprites using GANs, in: 2022 21st Brazilian Symposium on Computer Games and Digital Entertainment, Vol. 10, SBGames, IEEE, Natal, Brazil, 2022, pp. 1–6, [Online]. Available, https://ieeexplore.ieee.org/document/9961120/.
[7] P. Isola, J.-Y. Zhu, T. Zhou, A.A. Efros, Image-to-image translation with conditional adversarial networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, Vol. 7, CVPR, 2017-Janua., IEEE, 2017, pp. 5967–5976, [Online]. Available: http://ieeexplore.ieee.org/document/8100115/.
[8] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012) [Online]. Available: http://code.google.com/p/cuda-convnet/.
[9] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, T. Aila, Training generative adversarial networks with limited data, Adv. Neural Inf. Process. Syst. 33 (6) (2020) 12104–12114, [Online]. Available: http://arxiv.org/abs/2006.06676.
[10] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, GANs trained by a two time-scale update rule converge to a local Nash equilibrium, in: Advances in Neural Information Processing Systems, vol. 2017-December, 2017, pp. 6627–6638, [Online]. Available: https://arxiv.org/abs/1706.08500v6.
[11] Y. Pang, J. Lin, T. Qin, Z. Chen, Image-to-image translation: Methods and applications, IEEE Trans. Multimed. (2021) 1, [Online]. Available: https://ieeexplore.ieee.org/document/9528943/.
[12] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9351, 2015, pp. 234–241, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28.
[13] Y.R. Serpa, M.A.F. Rodrigues, Towards Machine-Learning Assisted Asset Generation for Games: A Study on Pixel Art Sprite Sheets, in: 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment, Vol. 2019-Octob, (SBGames), IEEE, Rio de Janeiro, 2019, pp. 182–191, [Online]. Available: https://ieeexplore.ieee.org/document/8924853/.
[14] Z. Jiang, P. Sweetser, GAN-Assisted YUV pixel art generation, in: Australasian Joint Conference on Artificial Intelligence, 2021, pp. 1–12.
[15] A. Gonzalez, M. Guzdial, F. Ramos, Generating gameplay-relevant art assets with transfer learning, in: Proceedings of the AIIDE Workshop on Experimental AI in Games, 2020, pp. 1–7, [Online]. Available: http://arxiv.org/abs/2010.01681.
[16] S. Hong, S. Kim, S. Kang, Game sprite generator using a multi discriminator GAN, KSII Trans. Internet Inf. Syst. 13 (8) (2019) 4255–4269, [Online]. Available: http://itiis.org/digital-library/manuscript/2473.
[17] F. Coutinho, L. Chaimowicz, On the challenges of generating pixel art character sprites using GANs, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 18, (no. 1) 2022, pp. 87–94, [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/21951.
[18] A. Saravanan, M. Guzdial, Pixel VQ-VAEs for improved pixel art representation, in: Experimental AI in Games Workshop (EXAG), 2022, pp. 1–9, [Online]. Available: http://arxiv.org/abs/2203.12130.
[19] A. van den Oord, O. Vinyals, K. Kavukcuoglu, Neural discrete representation learning, in: Advances in Neural Information Processing Systems, vol. 30, 2017.
[20] G. Lai, F.F. Leymarie, W. Latham, On mixed-initiative content creation for video games, IEEE Trans. Games 14 (4) (2022) 543–557, [Online]. Available: https://ieeexplore.ieee.org/document/9779792/.
[21] J. Zhang, R. Taarnby, A. Liapis, S. Risi, DrawCompileEvolve: Sparking interactive evolutionary art with human creations, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9027, 2015, pp. 261–273, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-16498-4_23.
[22] P. Karimi, M.L. Maher, N. Davis, K. Grace, Deep learning in a computational model for conceptual shifts in a co-creative design system, in: Proceedings of the 10th International Conference on Computational Creativity, ICCC 2019, 2019, pp. 17–24, [Online]. Available: https://arxiv.org/abs/1906.10188v1.
[23] F. Ibarrola, O. Bown, K. Grace, Towards co-creative drawing based on contrastive language-image models, in: International Conference on Computational Creativity, vol. 10, 2022.
[24] I. Karth, A.M. Smith, Addressing the fundamental tension of PCGML with discriminative learning, in: ACM International Conference Proceeding Series, Association for Computing Machinery, 2019.
[25] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, in: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings, 2015, [Online]. Available: http://arxiv.org/abs/1511.06434.
[26] M. Mirza, S. Osindero, Conditional generative adversarial nets, 2014, arXiv preprint arXiv:1411.1784, [Online]. Available: http://arxiv.org/abs/1411.1784.
[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-December, 2015, pp. 2818–2826, [Online]. Available: https://arxiv.org/abs/1512.00567v3.
[28] M.D. Fairchild, R.S. Berns, Image color-appearance specification through extension of CIELAB, Color Res. Appl. 18 (3) (1993) 178–190, [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/col.5080180308, https://onlinelibrary.wiley.com/doi/abs/10.1002/col.5080180308, https://onlinelibrary.wiley.com/doi/10.1002/col.5080180308.
[29] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, J. Choo, StarGAN: Unified Generative Adversarial Networks for Multi-domain Image-to-Image Translation, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, 2018, pp. 8789–8797, [Online]. Available: https://ieeexplore.ieee.org/document/8579014/.
[30] G. Liu, F.A. Reda, K.J. Shih, T.C. Wang, A. Tao, B. Catanzaro, Image inpainting for irregular holes using partial convolutions, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11215 LNCS, 2018, pp. 89–105, [Online]. Available: https://arxiv.org/abs/1804.07723v2.
[31] D. Lee, J. Kim, W.-J. Moon, J.C. Ye, CollaGAN: Collaborative GAN for missing image data imputation, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vol. 2019-June, CVPR, IEEE, 2019, pp. 2482–2491, [Online]. Available: https://ieeexplore.ieee.org/document/8953779/.
[32] J.Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A.A. Efros, O. Wang, E. Shechtman, Toward multimodal image-to-image translation, in: Advances in Neural Information Processing Systems, vol. 2017-December, 2017, pp. 466–477, [Online]. Available: https://arxiv.org/abs/1711.11586v4.