# VAE Explainer: Supplement Learning Variational Autoencoders with Interactive Visualization

Donald Bertucci and Alex Endert

Georgia Institute of Technology

**Abstract:** Variational Autoencoders are widespread in Machine Learning, but are typically explained with dense math notation or static code examples. This paper presents VAE Explainer, an interactive Variational Autoencoder running in the browser to supplement existing static documentation (e.g., Keras Code Examples). VAE Explainer adds interactions to the VAE summary with interactive model inputs, latent space, and output. VAE Explainer connects the high-level understanding with the implementation: annotated code and a live computational graph. The VAE Explainer interactive visualization is live at https://xnought.github.io/vae-explainer and the code is open source at https://github.com/xnought/vae-explainer.

## 1 Introduction

Variational Autoencoders (VAE) [11] compress data effectively and produce a latent space that can be nicely interpolated through. However, VAEs are conceptually more difficult than regular Autoencoders (i.e., Reparameterization) and are described with dense mathematical notation [11]. Furthermore, documentation or notebooks on VAEs include code, but no live interactive exploration to show off key pieces of the VAE [2, 5, 9, 12, 16, 18].

VAE Explainer doesn't aim to replace existing examples, but to supplement them with interactive visualization. VAE Explainer specifically builds off of the demonstrated educational effectiveness of interactive explainers like CNN Explainer [17], Diffusion Explainer [14], and Transformer Explainer [3] but to explain VAEs.

With VAE Explainer, we don't display low-level details first. We hide the math notation and provide an interactive high-level overview (see Figure 1). For example, a user can hand-draw the input and view how the encoded distribution and reconstruction changes. When a user is ready, they can display low-level implementation details such as the Log-Var Trick [15] and Reparameterization Trick [11] (see Figure 2). For simplicity and familiarity, we use the MNIST Digit dataset [19] to align with existing documentation on VAEs [2, 5].

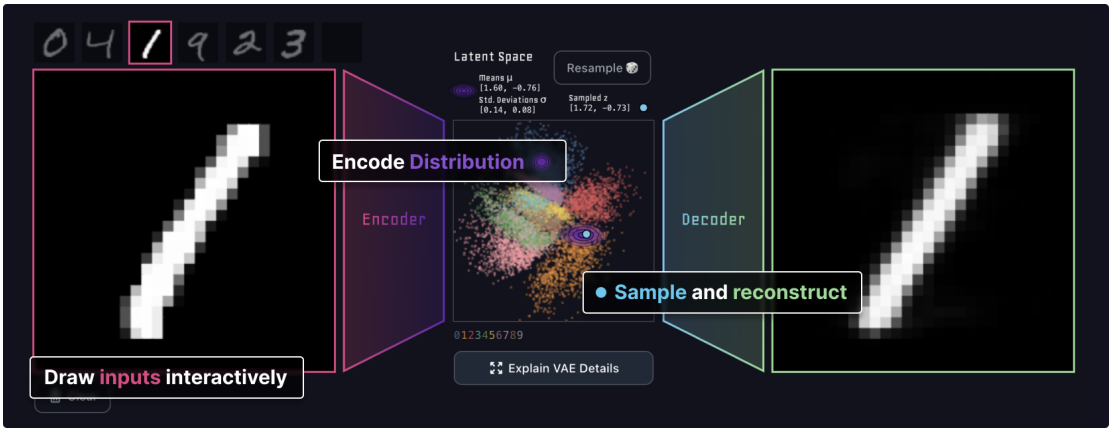


Figure 1: Users can draw a digit as input and the VAE runs in real-time. VAE Explainer displays the encoded distribution on top of the latent space. Then, we sample a point from the distribution and decode into the reconstruction.

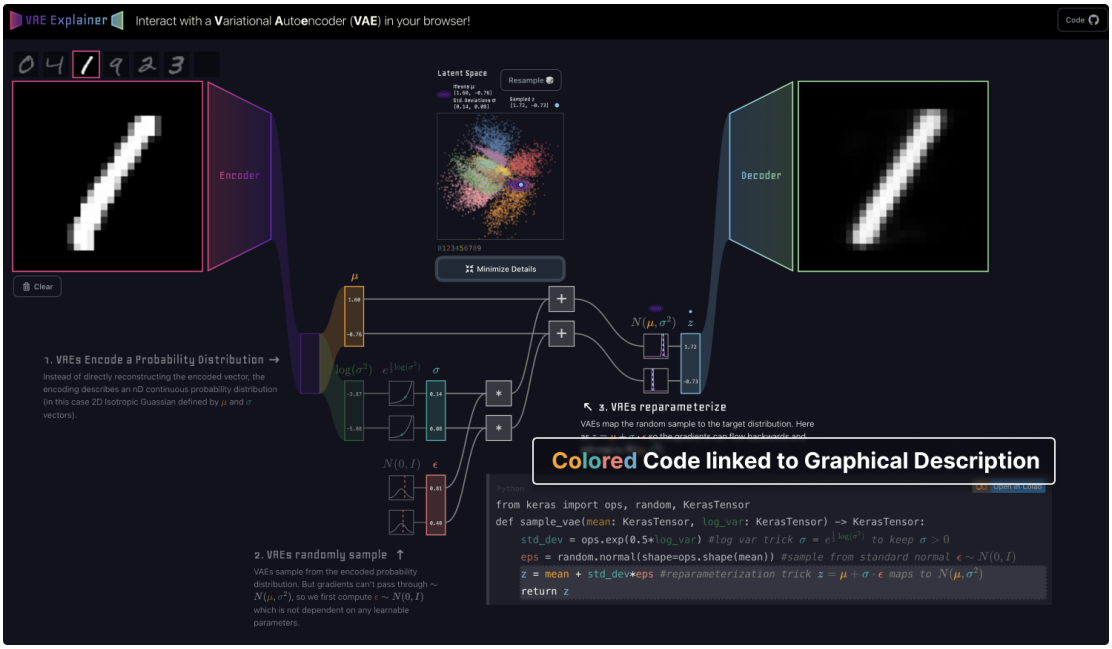


Figure 2: Users can click "Explain VAE Details" to show annotated code connected to a computational graph. Hovering over lines in the code will highlight portions of the graph.

To be very specific, this paper contributes the following:

- A high-level summary view of a VAE with interactive inputs and latent space (Section 2.1).
- A low-level graph view that describes implementation details (i.e., Log-Var [15] and Reparameterization [11] Tricks) with code and an annotated computational graph (Section 2.2).
- Open source and browser implementation to make VAE Explainer accessible to anyone with a browser (Section 3).

# 2 System

This section describes the entire VAE Explainer tool in two subsections: the **High-Level Summary View** (Section 2.1) and the **Low-Level Graph View** (Section 2.2).

## 2.1 High-Level Summary View

To explain the main ideas from static documentation on VAEs, VAE Explainer distills the main point of a VAE as encoding a probability distribution of the input data, which we then sample and reconstruct (see Figure 1).

The encoder takes a hand-written digit input and encodes the data as a two-dimensional isotropic normal distribution. We chose 2D so the latent space could be easily visualized by humans. The distribution itself is displayed directly on the latent space in gradually increasing and diffuse purple circles (see middle of Figure 1). Since the distribution has no covariance, it'll always be stretched in the vertical or horizontal direction. When you change the input data, you'll see that the distribution changes location and shape to other places in the latent space. For example on the left side in Figure 3, as we draw the digit "0" as the input, the latent space gradually interpolates through "9" and"2" regions before finding itself in the "0" region.

The latent space itself has many colored points in the background. These points are training data with labels from the MNIST dataset [19]. When a user hovers over the latent space, they


Figure 3: **Left side**: as we draw the digit "0" in the input, the encoded distribution changes location and size to represent the distribution of possible "0"s. **Right side**: as we hover the latent space and change the sampled point, we interpolate the reconstruction.

can change the sampled blue point to anywhere in space and see the reconstructed output. For example, on the right side in Figure 3, by hovering and moving the blue point from the "1" region to the "2" region in the latent space, we can see the interpolated reconstruction.

## 2.2 Low-Level Graph View

Once the user has a grasp of the overview, they can view the computations involved with the VAE by revealing the VAE computational graph as shown in Figure 2. This section connects the static documentation to the interactive pieces.

First, the Keras [4, 5] Python code is displayed and colored so the notation is easier to understand [6]. The code can be visualized as a computational graph as shown in Figure 4. We show the mean vector $\mu$ and the log of the variances vector $\log(\sigma^2)$ with the real numbers on the graph. The encoder doesn't directly output the standard deviation $\sigma$ since the standard deviation must be greater than 0. Here we show the Log-Var Trick [15] where we recover the $\sigma$ by applying

$$\begin{aligned}\sigma &= e^{\frac{1}{2}\log(\sigma^2)} \\ &= e^{\frac{1}{2}2\log(\sigma)} \\ &= \sigma\end{aligned}$$

which forces the standard deviation $\sigma$ to be positive [15]. The Log-Var trick is represented on the graph as mapping the encoding ($\log(\sigma^2)$) through the exponential function node to produce the output ($\sigma$) vector (see Figure 4).

The parameters $\mu$ and $\sigma$ specify the normal distribution $z \sim N(\mu, \sigma^2)$ we sample from. The Reparameterization Trick [11] samples $N(\mu, \sigma^2)$ by sampling a standard normal distribution

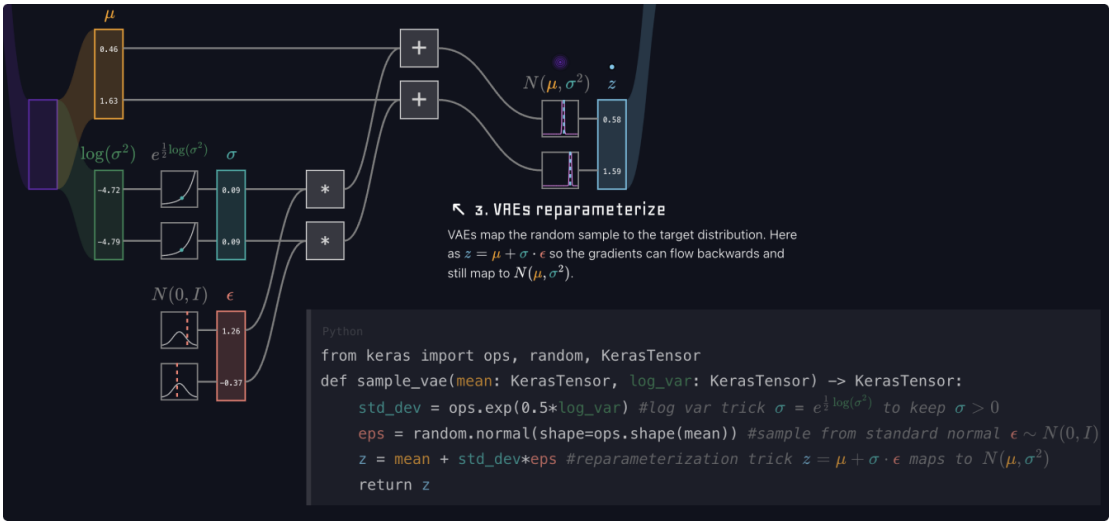


Figure 4: VAE sampling Keras code [4] accompanied by its computational graph. Extra labels have been removed for figure presentation.

labeled as $\epsilon \sim N(0, I)$ and mapped to $N(\mu, \sigma^2)$ by

$$z = \mu + \sigma \cdot \epsilon.$$

The computational graph highlights the Reparameterization Trick [11] by separating the $\epsilon$ from the main pathway. A user can see that no parameters depend on $\epsilon$ and that gradients can pass back to the encoder easily. In Figure 4, both probability distributions are shown as curves with vertical dotted lines to show values that are sampled. In Figure 1, on the latent space the $z$ is the blue point and the $N(\mu, \sigma^2)$ is the purple distribution.

To make it completely obvious which code corresponds to what part of the graph, there is a two-way interaction. When a user hovers over a line of code, the graph highlights the corresponding computation and vice versa (see Figure 2).

# 3 Implementation

To make VAE Explainer, we trained an existing implementation of a VAE directly copied from the Keras Variational Autoencoder Example [5] with some modifications for presentation. The training can be found in a `Colab Notebook`.

Just to summarize from [5], the model consists of a Convolutional Neural Network [13] as the encoder and the opposite as the decoder (Convolution Transposes). The model was trained with the Adam optimizer [10] over 30 epochs of the 60,000 MNIST Digits train set [19].

After training the model, we converted the Keras model to a TensorFlow graph and exported the graph to a TensorFlowJS format so it could be run in the browser [1, 4, 7]. We specifically exported the encoder and decoder as separate models so that the middle computation could be computed and visualized in the browser easily. Additionally, we computed the encodings for the first 10,000 MNIST Digit train set [19] images to better map out the latent space in the browser.

We used JavaScript, TensorflowJS [7], and Svelte [8] for the interactive frontend. The visualizations are primarily SVG and Canvas elements. The frontend code can be found at the open source repository `https://github.com/xnought/vae-explainer` and the live site can be found at `https://xnought.github.io/vae-explainer`.

# 4 Conclusion

VAE Explainer adds live interaction to static explanation. First a user can summarize what a VAE does, then they can view the real code and computational graph for how a VAE works.

To improve this work, more explanation on the VAE loss function would further help someone understand how the encoded normal distributions are regularized to standard normal. Additionally, extending to Vector Quantized Variational Autoencoders (VQ-VAE) would cover the latest and greatest for Autoencoders.


## Acknowledgments

We thank Adam Coscia for valuable feedback on early versions of the interactive website. Thank you Adam!


# References


[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Dave Bergmann and Cole Stryker. *What is a Variational Autoencoder?* https://www.ibm.com/think/topics/variational-autoencoder. 2024.

[3] Aeree Cho et al. "Transformer Explainer: Interactive Learning of Text-Generative Models". In: *arXiv preprint arXiv:2408.04619* (2024).

[4] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[5] François Chollet. *Variational AutoEncoder*. URL: https://keras.io/examples/generative/vae/.

[6] *Colorized Math Equations*. https://betterexplained.com/articles/colorized-math-equations/. 2017.

[7] Google. *TensorflowJS*. https://github.com/tensorflow/tfjs. 2018.

[8] Rich Harris et al. *Svelte*. https://github.com/sveltejs/svelte. 2016.

[9] Jackson Kang. *Pytorch VAE tutorial*. https://github.com/Jackson-Kang/Pytorch-VAE-tutorial. 2021.

[10] Diederik P Kingma. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[11] Diederik P Kingma. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[12] Diederik P. Kingma and Max Welling. *An Introduction to Variational Autoencoders*. 2019.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[14] Seongmin Lee et al. "Diffusion explainer: Visual explanation for text-to-image stable diffusion". In: *arXiv preprint arXiv:2305.03509* (2023).

[15] Sebastian Raschka. *The Log-Var Trick*. https://www.youtube.com/watch?v=pmvo0S3-G-I. 2021.

[16] Xander Steenbrugge. *Vartiational Autoencoders*. https://www.youtube.com/watch?v=9zKuYvjFFS8. 2018.

[17] Zijie J Wang et al. "CNN explainer: learning convolutional neural networks with interactive visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2020), pp. 1396–1406.

[18] Dagang Wei. *Demystifying Neural Networks: Variational AutoEncoders*. https://medium.com/@weidagang/demystifying-neural-networks-variational-autoencoders-6a44e75d0271. 2024.

[19] LeCun Yann. "The mnist database of handwritten digits". In: *R* (1998).