Supplementary Material for TreeDiffusion: Hierarchical Generative Clustering for Conditional Diffusion

Jorge da Silva Gonçalves (✉), Laura Manduchi, Moritz Vandenhirtz, and Julia E. Vogt

ETH Zurich, Rämistrasse 101, 8092 Zurich, Switzerland
jorge.dasilvagoncalves@inf.ethz.ch

1 Detailed Formulation of TreeVAE

Here, we provide a more in-depth introduction to TreeVAE, closely following the original paper [4]. Built upon the VAE framework [3], TreeVAEs inherently possess stochastic latent variables. However, unlike traditional VAEs, TreeVAEs organize these latent variables in a learnable binary tree structure as shown in Figure 1, enabling them to capture complex hierarchical relationships.

In the TreeVAE framework, the tree hierarchically divides the data, yielding separate stochastic embeddings for each node. This division into nodes induces a probability distribution, allowing each sample to navigate through the nodes of the tree in a probabilistic manner, from the root to the leaf. Ideally, high-variance features should be partitioned at earlier stages in the tree, while deeper nodes encapsulate more detailed concepts. The flexible tree structure is learned during training and is specific to the data distribution. This structure allows for the incorporation of sample-specific probability distributions over the different paths in the tree, as further explained in Section 1.1. In this manner, TreeVAEs contribute to the generation of comprehensive hierarchical data representations.

TreeVAE combines elements from both LadderVAE [8] and ANT [9], with the aim to create a VAE-based model capable of performing hierarchical clustering via the latent variables. Similar to LadderVAE, the inference and the generative model share the same top-down hierarchical structure. In fact, if we isolate a path from the root to any leaf in the tree structure of TreeVAE, we obtain an instance of the top-down model seen in LadderVAE. On the other side, both Adaptive Neural Trees and TreeVAE engage in representation and architecture learning during training. However, while ANTs are tailored for regression and classification tasks, TreeVAE's focus lies in hierarchical clustering and generative modeling. This allows TreeVAEs to generate new class-specific data samples based on the latent embeddings of the leaves from the tree.

1.1 Model Formulation

TreeVAE comprises an inference model and a generative model, which share the global structure of the binary tree $\mathcal{T}$. Following the original work [4], the tree structure is learned during training using dataset $\boldsymbol{X}$ and a predetermined maximum tree depth denoted as H. Specifically, the tree structure entails the set of nodes $\mathbb{V} = \{0, ..., V\}$, the subset of leaves $\mathbb{L} \subset \mathbb{V}$, and the set of edges $\mathcal{E}$.

While the global structure of the tree is the same for all samples in the data, the latent embeddings $\boldsymbol{z} = \{\boldsymbol{z}_0, ..., \boldsymbol{z}_V\}$ for all nodes and the so-called decisions $\boldsymbol{c} = \{\mathrm{c}_0, ..., \mathrm{c}_{V-|\mathbb{L}|}\}$ for all non-leaf nodes are unique to each sample, representing sample-specific random variables. The latent embeddings $\boldsymbol{z}$ are modeled as Gaussian random variables. Their distribution parameters are determined by functions that depend on the latent embeddings of the parent node. These functions, referred to as "transformations", are implemented as MLP. Moreover, each decision variable c_i corresponds to a Bernoulli random variable, where $\mathrm{c}_i = 0$ signifies the selection of the left child at internal node i. These decision variables influence the traversal path within the tree during both the generative and inference processes. The parameters governing the Bernoulli distributions are functions of the respective node value, parametrized by MLP termed "routers".

The transformations and routers are learned during the model training process, as further elaborated in Section 1.2, which comprehensively covers all aspects of model training. The exact parametrizations of the transformations and routers vary depending on whether they are applied in the context of the inference or generative model. Subsequently, the specifics of both of these models will be explored.

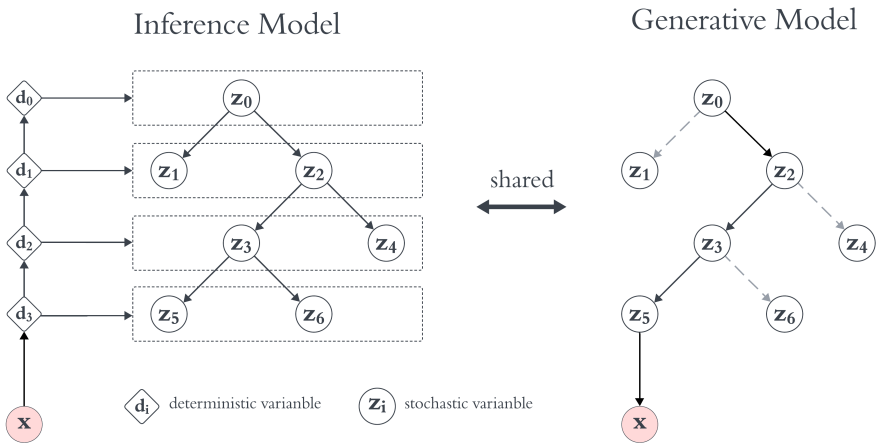

Fig. 1. Illustration of the TreeVAE model structure. The learned tree topology is shared between the inference and generative models.

Generative Model

The top-down generative model of TreeVAE, illustrated in Figure 1 on the right side, governs the generation of new samples. It starts by sampling a latent representation $\boldsymbol{z}_0$ from a standard Gaussian distribution for the root node, serving as the initial point for the generation process:

$$p_\theta\left(\boldsymbol{z}_0\right) = \mathcal{N}\left(\boldsymbol{z}_0 \mid \mathbf{0}, \boldsymbol{I}\right). \tag{1}$$

This latent embedding then traverses through the tree structure, leading to new samples being generated at each node based on their ancestor nodes. These remaining nodes in the tree are characterized by their own latent representations, which are sampled conditionally on the sample-specific embeddings of their parent nodes. This process can be mathematically expressed as follows:

$$p\left(\boldsymbol{z}_i \mid \boldsymbol{z}_{pa(i)}\right) = \mathcal{N}\left(\boldsymbol{z}_i \mid \mu_{p,i}(\boldsymbol{z}_{pa(i)}), \sigma^2_{p,i}(\boldsymbol{z}_{pa(i)})\right), \tag{2}$$

where $\{\mu_{p,i}, \sigma_{p,i} \mid i \in \mathbb{V}\backslash\{0\}\}$ correspond to the transformation neural networks associated with the generative model, as denoted by the subscript p.

Following the generation of latent representations for each node i in the tree, the decision regarding traversal to the left or right child node are determined based on the sampled $\boldsymbol{z}_i$ value. These decisions, denoted by c_i, for each non-leaf node i are guided by routers, which are neural network functions linked to the generative model and defined as $\{r_{p,i} \mid i \in \mathbb{V}\backslash\mathbb{L}\}$. Here, $\mathrm{c}_i = 0$ indicates the selection of the left child of the internal node i. As mentioned before, these decisions follow a Bernoulli distribution, $\mathrm{c}_i \mid \boldsymbol{z}_i \sim \mathrm{Ber}\left(r_{p,i}\left(\boldsymbol{z}_i\right)\right)$, which, in turn, induces the following path probability between a node i and its parent node $pa(i)$:

$$p\left(\mathrm{c}_{pa(i)\to i} \mid \boldsymbol{z}_{pa(i)}\right) = \mathrm{Ber}\left(\mathrm{c}_{pa(i)\to i} \mid r_{p,pa(i)}(\boldsymbol{z}_{pa(i)})\right). \tag{3}$$

Each decision made at a parent node determines the selection of the subsequent child node along the path. Consequently, the generative process progresses recursively until a leaf node l is reached. Importantly, each decision path $\mathcal{P}_l$ from the root to the leaf l is associated with a distinct set of decisions, which, in turn, determine the probabilities of traversing to different child nodes. These path probabilities, computed by the routers, play a critical role in shaping the distribution of the generated samples. The path $\mathcal{P}_l$ can be defined by the set of traversed nodes in the path. Moreover, let $\boldsymbol{z}_{\mathcal{P}_l} = \{\boldsymbol{z}_i \mid i \in \mathcal{P}_l\}$ denote the set of latent embeddings for each node in the path $\mathcal{P}_l$. Then, the prior probability of the latent embeddings and the path given the tree structure $\mathcal{T}$ corresponds to

$$p_\theta\left(\boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l\right) = p\left(\boldsymbol{z}_0\right) \prod_{i \in \mathcal{P}_l \backslash \{0\}} \underbrace{p\left(\mathrm{c}_{pa(i)\to i} \mid \boldsymbol{z}_{pa(i)}\right)}_{\text{decision probability}} \underbrace{p\left(\boldsymbol{z}_i \mid \boldsymbol{z}_{pa(i)}\right)}_{\text{sample probability}}. \tag{4}$$

To conclude the generative process, $\boldsymbol{x}$ is obtained based on the latent embeddings of a selected leaf l. Nevertheless, assumptions on the distribution of the inputs

are required. For real-valued $\boldsymbol{x}$, such as in colored datasets, [4] assume a Gaussian distribution. For grayscale images, they consider the Bernoulli distribution. Thus,

$$p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l\right) = \begin{cases} \mathcal{N}\Big(\boldsymbol{x} \mid \mu_{x,l}\left(\boldsymbol{z}_l\right), \sigma^2_{x,l}\left(\boldsymbol{z}_l\right)\Big) & \text{for colored datasets,} \\ \text{Ber}\left(\boldsymbol{x} \mid \mu_{x,l}\left(\boldsymbol{z}_l\right)\right) & \text{for grayscale datasets,} \end{cases} \tag{5}$$

where $\{\mu_{x,l}, \sigma_{x,l} \mid l \in \mathbb{L}\}$ or $\{\mu_{x,l} \mid l \in \mathbb{L}\}$, respectively, are implemented as leaf-specific neural networks called "decoders". Typically, a simplification is made by assuming $\sigma^2_{x,l}\left(\boldsymbol{z}_l\right) = \boldsymbol{I}$ for convenience.

Inference Model

The inference model of TreeVAE, depicted on the left side of Figure 1, introduces a deterministic bottom-up pass to incorporate the conditioning on $\boldsymbol{x}$, distinguishing it from the generative model. This bottom-up process is akin to the framework proposed by LadderVAE [8]. In this hierarchical structure, the bottom-up deterministic variables depend on each other via a series of neural network operations, specifically MLP of the same architecture as the transformation MLP defined previously,

$$\mathbf{d}_h = \text{MLP}\left(\mathbf{d}_{h+1}\right). \tag{6}$$

The first deterministic variable in this chain, denoted as $\mathbf{d}_H$, serves as the output of an encoder neural network. This encoder accepts the input image $\boldsymbol{x}$ and transforms it into the flattened, low-dimensional vector $\mathbf{d}_H$. H corresponds to both the number of deterministic variables involved in the bottom-up process and the maximum depth to which the tree structure can grow during training as further explained in Section 1.2.

The tree structure is shared between the inference and the generative model, though adjustments are made to the node-specific parameterizations of the Gaussian distributions. Notably, in the inference phase, information is introduced through conditioning on $\boldsymbol{x}$, influencing the distribution of latent embeddings across all nodes. Hence, similar to LadderVAE, the means $\boldsymbol{\mu}_{q,i}$ and variances $\boldsymbol{\sigma}^2_{q,i}$ of the variational posterior distribution for each node i are calculated using dense linear network layers conditioned on the deterministic variable of the same depth as the node i:

$$\hat{\boldsymbol{\mu}}_{q,i} = \text{Linear}\left(\mathbf{d}_{\text{depth}(i)}\right), \quad i \in \mathbb{V} \tag{7}$$

$$\hat{\boldsymbol{\sigma}}^2_{q,i} = \text{Softplus}\left(\text{Linear}\left(\mathbf{d}_{\text{depth}(i)}\right)\right), \quad i \in \mathbb{V} \tag{8}$$

$$\boldsymbol{\sigma}_{q,i} = \frac{1}{\hat{\boldsymbol{\sigma}}^{-2}_{q,i} + \boldsymbol{\sigma}^{-2}_{p,i}}, \quad \boldsymbol{\mu}_{q,i} = \frac{\hat{\boldsymbol{\mu}}_{q,i}\hat{\boldsymbol{\sigma}}^{-2}_{q,i} + \boldsymbol{\mu}_{p,i}\boldsymbol{\sigma}^{-2}_{p,i}}{\hat{\boldsymbol{\sigma}}^{-2}_{q,i} + \boldsymbol{\sigma}^{-2}_{p,i}}, \tag{9}$$

where the subscript q denotes the parameters specific to the inference model. Given these posterior parameters, the following equations determine the variational distributions of the latent embeddings for the root and the succeeding

nodes in the tree structure:

$$q\left(\boldsymbol{z}_0 \mid \boldsymbol{x}\right) = \mathcal{N}\left(\boldsymbol{z}_0 \mid \mu_{q,0}(\boldsymbol{x}), \sigma^2_{q,0}(\boldsymbol{x})\right), \tag{10}$$

$$q_\phi\left(\boldsymbol{z}_i \mid \boldsymbol{z}_{pa(i)}\right) = \mathcal{N}\left(\boldsymbol{z}_i \mid \mu_{q,i}\left(\boldsymbol{z}_{pa(i)}\right), \sigma^2_{q,i}\left(\boldsymbol{z}_{pa(i)}\right)\right), \quad \forall i \in \mathcal{P}_l. \tag{11}$$

The routers in the inference model maintain the same architecture as those in the generative model. Nevertheless, in the inference process, decisions are now conditioned on $\boldsymbol{x}$. This means that while the routers retain their original structure, the distribution of decision variables c_i at node i now depends on the deterministic variable of the corresponding depth, $\mathrm{c}_i \mid \boldsymbol{x} \sim \mathrm{Ber}(r_{q,i}(\mathbf{d}_{\mathrm{depth(i)}}))$, resulting in the following variational path probability between a node i and its parent node $pa(i)$:

$$q\left(\mathrm{c}_{pa(i)\to i} \mid \boldsymbol{x}\right) = q\left(\mathrm{c}_i \mid \mathbf{d}_{\mathrm{depth}(pa(i))}\right) = \mathrm{Ber}\left(\mathrm{c}_{pa(i)\to i} \mid r_{q,pa(i)}\left(\mathbf{d}_{\mathrm{depth}(pa(i))}\right)\right), \tag{12}$$

Finally, given the variational distributions of the latent embeddings equation 10 & equation 11 and the variational distributions of the decisions equation 12, the variational posterior distribution of the latent embeddings and paths corresponds to:

$$q\left(\boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l \mid \boldsymbol{x}\right) = q\left(\boldsymbol{z}_0 \mid \boldsymbol{x}\right) \prod_{i \in \mathcal{P}_l \setminus \{0\}} q\left(\mathrm{c}_{pa(i)\to i} \mid \boldsymbol{x}\right) q\left(\boldsymbol{z}_i \mid \boldsymbol{z}_{pa(i)}\right). \tag{13}$$

1.2 Model Training

TreeVAE [4] entails the training of various components, involving both the parameters of the neural network layers present in the inference and generative models, as detailed in Section 1.1, and the binary tree structure $\mathcal{T}$. The training process alternates between optimizing the model parameters with a fixed tree structure and expanding the tree.

During a training iteration given the current tree structure, the neural layer parameters are optimized. These include the parameters of the inference model and the generative model, encompassing the encoder $(\mu_{q,0}, \sigma_{q,0})$, the bottom-up MLPs from equation 6, the dense linear layers from equation 7 & equation 8, the transformations $(\{(\mu_{p,i}, \sigma_{p,i}), (\mu_{q,i}, \sigma_{q,i}) \mid i \in \mathbb{V}\setminus\{0\}\})$, the routers $(\{r_{p,i}, r_{q,i} \mid i \in \mathbb{V}\setminus\mathbb{L}\})$, and the decoders $(\{\mu_{x,l}, \sigma_{x,l} \mid l \in \mathbb{L}\})$. The objective in training these parameters is to maximize the likelihood of the data given the learned tree structure $\mathcal{T}$, denoted as $p(\boldsymbol{x} \mid \mathcal{T})$. This corresponds to modeling the distribution of real data via the hierarchical latent embeddings. To compute $p(\boldsymbol{x} \mid \mathcal{T})$, the latent embeddings must be marginalized out from the joint distribution:

$$p(\boldsymbol{x} \mid \mathcal{T}) = \sum_{l \in \mathbb{L}} \int_{\boldsymbol{z}_{\mathcal{P}_l}} p\left(\boldsymbol{x}, \boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l\right) = \sum_{l \in \mathbb{L}} \int_{\boldsymbol{z}_{\mathcal{P}_l}} p_\theta\left(\boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l\right) p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l\right). \tag{14}$$

Similar to other Variational Autoencoders [3], the optimization of TreeVAE ultimately comes down to maximizing the ELBO. As shown by [4], the ELBO in

this setting can be written as follows:

$$\mathcal{L}(\boldsymbol{x} \mid \mathcal{T}) := \underbrace{\mathbb{E}_{q(\boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l \mid \boldsymbol{x})}[\log p(\boldsymbol{x} \mid \boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l)]}_{\mathcal{L}_{rec}} - \underbrace{\mathrm{KL}(q(\boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l \mid \boldsymbol{x}) \| p(\boldsymbol{z}_{\mathcal{P}_l}, \mathcal{P}_l))}_{\mathrm{KL}_{root} + \mathrm{KL}_{nodes} + \mathrm{KL}_{decisions}}. \quad (15)$$

Hereby, we distinguish between the reconstruction term and the KL divergence term between the variational posterior and the prior of the tree, which can be further decomposed into contributions from the root, the remaining nodes, and decisions, as indicated in Equation equation 15. Both terms are approximated using Monte Carlo (MC) sampling during training, as elaborated further in [4].

To grow the binary tree structure $\mathcal{T}$, TreeVAE begins with a simple tree configuration, typically composed of a root and two leaves. This initial structure is trained for a defined number of epochs, optimizing the ELBO. Subsequently, the model iteratively expands the tree by attaching two new child nodes to a current leaf node in the model. In their approach, the authors [4] opted to expand the tree by selecting nodes with the highest number of assigned samples, thus implicitly encouraging balanced leaves. The sub-tree formed by the new leaves and the parent node undergoes training for another number of epochs, keeping the weights of the remaining model frozen. This expansion process continues until either the tree reaches its maximum depth H, a predefined maximum number of effective leaves, or another predefined condition is met. Optionally, after the tree has been expanded, all parameters in the model may be fine-tuned for another predefined number of epochs. Finally, the tree is pruned to remove empty branches.

Contrastive learning is used to improve the clustering performance, especially for colored images. This addition enables the model to encode prior knowledge on data invariances through augmentations, facilitating the learning process and better capturing meaningful relationships within complex data. By incorporating contrastive objectives into the training process, TreeVAE becomes more adept at retrieving semantically meaningful clusters from colored image data.

2 Detailed Formulation of Diffusion Models

Diffusion models have garnered considerable attention in recent years for their remarkable image-generation capabilities, outperforming traditional GANs in achieving high-quality results. This surge in interest has led to the development of numerous diffusion-based models, with the initial concept being introduced by [7], drawing inspiration from principles rooted in thermodynamics. In this section, we delve into one of the first and most well-known diffusion models, namely the DDPM, outlined in Section 2.1. Additionally, we explore one possible integration of DDPM with VAEs to leverage the interpretable latent space offered by VAEs along with the superior generation quality of DDPM. Note that we use the DDIM instantiation at inference time.

2.1 Diffusion Denoising Probabilistic Models

Diffusion Denoising Probabilistic Models (DDPM) [2] are latent variable models that consist of two opposed processes. The main idea behind DDPM involves iteratively adding noise to the original image, thereby progressively degrading the signal with each step until only noise remains. In a second step, the image is successively reconstructed through a denoising process, employing a learned function to remove the noise. Therefore, DDPM essentially entail a forward noising process followed by a reverse denoising process, aiming to restore the original image from its noisy counterpart. Subsequently, both these processes will be explored in more detail.

Forward Process

The forward process, also known as the diffusion process or forward noising process, serves a similar purpose as the inference model in VAEs. It operates as a Markov chain that gradually introduces noise to the data signal $\boldsymbol{x}_0$ over T steps, where the intermediate states of the deformed data are denoted as $\boldsymbol{x}_t$ for $t = 1, \ldots, T$. This process follows a trajectory determined by a noise schedule $\beta_1, \ldots, \beta_T$, which controls the rate at which the original data is degraded. While it is possible to learn the variances β_t via reparametrization, they are often chosen as hyperparameters with a predetermined schedule [2]. Assuming Gaussian noise, the forward process can be represented as follows:

$$q\left(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0\right) = \prod_{t=1}^{T} q\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}\right) \tag{16}$$

$$q\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}\right) = \mathcal{N}\left(\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}\right) \tag{17}$$

Importantly, $\boldsymbol{x}_t$ for any step t can be sampled directly given $\boldsymbol{x}_0$ using:

$$q\left(\boldsymbol{x}_t \mid \boldsymbol{x}_0\right) = \mathcal{N}\left(\sqrt{\overline{\alpha}_t}\boldsymbol{x}_0, \left(1-\overline{\alpha}_t\right)\boldsymbol{I}\right), \tag{18}$$

where $\alpha_t = (1 - \beta_t)$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$. This property significantly boosts training efficiency by eliminating the need to sample every state between $\boldsymbol{x}_0$ and $\boldsymbol{x}_t$. Additionally, when conditioned on $\boldsymbol{x}_0$, the posteriors of the forward process are tractable and can be determined in closed form:

$$q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) = \mathcal{N}\left(\boldsymbol{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\boldsymbol{x}_t, \boldsymbol{x}_0\right), \tilde{\beta}_t \boldsymbol{I}\right), \tag{19}$$

$$\text{where} \quad \tilde{\boldsymbol{\mu}}_t\left(\boldsymbol{x}_t, \boldsymbol{x}_0\right) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\boldsymbol{x}_t \quad \text{and} \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{20}$$

Reverse Process

The reverse process, also known as the reverse denoising process or backward process, is comparable to the generative model in Variational Autoencoders.

Beginning with pure Gaussian noise, the reverse process governs the sample generation by progressively eliminating noise through a sequence of T denoising steps. These steps are facilitated by time-dependent, learnable Gaussian transitions that adhere to the Markov property, meaning each transition relies solely on the state of the previous time step in the Markov chain. This denoising sequence reverses the noise addition steps of the forward process, enabling the gradual recovery of the original data.

In summary, the reverse process models a complex target data distribution by sequentially transforming simple distributions through a generative Markov chain. This approach overcomes the traditional tradeoff between tractability and flexibility in probabilistic models [7], leading to a flexible and efficient generative model:

$$p\left(\boldsymbol{x}_{0:T}\right) = p\left(\boldsymbol{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t\right) \tag{21}$$

$$p_\theta\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t\right) = \mathcal{N}\left(\boldsymbol{\mu}_\theta\left(\boldsymbol{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\boldsymbol{x}_t, t\right)\right) \tag{22}$$

[2] keep the time dependent variances $\boldsymbol{\Sigma}_\theta\left(\boldsymbol{x}_t, t\right) = \sigma_t^2 \boldsymbol{I}$ constant, typically choosing $\sigma_t^2 = \beta_t$ or $\sigma_t^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. Therefore, only the time-dependent posterior mean function $\boldsymbol{\mu}_\theta$ is learned during training. In fact, [2] further suggest the following reparametrization of the posterior means:

$$\begin{aligned}
\boldsymbol{\mu}_\theta\left(\boldsymbol{x}_t, t\right) &= \tilde{\boldsymbol{\mu}}_t\left(\boldsymbol{x}_t, \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\boldsymbol{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}_t\right)\right)\right) \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}_t, t\right)\right),
\end{aligned}$$

where $\boldsymbol{\epsilon}_\theta$ is a learnable function that predicts the noise at any given time step t.

Model Training

Training a diffusion model corresponds to optimizing the reverse Markov transitions to maximize the likelihood of the data. This is achieved by maximizing the ELBO or, equivalently, minimizing the variational upper bound on the negative log-likelihood:

$$\begin{aligned}
\mathcal{L} = \mathbb{E}_q[&\underbrace{\mathrm{KL}\left(q\left(\boldsymbol{x}_T \mid \boldsymbol{x}_0\right) \| p\left(\boldsymbol{x}_T\right)\right)}_{\mathcal{L}_T} \\
&+ \sum_{t>1} \underbrace{\mathrm{KL}\left(q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) \| p_\theta\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t\right)\right)}_{\mathcal{L}_{t-1}} \\
&- \underbrace{\log p_\theta\left(\boldsymbol{x}_0 \mid \boldsymbol{x}_1\right)}_{\mathcal{L}_0}]
\end{aligned}$$

For small diffusion rates β_t, the forward and the reverse processes share the same functional form, following Gaussian distributions [7]. Thus, the KL divergence terms between the posteriors of the forward process equation 19 and the

reverse process equation 21 have a closed form. The training process can be further simplified [2] by introducing a more efficient training objective. Given the assumption of fixed variances, $\mathcal{L}_{t-1}$ can be written as:

$$\mathcal{L}_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\boldsymbol{\mu}}_t \left(\boldsymbol{x}_t, \boldsymbol{x}_0 \right) - \boldsymbol{\mu}_\theta \left(\boldsymbol{x}_t, t \right) \right\|^2 \right] + C, \tag{23}$$

where C is a constant that does not depend on the model parameters θ. Thus, the reverse process mean function, $\boldsymbol{\mu}_\theta$, is optimized to predict $\tilde{\boldsymbol{\mu}}_t$, the fixed noisy mean function at time step t from the forward process equation 20. However, instead of directly comparing $\boldsymbol{\mu}_\theta$ and $\tilde{\boldsymbol{\mu}}_t$, by reparametrization, the model can be trained to predict the noise $\boldsymbol{\epsilon}$ at any given time step t which results in more stable training results according to [2]. Thus, equation 23 can be further rewritten as

$$\mathcal{L}_{t-1} - C = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t \left(1 - \bar{\alpha}_t \right)} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta \left(\sqrt{\bar{\alpha}_t} \boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t \right) \right\|^2 \right], \tag{24}$$

where $\boldsymbol{\epsilon}_\theta$ is a function parametrized as a neural network that maintains equal dimensionality for input and output. The preferred architecture for $\boldsymbol{\epsilon}_\theta$ is a U-Net [6], as used by [2,1,5]. The training of $\boldsymbol{\epsilon}_\theta$ involves multiple epochs, where for each sample $\boldsymbol{x}_0$ of the original data, a time step t within the diffusion sequence $1, \ldots, T$ is chosen at random. Subsequently, some noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is sampled for that specific time step, which needs to be predicted by $\boldsymbol{\epsilon}_\theta$. To optimize the model, a gradient step is computed with respect to the loss function detailed in Equation equation 24. Therefore, this training procedure does not require iterating through every step of the diffusion model, circumventing the issue of slow sample generation that is typical for diffusion models.

3 Additional Qualitative Results

3.1 Generations on CIFAR-10

Figure 2 presents an additional plot similar to the one for FashionMNIST from the main text. This plot illustrates the generated images of the TreeDiffusion model when trained on the CIFAR-10 dataset. In each of these plots, we display randomly generated images for each cluster. Below each set of leaf-specific images, we provide a normalized histogram showing the distribution of predicted classes by an independent ResNet-50 classifier that has been pre-trained on the training data of the dataset. This visualization helps to understand how well the model can generate distinct and meaningful clusters in the context of different datasets.

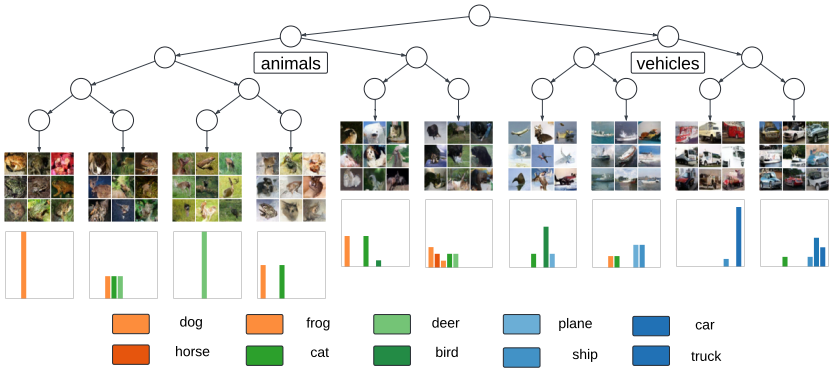

Fig. 2. TreeDiffusion model trained on CIFAR-10. For each cluster, random newly generated images are displayed. Below each set of images, a normalized histogram (ranging from 0 to 1) shows the distribution of predicted classes from an independent, pre-trained classifier on MNIST for all newly generated images in each leaf with a significant probability of reaching that leaf.

3.2 Additional Generation Examples for TreeVAE vs. TreeDiffusion

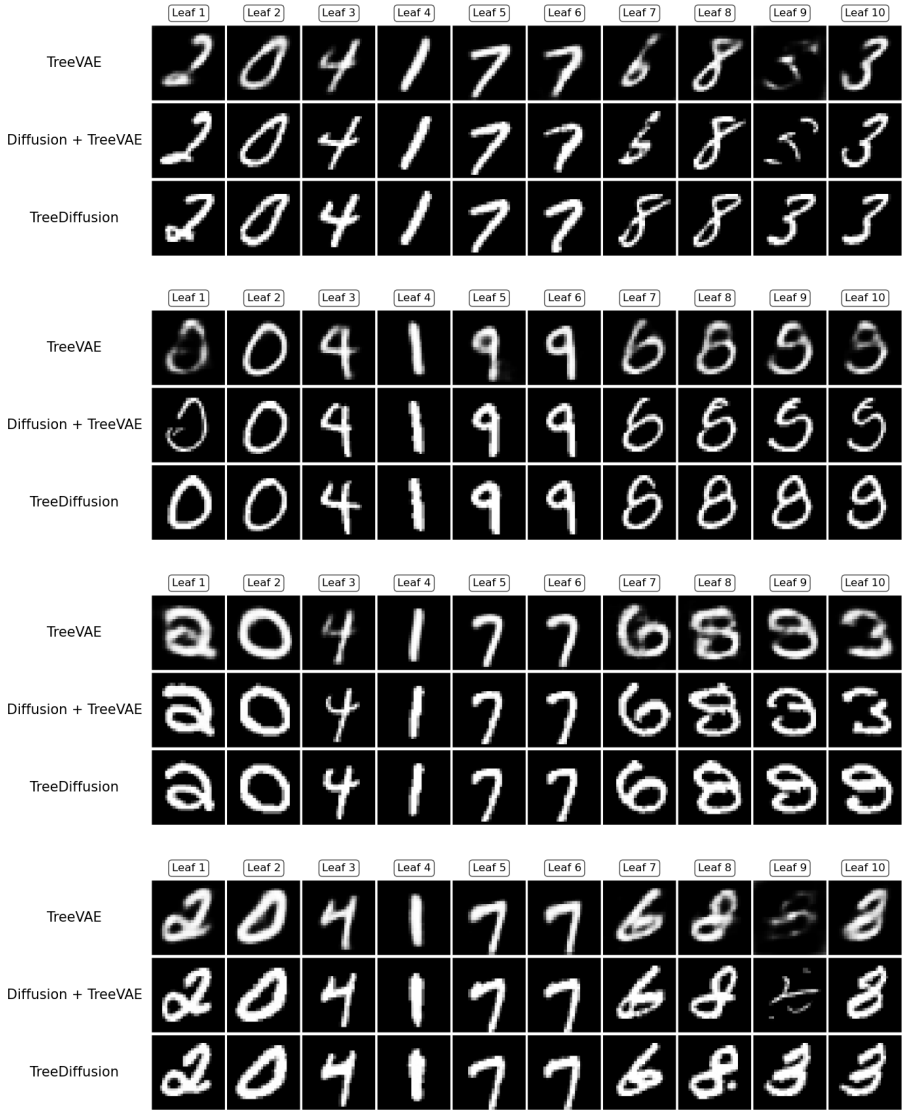

Fig. 3. For each example, we show image generations from every leaf of TreeVAE, TreeVAE + Diffusion, and TreeDiffusion model, all trained on MNIST. Each row shows the generated images from all leaves of the respective model, starting with the same root sample.

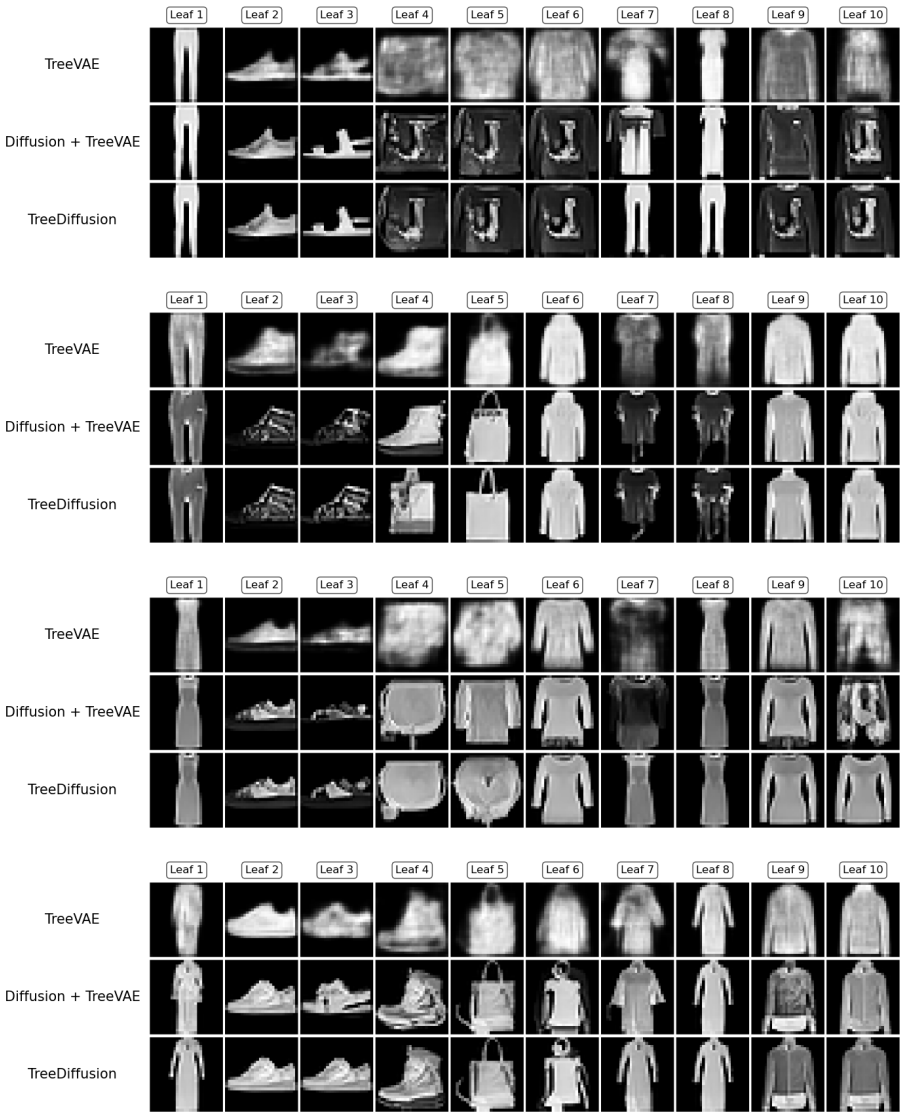

Fig. 4. For each example, we show image generations from every leaf of TreeVAE, TreeVAE + Diffusion, and TreeDiffusion model, all trained on FashionMNIST. Each row shows the generated images from all leaves of the respective model, starting with the same root sample.

Fig. 5. For each example, we show image generations from every leaf of TreeVAE, TreeVAE + Diffusion, and TreeDiffusion model, all trained on CIFAR-10. Each row shows the generated images from all leaves of the respective model, starting with the same root sample.

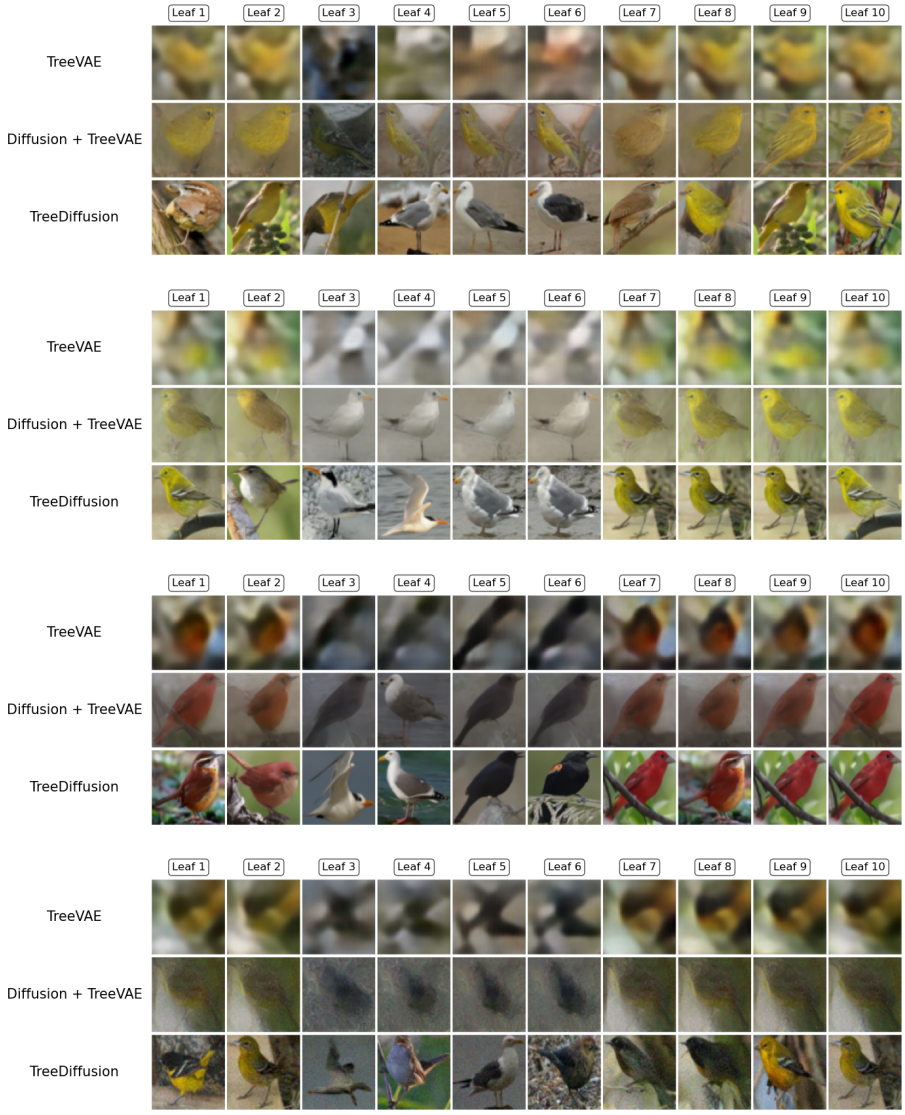

Fig. 6. For each example, we show image generations from every leaf of TreeVAE, TreeVAE + Diffusion, and TreeDiffusion model, all trained on CUBICC. Each row shows the generated images from all leaves of the respective model, starting with the same root sample.

4 Implementation Details

TreeVAE Training

Table 1. Overview of training configurations for TreeVAE, including model parameters, training hyperparameters, and contrastive learning specifics across datasets.

Data resolution	28x28x1	32x32x3	64x64x3
Encoder/Decoder Types	cnn1	cnn1	cnn2
Max. tree depth	7	7	7
Max. clusters	10	10	10
Representation dimensions	4	4	4
Latent channels	16	64	64
Bottom-up channels	32	128	128
Grow	True	True	True
Prune	True	True	True
Activation of last layer	sigmoid	mse	mse
Optimizer	Adam(lr=1e-3)	Adam(lr=1e-3)	Adam(lr=1e-3)
Effective batch size	256	256	128
Initial epochs	150	150	150
Smalltree epochs	150	150	150
Intermediate epochs	80	0	0
Fine-tuning epochs	200	0	0
lr decay rate	0.1	0.1	0.1
lr decay step size	100	100	100
Weight decay	1e-5	1e-5	1e-5
Contrastive augmentations	False	True	True
Augmentation method	None	InfoNCE	InfoNCE
Augmentation weight	None	100	100

We use a modified variant of the TreeVAE model, employing CNNs for its operations. The representations within TreeVAE are maintained in a 3D format, where the first dimension signifies the number of channels, while the subsequent two dimensions denote the spatial dimensions of the representations. Consequently, the deterministic variables in the bottom-up pathway possess a dimensionality of (Nb. of bottom-up channels, representation dimension, representation dimension), while the stochastic variables in the top-down tree structure have a dimensionality of (Nb. of latent channels, representation dimension, representation dimension). Table 1 provides details on the remaining parameters utilized for the TreeVAE model and its training. For more information, please refer to the code.

TreeDiffusion Training

Table 2 provides details on the remaining parameters utilized for the diffusion model and its training. For more information, please refer to the code.

Table 2. Overview of training configurations for the DDPM of the TreeDiffusion, including model parameters and training hyperparameters.

Data resolution	28x28x1	32x32x3	64x64x3
Noise Schedule	Linear(1e-4, 0.02)	Linear(1e-4, 0.02)	Linear(1e-4, 0.02)
U-Net channels	64	128	128
Scale(s) of attention block	[16]	[16,8]	[16,8]
Res. blocks per scale	2	2	2
Channel multipliers	(1,2,2,2)	(1,2,2,2)	(1,2,2,2,4)
Dropout	0.3	0.3	0.3
Diffusion loss type	L2	L2	L2
Optimizer	Adam(lr=2e-4)	Adam(lr=2e-4)	Adam(lr=2e-4)
Effective batch size	256	256	32
lr annealing steps	5000	5000	5000
Grad. clip threshold	1.0	1.0	1.0
EMA decay rate	0.9999	0.9999	0.9999

References

1. Dhariwal, P., Nichol, A.: Diffusion Models Beat GANs on Image Synthesis. In: Advances in Neural Information Processing Systems. vol. 34, pp. 8780–8794. Curran Associates, Inc. (2021)
2. Ho, J., Jain, A., Abbeel, P.: Denoising Diffusion Probabilistic Models. In: Advances in Neural Information Processing Systems. vol. 33, pp. 6840–6851. Curran Associates, Inc. (2020)
3. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014)
4. Manduchi, L., Vandenhirtz, M., Ryser, A., Vogt, J.: Tree Variational Autoencoders. In: Advances in Neural Information Processing Systems. vol. 36 (2023)
5. Pandey, K., Mukherjee, A., Rai, P., Kumar, A.: DiffuseVAE: Efficient, Controllable and High-Fidelity Generation from Low-Dimensional Latents. Transactions on Machine Learning Research (2022)
6. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. pp. 234–241. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015)
7. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: International conference on machine learning. pp. 2256–2265. PMLR (2015)
8. Sønderby, C.K., Raiko, T., Maaløe, L., Sønderby, S.K., Winther, O.: Ladder Variational Autoencoders. In: Advances in Neural Information Processing Systems. vol. 29. Curran Associates, Inc. (2016)
9. Tanno, R., Arulkumaran, K., Alexander, D., Criminisi, A., Nori, A.: Adaptive Neural Trees. In: Proceedings of the 36th International Conference on Machine Learning. pp. 6166–6175. PMLR (2019)