Online Appendix: Leveraging Generative AI to Create Visual Content in Digital Advertising

Remi Daviet[1] and Yohei Nishimura[1]

[1]Department of Marketing, Wisconsin School of Business, University of Wisconsin–Madison

April 3, 2026

A VAE-GAN for Layout Generation

The generative model for semantic layouts is a Variational Auto-Encoder combined with a Generative Adversarial Network (VAE-GAN). This architecture is chosen for its ability to learn a smooth and structured latent space while producing high-quality, realistic outputs. The model consists of three main components: an Encoder, a Decoder (or Generator), and a Discriminator.

A.1 Model Architecture

The Encoder network is designed to map an input semantic layout image, L, to a distribution in the 48-dimensional latent space. Specifically, it outputs the parameters of a diagonal Gaussian distribution: a mean vector $\mu \in \mathbb{R}^{48}$ and a log-variance vector $\log(\sigma^2) \in \mathbb{R}^{48}$. A latent vector x_L is then sampled from this distribution using the reparameterization trick, $x_L = \mu + \sigma \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$. The encoder is composed of a series of convolutional layers with max-pooling and batch normalization, which progressively downsample the input to produce the latent parameters.

The Decoder (Generator), G, maps a point x_L sampled from the latent space back to the image space, producing a generated layout $\hat{L} = G(x_L)$. Its architecture is symmetric to the Encoder, consisting of a series of transposed convolutional layers with upsampling and batch normalization to reconstruct the layout's original dimensions.

The Discriminator, D, is a convolutional neural network trained to distinguish between real layout images from the training data and fake layouts generated by the Decoder. It takes a layout image as input and outputs a scalar probability indicating whether the image is real or generated.

A.2 Training Data and Pre-processing

The VAE-GAN is trained on a dataset of semantic layouts extracted from a large corpus of landscape images. The image corpus contains 135,719 pictures, combining 45,719 images tagged as "landscape" from the Flickr API and 90,000 images from the Landscapes High-Quality (LHQ) dataset (Skorokhodov et al., 2021).

For each image, we perform a pre-processing step to obtain a semantic layout. We use a pre-trained DeepLabV2 model (Chen et al., 2017) to perform semantic segmentation, which assigns a categorical label (e.g., sky, tree, water, mountain) to each pixel. The resulting segmentation maps serve as the training data for our VAE-GAN.

A.3 Training Objective

The model's parameters are trained jointly by optimizing a composite loss function that combines objectives from both the VAE and GAN frameworks. The overall objective is to learn a latent space that allows for both accurate reconstruction and the generation of realistic, novel layouts. The loss function consists of three primary components:

1. **The VAE Reconstruction Loss ($\mathcal{L}_{\text{recon}}$):** This term encourages the Decoder to produce outputs that are faithful to the original inputs when passed through the Encoder. It is measured as the pixel-wise cross-entropy between the input layout L and the reconstructed layout $\hat{L}$.

2. **The Latent Loss ($\mathcal{L}_{\text{latent}}$):** This is the Kullback-Leibler (KL) divergence between the Encoder's output distribution, $q(x_L|L)$, and a standard normal prior, $p(x_L) = \mathcal{N}(0, I)$. This term acts as a regularizer, encouraging the Encoder to learn a smooth and well-structured latent space.

$$\mathcal{L}_{\text{latent}} = D_{KL}(q(x_L|L)||p(x_L)) \tag{1}$$

3. **The Adversarial Loss ($\mathcal{L}_{\text{adv}}$):** This loss is derived from the GAN framework. The Decoder (Generator) G is trained to minimize this loss by producing layouts that the Discriminator D misclassifies as real. The Discriminator is simultaneously trained to maximize it by correctly distinguishing between real and generated layouts. This adversarial dynamic pushes the Decoder to generate increasingly realistic images.

$$\mathcal{L}_{\text{adv}} = \min_G \max_D \mathbb{E}_{L\sim p_{\text{data}}(L)}[\log D(L)] + \mathbb{E}_{x_L\sim p(x_L)}[\log(1 - D(G(x_L)))] \tag{2}$$

In practice the expectations are approximated by averaging over batches of data samples. The complete training objective combines these losses to train the Encoder, Decoder, and Discriminator networks simultaneously, ensuring that the model learns to generate a diverse and realistic set of landscape layouts from the learned 48-dimensional creative space.

B Bayesian Neural Network Architecture

The AcceptAI and PerfAI models are both Bayesian neural networks (BNNs) and share the same architecture. This architecture was determined with the simulation exercise and designed to identify a structure that was sufficiently expressive to capture the complex relationships in the creative space while also converging rapidly during training.

The retained model is a fully connected feed-forward neural network. The input layer has 304 nodes, corresponding to the concatenated 48-dimensional layout embedding and the 256-dimensional style embedding. The network has four hidden layers with sizes [128, 64, 32, 8] and a single output node. All hidden layers use a Leaky ReLU activation function, while the final output layer uses a Sigmoid activation function to scale the prediction to a probability for AcceptAI or to a bounded CTR within the range [0-10%] for PerfAI.

To account for the seasonal effects in the PerfAI model, which arise from testing batches sequentially over time, we augment the network with a set of control parameters θ_S. For each batch t, we provide an indicator vector z_t where the current period is indicated by a 1 (and the rest is 0) as an additional input to the model's forward pass. The θ_S parameters are multiplied by the indicator vector and added to the output of the final layer just before the sigmoid activation function. The final output for an embedding w in batch t is therefore computed as

$$\text{sigmoid}(f_{\text{NN}}(x; \theta_{NN}) + z_t^\top \theta_S). \tag{3}$$

This allows the model to learn a separate intercept for each batch, effectively capturing time-based shocks to the baseline CTR without confounding the learned relationship between the visual embeddings and performance.

C Computational Implementation

C.1 Hamiltonian Sequential Monte Carlo

Due to the multi-modal nature of the posterior distribution generated by neural networks, conventional MCMC methods like Metropolis-Hastings or standard Hamiltonian Monte-Carlo (HMC) can struggle to explore the parameter space efficiently. To overcome this, we employ Hamiltonian Sequential Monte Carlo (HSMC), which combines sequential Monte Carlo (particle filtering) with HMC-based mutation steps (Burda and Daviet, 2022). This method is particularly adept at sampling from high-dimensional, multi-modal posteriors.

The HSMC algorithm maintains a set of H weighted samples, or "particles," $\{\theta_h, \omega_h\}_{h=1}^H$, which represent the posterior distribution $p(\theta|\mathcal{D})$. When new data arrives, the algorithm proceeds in three steps:

1. **Correction:** The weight ω_h of each particle θ_h is updated based on the likelihood of the new data under that particle. This is an application of importance sampling, where particles that better explain the new data are assigned higher weights.

2. **Selection:** A new set of particles is resampled from the old set based on their updated weights. Particles with higher weights are more likely to be duplicated, while those with

low weights are eliminated. This step focuses computational effort on more promising regions of the parameter space.

3. **Mutation:** Each resampled particle is evolved using several steps of an HMC transition kernel. This gradient-informed step allows the particles to efficiently explore the local geometry of the posterior distribution, leading to a new set of diverse and high-probability samples.

This sequential process allows the set of particles to adaptively track the posterior distribution as more data is incorporated. In our application, we set $H = 128$ and perform 16 mutation steps at each iteration.

C.2 Computing Information Gain

The core challenge is to efficiently approximate the KL divergence between the prior, $p(\mathcal{X}|\mathcal{D}_{\text{old}})$, and posterior, $p(\mathcal{X}|\mathcal{D}_{\text{new}})$, distributions over the optimal sets. The KL divergence is defined as an integral over the posterior distribution, which is itself intractable.

To solve this, we turn to importance sampling, a Monte Carlo technique for approximating expectations with respect to a target distribution (our posterior) using samples drawn from a different proposal distribution (our prior). In general, to estimate the expectation of a function $f(x)$ under a target distribution $p(x)$ using samples $\{x_h\}_{h=1}^H$ from a proposal distribution $q(x)$, the formula is:

$$\mathbb{E}_{p(x)}[f(x)] = \int f(x)p(x)dx \tag{4}$$

$$\approx \frac{1}{H}\sum_{h=1}^{H} f(x_h)\frac{p(x_h)}{q(x_h)} \tag{5}$$

where the ratio $w_h = p(x_h)/q(x_h)$ is called the importance weight.

Our application of this principle begins by generating samples from the proposal distribution, which in our case is the prior over optimal creatives, $p(\mathcal{X}|\mathcal{D}_{\text{old}})$. To do this, we find the optimal embedding vector x_h^* for each of the H particles $\{\theta_h\}_{h=1}^H$ from our HSMC sampler by running a gradient ascent optimization for each particle. This resulting collection of optimal points, $\{x_h^*\}_{h=1}^H$, represents our discrete sample. For computational purposes, we then create a smooth density approximation from these samples, $\hat{p}(\mathcal{X}|\mathcal{D}_{\text{old}})$, using a Gaussian Kernel Density Estimator (KDE).

Next, we must approximate the importance weights, which requires the posterior density, $p(\mathcal{X}|\mathcal{D}_{\text{new}})$. We approximate this density using the same weights as those generated during a HSMC correction step. Instead of re-running the expensive optimization for every particle to obtain a new set of samples from the posterior, we can approximate the posterior density $\hat{p}(\mathcal{X}|\mathcal{D}_{\text{new}})$ by constructing a *weighted* Gaussian KDE over our existing set of optimal points $\{x_h^*\}_{h=1}^H$. Each point x_h^* is weighted by the $\omega_h \propto p(\mathcal{D}_{\text{new}}|\theta_h)$ of its corresponding particle.

With these two density approximations, the KL divergence can be estimated using the importance sampling framework, where our function is $f(\mathcal{X}) = \log \frac{p(\mathcal{X}|\mathcal{D}_{\text{new}})}{p(\mathcal{X}|\mathcal{D}_{\text{old}})}$. This gives the

estimator:

$$D_{KL}\left(p(\mathcal{X}|\mathcal{D}_{\text{new}})\|p(\mathcal{X}|\mathcal{D}_{\text{old}})\right) \tag{6}$$

$$= \mathbb{E}_{p(\mathcal{X}|\mathcal{D}_{\text{new}})}\left[\log \frac{p(\mathcal{X}|\mathcal{D}_{\text{new}})}{p(\mathcal{X}|\mathcal{D}_{\text{old}})}\right] \tag{7}$$

$$\approx \frac{1}{H}\sum_{h} \frac{\hat{p}(x_h^* \in \mathcal{X}|\mathcal{D}_{\text{new}})}{\hat{p}(x_h^* \in \mathcal{X}|\mathcal{D}_{\text{old}})} \log \frac{\hat{p}(x_h^* \in \mathcal{X}|\mathcal{D}_{\text{new}})}{\hat{p}(x_h^* \in \mathcal{X}|\mathcal{D}_{\text{old}})}, \tag{8}$$

where the densities are evaluated using the unweighted KDE (for the prior) and the weighted KDE (for the posterior) at each sample point x_h^*. This integration approach is numerically stable as it is evaluated at points where the respective densities are not close to zero, avoiding instabilities from the log and division operations.

C.3 Approximating Expected Information Gain

The objective function for selecting the next batch requires calculating the *expected* information gain, which involves an intractable integral over all possible future data outcomes for a candidate batch X^B. We approximate this expectation using Monte Carlo simulation. The simulation proceeds by generating a set of H plausible future datasets, where each plausible future is generated from the predictive distribution of one of the H particles from our HSMC sampler.

For the PerfAI model, generating a plausible outcome for a given particle θ_h involves first predicting the expected CTRs $\mu_h = f_{\text{PerfAI}}(X^B; \theta_h)$. We then obtain the corresponding number of clicks y_h by using the rounded expected value (for 4000 impressions), which keeps the simulation deterministic given the particle. A similar process is used for AcceptAI, where predicted probabilities are converted into expected acceptability labels, interpreted as the expected number of times it would be labeled as acceptable when presented to a set of managers. This process results in a set of H distinct, plausible future datasets, $\{y_h\}_{h=1}^H$.

The next step is to calculate the information gain that would be achieved for each of these simulated futures. For a given plausible outcome y_h, we can calculate the corresponding KL divergence, $D_{KL}(y_h)$, by performing a "what-if" analysis: if we were to observe this outcome, the importance weight for every other particle $\theta_{h'}$ would be updated to $\omega_{h'}(y_h) \propto p(y_h|\theta_{h'})$. These updated weights would then be used to compute a new posterior distribution over the optimal set, $\hat{p}(\mathcal{X}|y_h)$, allowing for the calculation of the resulting KL divergence.

Finally, the expected information gain is approximated by taking the average of the KL divergences calculated across all simulated future outcomes:

$$\mathbb{E}[D_{KL}] \approx \frac{1}{H}\sum_{h=1}^{H} D_{KL}(y_h)$$

This provides a computationally feasible estimate of the objective function, which can then be maximized using a standard gradient-based optimizer to find the most informative batch of creatives to test next.

C.4 Optimization Approach

To find the optimal batch of creatives X^B that maximizes the expected information gain, we need an efficient optimization algorithm. The creative space of embeddings, w, is continuous. Critically, because our entire computational pipeline—from the BNNs to the KDEs used to approximate the information gain—is composed of differentiable functions, the expected information gain objective function $G(X^B)$ is differentiable with respect to the embedding vectors in the batch X^B. This allows us to use gradient-based optimization methods.

We employ gradient ascent, a standard iterative optimization algorithm. The core principle of gradient ascent is to iteratively take small steps in the direction of the steepest increase of the objective function. The direction of steepest ascent is given by the gradient of the function, denoted ∇G.

The process starts with a randomly initialized batch of creatives, X_0^B. At each iteration k, the batch is updated according to the following rule:

$$X_{k+1}^B = X_k^B + \alpha \nabla G(X_k^B)$$

Here, α is the learning rate, a small positive scalar that controls the size of each step. The gradient $\nabla G(X_k^B)$ is a matrix containing the partial derivatives of the objective function with respect to each element of each embedding vector in the batch. When optimizing for a full batch, this differentiation is applied simultaneously to all creatives in the batch. This process is repeated until the value of the objective function no longer improves, at which point the algorithm has converged to a local maximum, yielding the next batch of creatives to be tested.

D Visualization of the Creative Space

To provide intuition for the high-dimensional search process, we use Parametric UMAP to project the 304-dimensional embedding space into a 2D visualization. We create separate projections for the 48-dimensional layout embeddings and the 256-dimensional style embeddings, as this better reveals the distinct search patterns for scene structure versus aesthetic qualities.

Figure 1 shows this UMAP space and the progression of the tested visuals over the nine training trials. Figure 2 shows examples of generated landscapes at the corresponding locations of the UMAP space. The algorithm's search for promising layouts converges to a specific region of the space, while the search for styles remains more exploratory, covering a wider area.

Figure 3 visualizes the predicted top quartiles for brand acceptability and CTR performance. We note that the dense regions differ for the 2 criteria, resulting in limited overlap between these two sets. This underscores the challenge of finding creatives that satisfy both criteria simultaneously.

Figure 1: UMAP representation of embedding vectors for the training data (dots) and tested visuals from each trial (squares, colored by trial number).

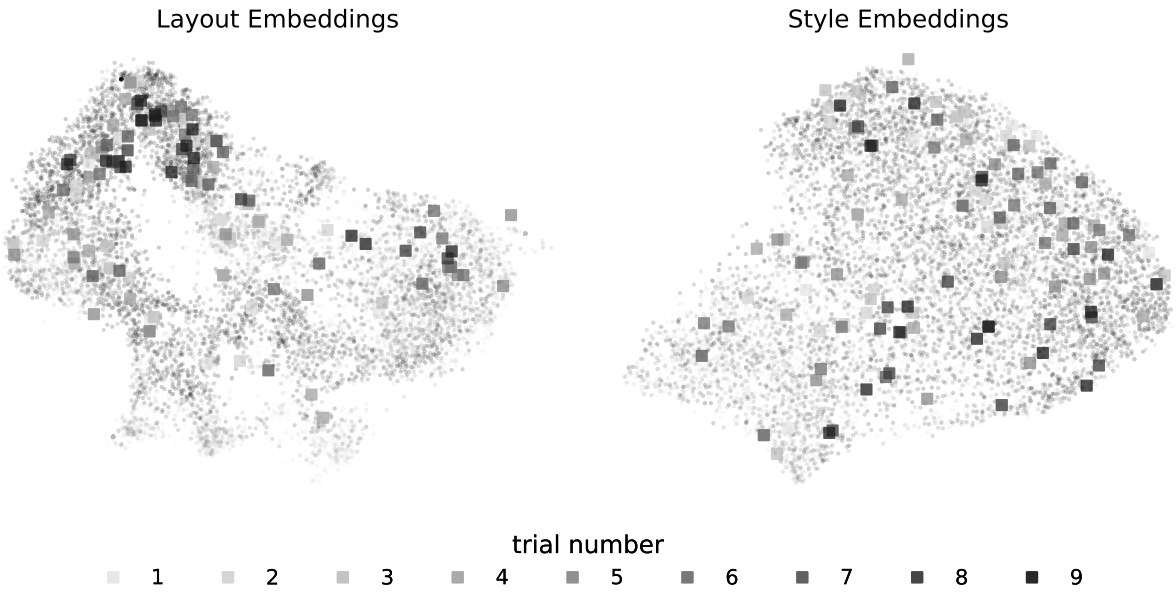
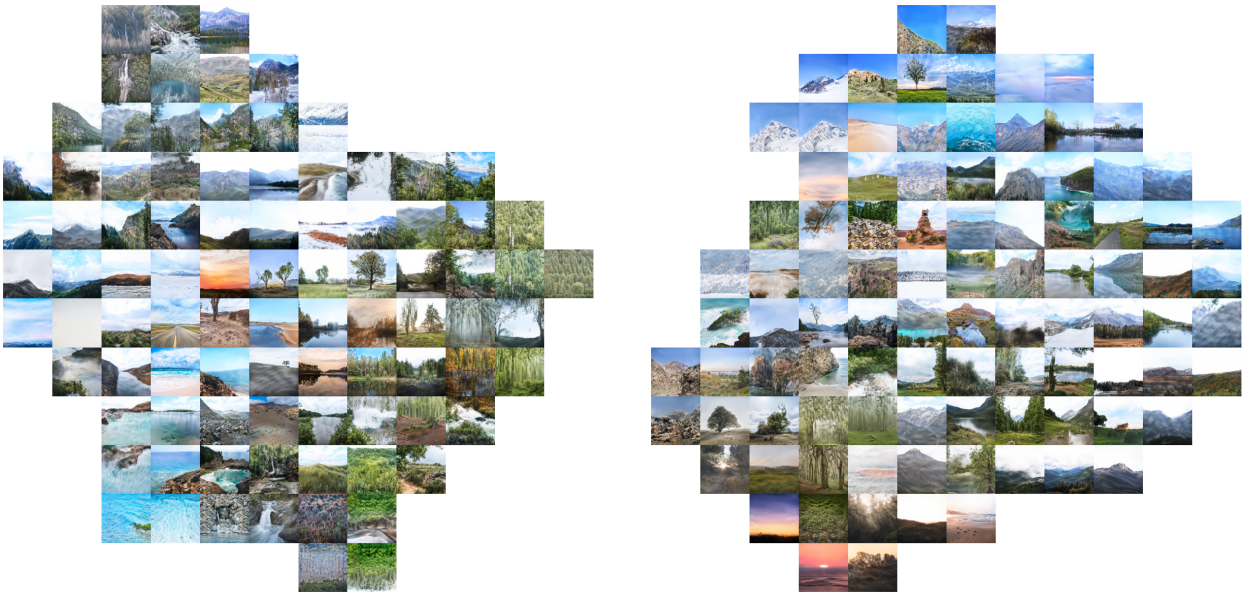


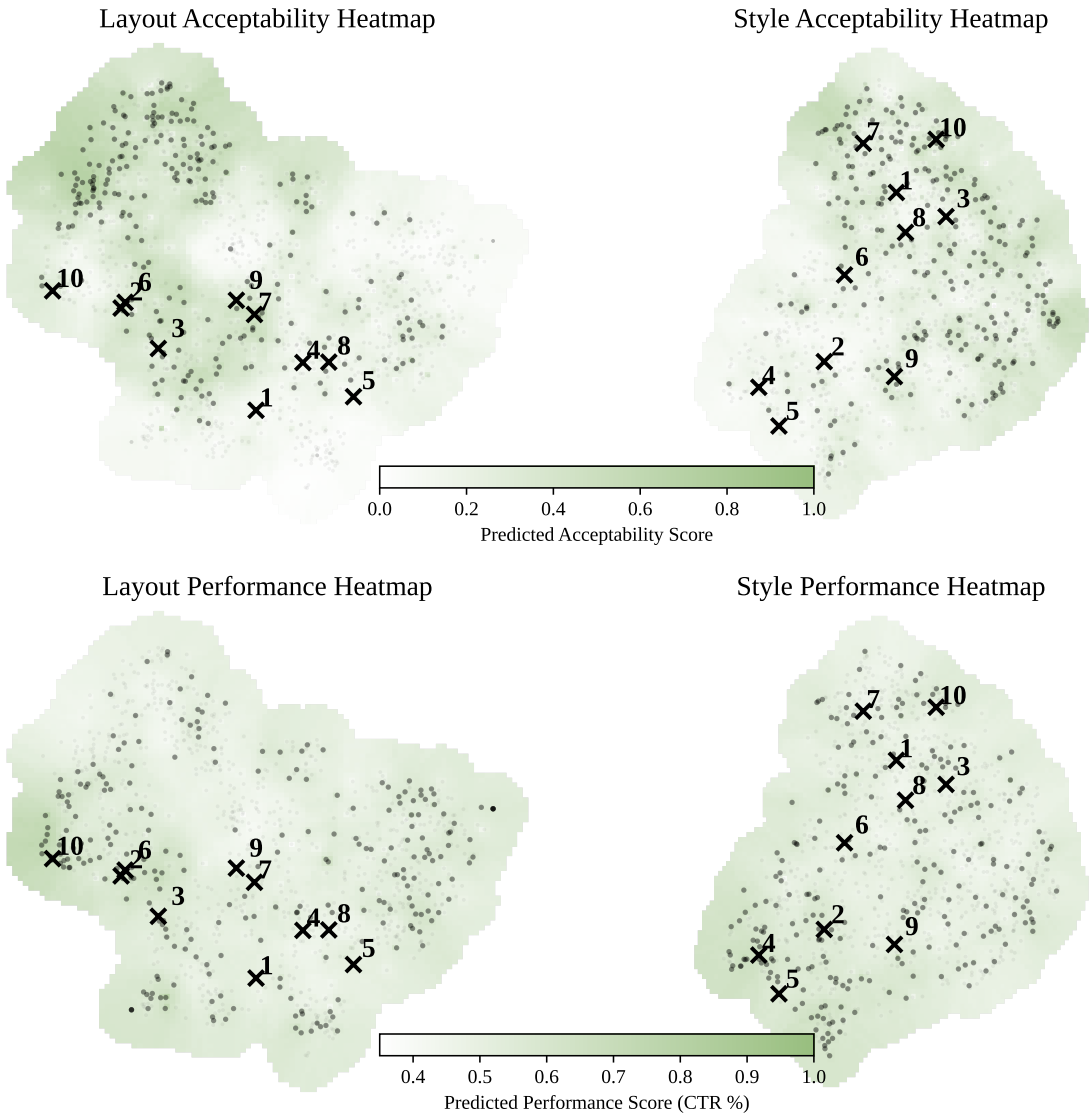
Figure 2: Generated pictures and their position in the UMAP spaces.



(a) Layout Space

(b) Style Space

Figure 3: Predicted acceptability scores (top) and performance scores (bottom).



Notes: Points represent top quartiles and crosses indicate the location of the final creatives sampled by probability matching.

References

Burda, M. and R. Daviet (2022). Hamiltonian sequential monte carlo with application to consumer choice behavior. *Econometric Reviews 618*, 21.

Chen, L.-C., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence 40*(4), 834–848.

Skorokhodov, I., G. Sotnikov, and M. Elhoseiny (2021). Aligning latent and image spaces to connect the unconnectable. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 14144–14153.