DALLE

A Preprint

1 Main

Authors suggest 2 stage training process:

- Discrtete variational autoencoder to compress each 256x256 image into 32x32 grid of image tokens, each element of which can be one of 8192 values.
- The input text (like a caption "a red apple on a table") is processed by a BPE tokenizer. This tokenizer breaks the text down into a sequence of up to 256 pieces. These pieces can be whole words, common subwords, or individual characters, depending on what the BPE algorithm learned from its training data. Each of these pieces is a "text token. Concatenate and train an autoregressive transformer.

Procedure can be explained as maximazing ELB.

1.1 ELB

ELB - Evidence lower bound Since we can't directly compute and maximize the probability of the observed data $P(x)$, we use a function called the Evidence Lower Bound (ELB), which is easier to compute.

The ELB has a key property:

$$\text{ELB} \leq \log P(x)$$

(We often work with log-probabilities for numerical stability and mathematical convenience.)

The strategy: By maximizing the ELB, we are effectively pushing up the true log-evidence $\log P(x)$. The closer the ELB is to $\log P(x)$, the better our approximation.

What the ELB Consists of (Standard VAE)

The ELB is derived using Jensen's inequality and an approximate posterior distribution $Q(z|x)$, which attempts to infer the latent variables z that could have generated x. It typically breaks down into two terms:

$$\text{ELB} = \mathbb{E}_{Q(z|x)}[\log P(x|z)] - D_{\text{KL}}(Q(z|x) \parallel P(z))$$

Reconstruction Term: $\mathbb{E}_{Q(z|x)}[\log P(x|z)]$

- $P(x|z)$ is the likelihood of generating x from latent variable z, modeled by the decoder.
- $\mathbb{E}_{Q(z|x)}[\cdot]$ denotes expectation over z sampled from $Q(z|x)$.
- Maximizing this term encourages the model to learn latent variables z that can accurately reconstruct x.

KL Divergence Term: $D_{\text{KL}}(Q(z|x) \parallel P(z))$

- $P(z)$ is the prior distribution (e.g., a standard Gaussian).

- $Q(z|x)$ is the approximate posterior learned by the encoder.
- The KL divergence measures how different $Q(z|x)$ is from $P(z)$.
- Minimizing this term (since it is subtracted) acts as a regularizer, keeping $Q(z|x)$ close to $P(z)$ and encouraging a structured, smooth latent space.

Why Maximize the ELB?

- Maximizing the reconstruction term ensures the model can generate data similar to the input.
- Minimizing the KL divergence keeps the latent space well-behaved and prevents overfitting.
- Mathematically, it is equivalent to minimizing $D_{\mathrm{KL}}(Q(z|x) \parallel P(z|x))$, bringing our approximation closer to the true posterior.
- Since ELB $\leq \log P(x)$, maximizing ELB indirectly maximizes the likelihood of the data.

Example from the Paper (Page 2)

The paper states:

"The overall procedure can be viewed as maximizing the evidence lower bound (ELB) on the joint likelihood of the model distribution over images x, captions y, and the tokens z for the encoded RGB image."

They model the joint distribution as:

$$p_{\theta,\psi}(x, y, z) = p_\theta(x|y, z) \cdot p_\psi(y, z)$$

This yields the lower bound:

$$\log p_{\theta,\psi}(x, y) \geq \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|y, z) - \beta D_{\mathrm{KL}}(q_\phi(y, z|x) \parallel p_\psi(y, z))\right]$$

- $q_\phi(z|x)$ is the encoder (Stage 1), producing tokens z from input x.
- $p_\theta(x|y, z)$ is the decoder that reconstructs x given text y and image tokens z.
- $p_\psi(y, z)$ is the prior over captions and tokens, modeled by a transformer (Stage 2).
- The first term inside the expectation is the reconstruction loss.
- The second term is the KL divergence, acting as a regularizer.

Thus, when they say they are "maximizing the ELB," they are referring to this well-established variational inference framework that trains the model to both reconstruct data and learn a structured latent representation.

Training stage 1: First of all we train cookbook. We maximize ELB with respect to θ and ϕ. Initially setting equal prior $p_\psi \sim U$ and q_ϕ as categorical distribution.

Discrete Latents and the Gumbel-Softmax Relaxation

When the approximate posterior distribution q_ϕ is discrete, optimizing the Evidence Lower Bound (ELB) becomes non-trivial. This is because the standard reparameterization trick, commonly used for continuous latent variables, cannot be directly applied.

Previous approaches, such as those by Oord et al. (2017) and Razavi et al. (2019), address this challenge using online cluster assignment procedures in conjunction with the straight-through estimator (Bengio et al., 2013), which enables gradients to pass through non-differentiable sampling operations during backpropagation.

In this work, we adopt the Gumbel-Softmax relaxation (Jang et al., 2016; Maddison et al., 2016), which enables differentiable sampling from categorical distributions. Instead of sampling directly from q_ϕ, we sample from a continuous relaxation q_ϕ^τ, where $\tau > 0$ is a temperature parameter. As $\tau \to 0$, the softmax output becomes increasingly peaked, and the relaxed distribution converges to the categorical distribution:

$$q_\phi^\tau(z_i) = \frac{\exp\left((\log \alpha_i + g_i)/\tau\right)}{\sum_j \exp\left((\log \alpha_j + g_j)/\tau\right)},$$

where $g_i \sim \text{Gumbel}(0, 1)$ and α_i are unnormalized logits.

This formulation allows us to compute gradients with respect to ϕ using standard backpropagation.

The likelihood under the generative model p_θ is evaluated using the log-Laplace distribution, which better models the data distribution in our context. A detailed derivation is provided in Appendix A.3.

The Core Problem: Differentiable Sampling from Categorical Distributions

A key challenge arises when we want neural networks to make discrete decisions—such as choosing a category from K options—while training via backpropagation, which fundamentally requires smooth, differentiable operations.

Assume we have K possible categories (e.g., K "visual words" as in DALL-E). The encoder network, parameterized by ϕ, observes input data and outputs logits for each category: $\log \alpha_1, \log \alpha_2, \ldots, \log \alpha_K$.

Objective: Sampling from a Categorical Distribution

Our goal is to sample a discrete category z based on these logits. Typically, we compute probabilities via the softmax:

$$p_i = \frac{\exp(\log \alpha_i)}{\sum_j \exp(\log \alpha_j)} = \frac{\alpha_i}{\sum_j \alpha_j}$$

and then draw a sample z from the categorical distribution defined by these probabilities.

However, sampling a one-hot vector (where $z_i = 1$ for the chosen category and 0 otherwise) is a discrete, non-differentiable operation. Small changes in the logits may not affect the sample at all—or cause abrupt jumps—resulting in zero or undefined gradients with respect to ϕ.

The Gumbel-Max Trick: A Discrete Reparameterization

To bridge this gap, the Gumbel-Max trick offers a reparameterization of categorical sampling:

- For each category i, draw $g_i \sim \text{Gumbel}(0, 1)$, where $g_i = -\log(-\log u_i)$ and $u_i \sim \text{Uniform}(0, 1)$.
- Compute perturbed scores: $x_i = \log \alpha_i + g_i$.
- Select: $z = \arg\max_i x_i$.

This correctly samples from the categorical distribution defined by p, but $\arg\max$ is still not differentiable.

The Gumbel-Softmax Relaxation: A Smooth Approximation

To enable gradient-based optimization, we replace the non-differentiable $\arg\max$ with a softmax operation. This results in a relaxed, differentiable approximation:

$$y_i = \frac{\exp\left((\log \alpha_i + g_i)/\tau\right)}{\sum_j \exp\left((\log \alpha_j + g_j)/\tau\right)}$$

The resulting vector $y = (y_1, \ldots, y_K)$ satisfies:

- $y_i \geq 0$ for all i,
- $\sum_i y_i = 1$.

Thus, y defines a probability distribution—a "soft"version of a one-hot vector—and is a sample from the Gumbel-Softmax distribution (also called the Concrete distribution).

The Role of Temperature τ

The temperature parameter τ controls how close the softmax distribution is to a one-hot vector:

- As $\tau \to 0$: The distribution sharpens, and y approaches a one-hot vector aligned with the maximum perturbed logit.
- As $\tau \to \infty$: The output becomes smoother and approaches a uniform distribution.

Why This Works: Differentiability and Approximation

The Gumbel-Softmax relaxation provides a differentiable approximation to categorical sampling. This is useful for gradient-based optimization because:

- The entire computation of y_i (using $\log \alpha_i$ and g_i) is differentiable with respect to $\log \alpha_i$, and hence to ϕ.
- As $\tau \to 0$, the soft sample y closely approximates a discrete sample from the target categorical distribution.

This enables us to train models with discrete latent variables using standard backpropagation techniques.

A.1 Architecture

The dVAE encoder and decoder are convolutional ResNets [? ?] with bottleneck-style residual blocks. Most convolutions are 3×3, with 1×1 used in skip connections and outer layers. The encoder starts with a 7×7 convolution and ends with a 1×1 convolution producing logits of shape $32 \times 32 \times 8192$. Max-pooling is used for downsampling (better ELB than average-pooling), and the decoder uses nearest-neighbor upsampling. See `dvae/encoder.py` and `dvae/decoder.py` for implementation.

A.2 Training

dVAE is trained with the same dataset and augmentations as the transformer (Listing 1). The following schedules use cosine annealing:

- KL weight β: increased from 0 to 6.6 over 5000 steps.
- Relaxation temperature τ: annealed from 1 to 1/16 over 150,000 steps (linear annealing led to divergence).
- Learning rate: reduced from 1×10^{-4} to 1.25×10^{-6} over 1.2M steps.

Optimization uses AdamW [?] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, weight decay 10^{-4}, and EMA with decay 0.999. The ELB's reconstruction term is over $256 \times 256 \times 3$ pixel values; the KL term spans the 32×32 spatial grid. Loss is normalized by $256 \times 256 \times 3$, making KL's effective weight $\beta/192$. Training uses mixed-precision with global loss scaling on 64 V100 GPUs (16GB), batch size 8 per GPU (total 512), for 3M updates.

A.3 Logit-Laplace Distribution

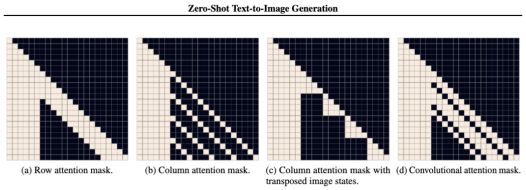
Standard ℓ_1 and ℓ_2 losses correspond to Laplace and Gaussian likelihoods, both defined over $\mathbb{R}$, which mismatches the bounded pixel domain $[0, 255]$. To address this, we use a logit-Laplace distribution, obtained by applying a sigmoid to a Laplace-distributed variable:

$$f(x \mid \mu, b) = \frac{1}{2bx(1-x)} \exp\left(-\frac{|(x) - \mu|}{b}\right),$$

defined on $(0, 1)$. We use the log-density of this distribution as the reconstruction loss. The decoder outputs six channels: three for μ (RGB), and three for $\log b$. Input pixels $x \in [0, 255]$ are rescaled to $(\varepsilon, 1 - \varepsilon)$ via:

$$\phi(x) = \frac{1 - 2\varepsilon}{255} x + \varepsilon, \quad \text{with } \varepsilon = 0.1.$$

This avoids numerical issues from $x(1 - x)$ in the denominator. For visualization or metrics, reconstructions use $\hat{x} = \phi^{-1}(\sigma(\mu))$, ignoring $\log b$.



(a) Row attention mask. (b) Column attention mask. (c) Column attention mask with transposed image states. (d) Convolutional attention mask.

Figure 11. Illustration of the three types of attention masks for a hypothetical version of our transformer with a maximum text length of 6 tokens and image length of 16 tokens (i.e., corresponding to a 4×4 grid). Mask (a) corresponds to row attention in which each image token attends to the previous 5 image tokens in raster order. The extent is chosen to be 5, so that the last token being attended to is the one in the same column of the previous row. To obtain better GPU utilization, we transpose the row and column dimensions of the image states when applying column attention, so that we can use mask (c) instead of mask (b). Mask (d) corresponds to a causal convolutional attention pattern with wraparound behavior (similar to the row attention) and a 3×3 kernel. Our model uses a mask corresponding to an 11×11 kernel.

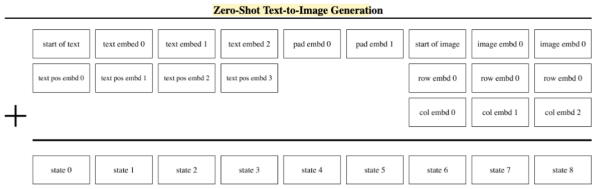


Figure 10. Illustration of the embedding scheme for a hypothetical version of our transformer with a maximum text length of 6 tokens. Each box denotes a vector of size $d_{\text{model}} = 3968$. In this illustration, the caption has a length of 4 tokens, so 2 padding tokens are used (as described in Section 2.2). Each image vocabulary embedding is summed with a row and column embedding.

B.1 Architecture

We use a decoder-only sparse Transformer similar to [?], with broadcasted row and column embeddings for image token context (see Fig. 10). The model has 64 layers with 62 heads per layer, each head having a hidden size of 64.

Three sparse attention masks are used (see Fig. 11):

- Row attention is the default.
- Column attention is used in every 4th layer, when $i - 2 \bmod 4 = 0$.
- Convolutional attention is applied only in the final layer, slightly improving performance.

B.2 Training

Before encoding images with the dVAE, we apply a separate set of augmentations (Listing 2). Captions use BPE encoding with 10% dropout.

Training uses per-resblock scaling (Sec. 2.4), gradient compression (Sec. 2.5) with total rank 896 (112 per GPU), and AdamW with:

$$\beta_1 = 0.9, \quad \beta_2 = 0.96, \quad \epsilon = 10^{-8}, \quad \text{weight decay} = 4.5 \times 10^{-2}.$$

Gradients are clipped by norm (threshold 4), mainly during warm-up. Most Adam moments are stored in 16-bit formats:

- Mean: 1-6-9 bit format.
- Variance: 0-6-10 bit format, clipped to a max of 5.

Model parameters are updated using exponentially weighted iterate averaging every 25 steps with decay 0.99.

Training is performed on 1024 NVIDIA V100 GPUs (16GB), batch size 1024, for 430k updates. Learning rate ramps up linearly to 4.5×10^{-4} over the first 5k steps, then is halved each time the loss plateaus (5x in total). Final LR is $4.5 \times 10^{-4}/32$.

Validation used 606k images; no overfitting was observed.

Algorithm 1 DALL·E Text-to-Image Pipeline

Require: Text prompt y
Ensure: Generated image $\hat{x}$

Stage 1: Train Discrete VAE (dVAE)
Train encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$ using ELB loss:

$$\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{\mathrm{KL}}(q_\phi(z|x) \parallel p(z))$$

q_ϕ: maps continuous image $x \in \mathbb{R}^{H\times W\times 3}$ to a grid of discrete tokens $z \in \{1,\ldots,K\}^{H'\times W'}$
p_θ: reconstructs image x from discrete token grid z

Stage 2: Train Transformer to Model Image Tokens Conditioned on Text
Tokenize text prompt $y = (y_1,\ldots,y_T)$ using BPE
Encode training image x to discrete image tokens z using q_ϕ
Train Transformer T_ψ to autoregressively model:

$$p_\psi(z|y) = \prod_{t=1}^{L} p_\psi(z_t|z_{<t}, y)$$

Stage 3: Generation
Tokenize input prompt y
Use Transformer to sample discrete image tokens $\hat{z} \sim p_\psi(z|y)$
Decode image from tokens: $\hat{x} \leftarrow p_\theta(x|\hat{z})$
return $\hat{x}$