

HIERARCHICAL QUANTIZED DIFFUSION BASED TREE GENERATION METHOD FOR HIERARCHICAL REPRESENTATION AND LINEAGE ANALYSIS

Anonymous authors

Paper under double-blind review

ABSTRACT

In single-cell research, tracing and analyzing high-throughput single-cell differentiation trajectories is crucial for understanding complex biological processes. Key to this is the modeling and generation of hierarchical data that represents the intrinsic structure within datasets. Traditional methods face limitations in terms of computational cost, performance, generative capacity, and stability. Recent VAE-based approaches have made strides in addressing these challenges but still require **branch-specific** network modules for each tree branch, limiting their stability and **scalability to deep hierarchies, while also suffering from posterior collapse**. To overcome these challenges, we introduce **HDTree**, a diffusion-based approach **that captures** tree relationships within a hierarchical latent space using a unified hierarchical codebook and quantized diffusion processes to model tree node transitions. This method improves stability by eliminating branch-specific modules and **enhances** generative capacity through gradual hierarchical changes simulated by the diffusion process. HDTree’s effectiveness is demonstrated through comparisons on both general-purpose and single-cell datasets, where it outperforms existing methods in **reconstruction quality, generation diversity, and hierarchical consistency**. These contributions provide a new tool for hierarchical lineage analysis, enabling more accurate and efficient modeling of cellular differentiation paths and offering insights for downstream biological tasks. (code: https://anonymous.4open.science/r/code_HDTree_review-A8DB).

1 INTRODUCTION

In single-cell research, tracing and analyzing cellular differentiation trajectories is essential for understanding dynamic biological processes. This task requires not only effective modeling of hierarchical structures (Zeng et al., 2022), but also the ability to generate data that faithfully captures such hierarchies (Guo et al., 2024). Accurately characterizing the hierarchical organization underlying cell differentiation facilitates the exploration of cellular systems and fate decisions, while conditional generation based on these hierarchies enables interpretable discovery of biological mechanisms. Importantly, hierarchical structures are not exclusive to biology—they also emerge in various domains such as recommendation systems, molecular design, and knowledge representation (Chehreghani & Chehreghani, 2024; Gyurek et al., 2024; Tian et al., 2024). Therefore, developing models that can both represent and generate data along hierarchical relationships not only enhances performance in downstream tasks such as classification and clustering, but also provides deeper insight into the intrinsic organization of complex data.

As shown in Fig. 1, Traditional methods (Murtagh & Legendre, 2014; Rokhlin & Tygert, 2017) often rely on a combination of dimension reduction (Jia et al., 2022), clustering (Oti & Olusola, 2024), and data regression (Ali & Younas, 2021) techniques to achieve hierarchical modeling and data generation. While these approaches can address the tasks to some extent, they face challenges regarding computation costs, performance, generative capacity, and stability (Zang et al., 2024b). These limitations make them inadequate for handling the demands of large-scale, high-dimensional biological data. Recent state-of-the-art (SOTA) methods based on Variational Autoencoders (VAEs) (Manduchi et al., 2023; Xiao & Su, 2024; Majima et al., 2024) have unified generative tasks and hierarchical representation within a single modeling framework, achieving notable advancements. These VAE-based

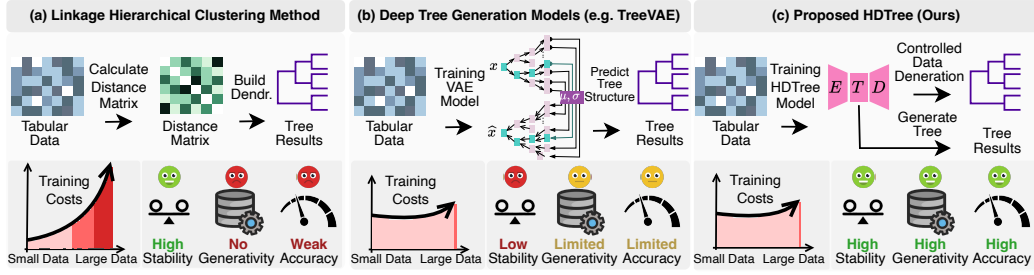


Figure 1: **The motivations.** Base method and deep method cannot meet the requirements of hierarchical representation and lineage analysis in terms of stability, generativity, accuracy, and training cost.

approaches effectively reduce computational costs when processing large-scale data while improving performance and data generation capabilities. However, a key limitation of existing SOTA methods lies in [their reliance on](#) specialized network modules for each tree branch (Manduchi et al., 2023). This design not only reduces [stability](#) but also constrains [the](#) ability to capture sufficiently deep and complex hierarchical [relationships](#). Specifically, [deep branches with sparse samples cannot effectively leverage representation knowledge learned from other branches, leading to limited generalization and difficulty in preserving global structure](#) (Ghahramani et al., 2010; Lakshminarayanan et al., 2016). Moreover, independent encoder-decoder pairs at each branch node are prone to overfitting due to [noise accumulation when training data is limited, thereby limiting](#) applicability in scenarios requiring robust and deep hierarchical modeling.

To address these challenges, we propose a novel deep learning-based method, **Hierarchical vector quantized Diffusion Model (HDTree)**, which captures tree relationships in a hierarchical latent space [through a unified hierarchical codebook](#) (Huang et al., 2024) and [models branch transitions via](#) a quantized diffusion process (Gu et al., 2022). The core innovation of HDTree lies in its integration of hierarchical latent space encoding with a quantized diffusion process, [systematically addressing the aforementioned limitations](#). First, [enhanced generalization is achieved by employing a unified latent space where all branches share the same codebook vectors, enabling even sparse deep nodes to leverage representation knowledge from other branches while preserving global hierarchical structure](#). Second, [improved stability is ensured by replacing branch-specific encoder-decoder pairs with a unified encoder and hierarchical codebook architecture, eliminating noise accumulation and overfitting risks associated with independent modules, and conditional diffusion steps, eliminating fragmented architecture risks while maintaining adaptability to complex tree topologies](#). Third, [strengthened generative capacity is realized by modeling branch transitions via a diffusion process](#) (Liu et al., 2024), [simulating gradual hierarchical changes to produce diverse and biologically plausible outputs](#). Finally, [performance gains](#) arise from soft contrastive learning and multi-scale latent space regularization, which sharpen the representation of hierarchical dependencies and improve lineage analysis accuracy. [The coordinated work of the above modules improves the performance of the entire model](#).

These advancements collectively enable HDTree to capture deep hierarchical relationships robustly, while the learned tree-structured embeddings can be directly applied to downstream tasks—such as lineage analyses via computationally efficient graph-based algorithms. *The contributions of this work are as:*

- We propose, HDTree, a novel hierarchical (tree) embeddings & data generation method that captures complex hierarchical relationships and generates high-quality data.
- We apply HDTree to the task of lineage analyses. It analyzes the cell differentiation path through a pathfinding algorithm based on the generated tree structure.
- Comparisons and visualizations of clustering performance, tree performance, generative performance, and lineage analyses performance on general-purpose datasets and single-cell datasets show that HDTree surpasses existing methods in terms of accuracy and performance, providing new tools for hierarchical lineage analysis.

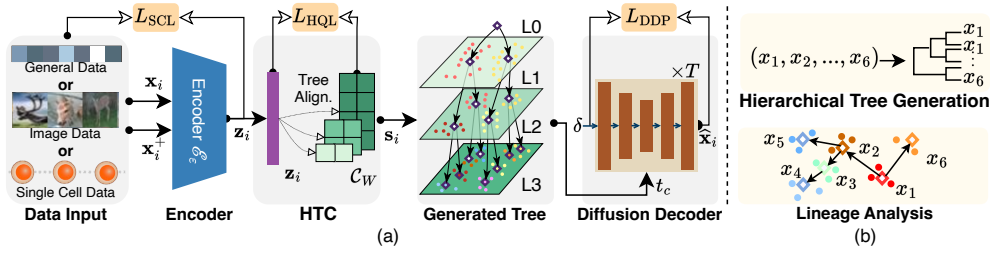


Figure 2: **Overview of the HDTree framework & tasks.** (a) The framework of HDTree, which consists of three main components: encoder for semantic representation, Hierarchical Tree Codebook (HTC) for tree-based structural modeling, and diffusion-based decoder for data generation. We use tree structures to model hierarchical relationships and generate data based on the hierarchical latent space. The soft contrastive loss (SCL), hierarchical quantization loss (HQL), and diffusion loss (DDP) are used to optimize the model. (b) The hierarchical tree generation and lineage analysis.

2 RELATED WORK

2.1 Tree-Structured Representation & Generative Models

Tree-structured representations are crucial for modeling hierarchical relationships in data (Zang et al., 2024b). Traditional methods like hierarchical clustering (Müllner, 2011) and distance-based techniques (Bouguettaya et al., 2015) rely on predefined metrics for tree construction. *Mondrian forests* (Lakshminarayanan et al., 2016) extend this paradigm by using hierarchical Gaussian priors over leaf node parameters, enabling efficient uncertainty quantification in large-scale regression tasks. Adams et al. (Ghahramani et al., 2010) propose tree-structured stick-breaking processes that provide flexible nonparametric priors over hierarchies with unbounded width and depth, allowing data to reside at internal nodes while maintaining infinite exchangeability. Recent deep learning advancements, such as TreeVAE (Manduchi et al., 2024), leverage recursive and hierarchical latent structures. Hyperbolic geometry methods, including HGNNs (Zhou et al., 2023), offer efficient hierarchical representations. TreeVI (Xiao & Su, 2024) enhances variational inference by utilizing tree structures for scalable training and improved performance in tasks like clustering and link prediction.

2.2 Deep Learning Based Cell Lineage Analysis

Cell lineage analysis is crucial for reconstructing developmental trajectories in single-cell genomics. Traditional methods like *Monocle* (Trapnell et al., 2014) and *Slingshot* (Street et al., 2018) infer pseudotime trajectories but are limited by predefined metrics and difficulty modeling unobserved progenitor states. Recent approaches such as *LineageVAE* (Majima et al., 2024) and *Waddington-OT* (Schiebinger et al., 2021) overcome some limitations with probabilistic models and optimal transport, though high dimensionality and sparsity remain challenges.

3 METHODS

3.1 Notation and Task Definition

Let $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^D\}_{i=1}^N$ denote a dataset with N samples, where each $\mathbf{x}_i$ is a D -dimensional feature vector. To enhance the generalization capability and robustness of the model, an augmented view $\mathbf{x}_i^+$ is generated for each $\mathbf{x}_i$ using kNN-based augmentation (Zang et al., 2024a), encouraging the encoder to preserve local semantic neighborhoods during contrastive learning. HDTree learns a hierarchical tree-structured latent representation $\mathcal{T}$ to capture multi-scale semantic relationships among data points. Formally, $\mathcal{T}$ is parameterized as a rooted binary tree of maximum depth L , where each node at depth l is associated with a learnable code vector $\mathbf{w}_j^l \in \mathbb{R}^d$. Both the node embeddings and the tree topology are jointly optimized during training, allowing the model to automatically discover a data-driven hierarchical organization.

Task 1 (Lineage Analysis). Given the learned hierarchical tree $\mathcal{T}$, the lineage analysis task aims to infer developmental trajectories by identifying paths that connect a specified origin node (e.g., a progenitor or stem cell state) to one or more destination nodes (e.g., differentiated cell types). The resulting paths represent discrete approximations of cell state transitions and reveal the hierarchical progression of differentiation.

Task 2 (Lineage-Conditioned Data Generation). Beyond trajectory inference, HDTree can synthesize new data points conditioned on specific lineage paths. Given a trajectory $\text{Path}_{\text{development}}$ in $\mathcal{T}$, the diffusion decoder $\mathcal{D}_\theta$ generates samples that are consistent with the hierarchical codes along the path, providing a controllable way to simulate intermediate or hypothetical cell states.

3.2 Model Design

HDTree addresses three key challenges in deep hierarchical modeling: (1) scalable representation of deep trees with exponentially growing branches, (2) capturing multi-granularity relationships across hierarchical levels, and (3) generating diverse samples along specific biological paths. These requirements motivate our design of three synergistic modules: HDTree is achieved through three key modules: encoder $\mathcal{E}_{\theta_E}$, hierarchical VQ codebook $\mathcal{C}_W$, and diffusion decoder $\mathcal{D}_\theta$.

3.2.1 Encoder $\mathcal{E}_{\theta_E}$ & Hierarchical Tree Codebook (HTC) $\mathcal{C}_W$.

Unlike prior methods with exponentially scaling parameters (Manduchi et al., 2023), HTC achieves linear complexity $O(d \cdot K)$ while explicitly encoding parent-child relationships through a unified codebook. The encoder $\mathcal{E}_{\theta_E}$ maps input data $\mathbf{x}_i$ into a latent space $\mathbf{z}_i \in \mathbb{R}^d$, where θ_E represents the learnable parameters of the encoder and d is the latent dimensions. The encoding process can be expressed as, $\mathbf{z}_i = \mathcal{E}_{\theta_E}(\mathbf{x}_i)$, $\mathbf{z}_i^+ = \mathcal{E}_{\theta_E}(\mathbf{x}_i^+)$.

To capture the hierarchical relationships in the data, HTC $\mathcal{C}_W$ is introduced, which is constructed as a binary tree, where each node represents a code vector in the latent space,

$$\mathcal{T}_j^l = \mathbf{w}_j^l \text{ if } l = L, \quad (\mathbf{w}_j^l, \mathcal{T}_{2j}^{(l+1)}, \mathcal{T}_{2j+1}^{(l+1)}) \text{ if } l < L. \quad (1)$$

where $\mathbf{w}_j^l$ is the learnable code vector at depth l and index j , $\mathcal{T}_0^0$ is the root node of the tree, L is the maximum depth of the tree. The $\mathbf{w}_j^l \in \mathbb{R}^d$ is the node embeddings at level l , for level l , we have w^l nodes. The tree structure is optimized during training to capture the semantic and structural relationships in the data. For input data $\mathbf{x}_i$, the HTC is used to quantize the latent representation $\mathbf{z}_i$ into a hierarchical sequence of code vectors $\mathbf{s}_i$,

$$\mathbf{s}_i = [\Omega^{w^1}(\mathbf{z}_i), \dots, \Omega^{w^L}(\mathbf{z}_i)], \Omega^{w^l}(\mathbf{z}_i) = \underset{\mathbf{w}_j^l \in \text{Children}(\mathbf{w}_j^{l-1})}{\text{argmin}} \|\mathbf{z}_i - \mathbf{w}_j^l\|_2, \quad (2)$$

where $\Omega^{(l)}(\mathbf{z}_i)$ selects the nearest code vector at depth l . In the rare case of a tie (i.e., multiple $\mathbf{w}_j^l$ with identical distance), we break ties deterministically by choosing the codeword with the smallest index j , ensuring that $\mathbf{s}_i$ is uniquely defined. The $\text{Children}(\mathbf{w}_j^{l-1})$ denotes the children of the code vector $\mathbf{w}_j^{l-1}$ at level $l-1$ which defined in Eq. (1). This design ensures sibling nodes (e.g., $\mathbf{w}_{2j}^{l+1}$, $\mathbf{w}_{2j+1}^{l+1}$) naturally inherit and refine their parent code $\mathbf{w}_j^l$, enabling knowledge sharing across branches while maintaining hierarchical specialization.

3.2.2 Diffusion Decoder $\mathcal{D}_{\theta_D}$

Unlike VAE decoders that suffer from posterior collapse and cannot enforce hierarchical constraints, or standard diffusion models that treat path labels as unstructured categories, the diffusion decoder $\mathcal{D}_{\theta_D}$ in HDTree explicitly aligns the generation process to the hierarchical codebook through quantized conditioning. It reconstructs data or generates new samples based on the hierarchical latent representations, guaranteeing valid path traversal across tree levels. It leverages a Denoising Diffusion Probabilistic Model (DDPM) (Ho et al., 2020b) to iteratively generate data starting from a noise distribution. Specifically, beginning with Gaussian noise $\mathbf{x}_T \sim \Omega(0, \mathbf{I})$, the model refines $\mathbf{x}_T$ through T diffusion steps to produce the final data $\mathbf{x}_0$ conditioned on the quantized code sequence $\mathbf{s}_i$ obtained from VQ (Eq. 5). The generation process $\tilde{\mathbf{x}}_i = \text{Gen}(\delta, \mathbf{s}_i | \mathcal{D}_{\theta_D}(\cdot))$ is formulated as,

$$\text{Gen}(\delta, \mathbf{s}_i | \phi^*) = \left\{ \tilde{\mathbf{x}}^0 | \tilde{\mathbf{x}}^{t-1} = \frac{1}{\sqrt{\alpha_t}} (\tilde{\mathbf{x}}^t - \tilde{\alpha}) + \sigma_t \Omega(0, 1) \right\}, \tilde{\alpha} = \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \mathcal{D}_{\theta_D}(\tilde{\mathbf{x}}^t, t, \mathbf{s}_i), \quad (3)$$

where $t \in \{T, \dots, 1\}$, $\mathbf{s}_i = \{c_{\mathbf{z}_i}^1, \dots, c_{\mathbf{z}_i}^L\}$ is the hierarchical code sequence from root to leaf, and $\mathcal{D}_{\theta_D}(\cdot)$ is a neural network approximator that predicts noise δ conditioned on both the noisy sample $\tilde{\mathbf{x}}^t$ and the hierarchical path $\mathbf{s}_i$, ensuring generated samples conform to the learned tree structure.

Although the hierarchical codebook $\mathcal{C}_W$ is parameterized as a full binary tree for efficient indexing, this does not restrict the generated tree topology to be binary (see Appendix Fig. B.1). Each data point follows a binary latent path during quantization, but multiple points can share partial paths and diverge at different levels, which naturally induces multi-branch structures in the resulting hierarchy.

Thus, HDTree is capable of representing arbitrary n -ary trees and unbalanced hierarchies, with the binary tree serving purely as an indexing mechanism for hierarchical codes.

3.3 Loss Function Design

To optimize the HDTree model, we design a composite loss function that integrates contrastive learning, vector quantization, and diffusion-based reconstruction.

3.3.1 Soft Contrastive Learning Loss (SCL) [$\mathcal{L}_{\text{SCL}}(\cdot)$]

Standard contrastive methods treat all negative pairs equally, failing to capture the *graded similarities* in hierarchical data (e.g., same-genus samples are more similar than cross-phylum ones). SCL (Zang et al., 2024a) addresses this by assigning distance-dependent penalties via tree-based weights to preserve hierarchical relationships. For a batch of embeddings $\mathbf{z} = \{\mathbf{z}_i\}_{i=1}^{N_b}$ and augmentations $\{\mathbf{z}_i^+\}_{i=1}^{N_b}$, the loss is:

$$\mathcal{L}_{\text{SCL}} = \frac{1}{2N} \sum_{i_1=1}^{N_b} \left(\log \sum_{i_2=1}^{N_b} \mathbf{S}_{i_1 i_2}^{\mathbf{z} \mathbf{z}^+} + \log \sum_{i_2=1}^{N_b} \mathbf{S}_{i_1 i_2}^{\mathbf{z}^+ \mathbf{z}} \right) - \sum_{i_1=1}^{N_b} \log \text{diag}(\mathbf{S}_{i_1 i_1}^{\mathbf{z} \mathbf{z}^+}), \quad (4)$$

where N_b is the batch size, $\mathbf{S}_{ij}^{\mathbf{z} \mathbf{z}^+}$ represents the similarity matrix calculated using the t-distribution kernel, $\mathbf{S}_{i_1 i_2} = (1 + (\mathbf{D}_{i_1 i_2}^2)/\nu)^{-\frac{\nu+1}{2}}$, and $\mathbf{D}_{i_1 i_2}$ is the pairwise distance between $\mathbf{z}_{i_1}$ and $\mathbf{z}_{i_1 i_2}$ in the hyperbolic space, where $\nu = 0.1$ is the degrees of freedom of the t-distribution.

3.3.2 Hierarchical Quantization Loss (HQL) [$\mathcal{L}_{\text{HQL}}(\cdot)$]

HQL learns robust hierarchical tree-structured representations in the HTC by aligning latent embeddings with multi-level code vectors while maintaining inter-level consistency. Vanilla vector quantization (Van Den Oord et al., 2017) only constrains leaf-level representations, ignoring intermediate hierarchical consistency and causing codebook collapse. Unlike vanilla VQ encoders, which rely on initialization and local perceptual losses, HQL addresses this by enforcing multi-level alignment to capture global structural relationships across the entire tree structure. The loss is defined as,

$$\mathcal{L}_{\text{HQL}} = \sum_{l=1}^L \mathcal{A}(\mathbf{z}_i, c_{\mathbf{z}_i}^l) + \lambda \mathcal{A}(c_{\mathbf{z}_i}^l, \Psi^{\mathbf{z}_i}(c_{\mathbf{z}_i}^l)), \quad \Psi^{\mathbf{z}}(\mathbf{w}_j) = \arg \min_{\mathbf{z}_i \in \mathbf{z}} \|\mathbf{z}_i - \mathbf{w}_j\|_2, \quad (5)$$

where $c_{\mathbf{z}_i}^l$ is the nearest code vector to $\mathbf{z}_i$ at level l , and $\lambda = 2$ balances alignment and consistency. The first term $\mathcal{A}(\mathbf{z}_i, c_{\mathbf{z}_i}^l)$ aligns embeddings with codes to preserve parent-child relations, while the second term $\mathcal{A}(c_{\mathbf{z}_i}^l, \Psi^{\mathbf{z}_i}(c_{\mathbf{z}_i}^l))$ enforces consistency by mapping codes back to their nearest embeddings, separating sibling nodes and anchoring children to their parents to maintain a coherent hierarchy. The $\mathbf{z} = \{\mathbf{z}_i\}_{i=1}^{N_b}$ denotes the latent embeddings in the current batch. The function $\mathcal{A}(a, b) = \|\mathbf{sg}(a) - b\|_2^2 + \|a - \mathbf{sg}(b)\|_2^2$, where $\mathbf{sg}(\cdot)$ denotes the stop-gradient operation.

3.3.3 Diffusion Loss [$\mathcal{L}_{\text{DDP}}(\cdot)$]

The diffusion loss trains the decoder $\mathcal{D}_{\theta_D}$ to predict the added Gaussian noise at each step and progressively denoise the sample. Following the standard DDPM formulation (Ho et al., 2020a), we define

$$\mathcal{L}_{\text{DDP}} = \mathbb{E}_{t \sim [1, T], \mathbf{x}, \epsilon \sim \mathcal{N}(0, I)} \left[\|\epsilon - \epsilon_{\theta_D}(\sqrt{\alpha_t} \mathbf{x} + \sqrt{1 - \alpha_t} \epsilon, t, \mathbf{s}_i)\|_2^2 \right], \quad (6)$$

where $\epsilon_{\theta_D}(\cdot)$ is the predicted Gaussian noise, β_t is the variance schedule, $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ is the cumulative product term. The conditional vector $\mathbf{s}_i$ is obtained from Eq. (2). This loss enforces the decoder to match the true noise ϵ and thus ensures faithful reconstruction of $\mathbf{x}$ while respecting the hierarchical conditions.

3.3.4 Overall Loss Function

The overall loss is defined as,

$$\mathcal{L} = \mathcal{L}_{\text{SCL}} + \lambda_{\text{HQL}} \mathcal{L}_{\text{HQL}} + \lambda_{\text{DDP}} \mathcal{L}_{\text{DDP}}, \quad (7)$$

where λ_{HQL} , and λ_{DDP} are hyperparameters controlling the contributions of the contrastive learning loss, vector quantization loss, and diffusion-based reconstruction loss, respectively. This composite loss ensures balanced

Algorithm 1: Training HDTree

Input: $\mathbf{X}, \mathbf{X}^+$; params $\Theta = \{\phi, W, \theta\}$; lr η ; batch N_b

Output: Θ^*

Init: random Θ

for mini-batch $\mathcal{B} = \{(\mathbf{x}_i, \mathbf{x}_i^+)\}_{i=1}^{N_b}$ **do**

 Encode: $\mathbf{z}_i = \mathcal{E}_\phi(\mathbf{x}_i)$, $\mathbf{z}_i^+ = \mathcal{E}_\phi(\mathbf{x}_i^+)$

 Quantize: $\mathbf{s}_i$ via HTC Eq. (2)

 Losses: $\mathcal{L}_{\text{SCL}}$ Eq. (4), $\mathcal{L}_{\text{HQL}}$ Eq. (5)

 DDPM step: sample $t \sim [1, T]$, $\epsilon \sim \mathcal{N}(0, I)$,

$\mathbf{x}_t = \sqrt{\alpha_t} \mathbf{x}_i + \sqrt{1 - \alpha_t} \epsilon$;

$\mathcal{L}_{\text{DDP}} = \|\epsilon - \epsilon_{\theta}(\mathbf{x}_t, t, \mathbf{s}_i)\|_2^2$ (Eq. 6)

5 Total: $\mathcal{L} = \mathcal{L}_{\text{SCL}} + \lambda_{\text{HQL}} \mathcal{L}_{\text{HQL}} + \lambda_{\text{DDP}} \mathcal{L}_{\text{DDP}}$

Update: $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}$

return Θ^*

optimization across all components of the HDTree model. The pseudocode for training the HDTree model is provided in Algorithm 1.

3.4 Trajectory Analysis with HDTree

3.4.1 Graph Construction

To infer developmental trajectories, the hierarchical tree structure generated by HDTree is transformed into a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$. The graph $\mathcal{G}$ inherits all nodes and edges from the hierarchical tree $\mathcal{T}$, while additional edges are added within the same depth using a k -nearest neighbors (KNN) approach. This augmentation enriches the connectivity of the graph by capturing local semantic relationships that are not explicitly represented in the original tree structure. The nodes in the graph correspond to the code vectors $\mathbf{w}_j^l$ at each depth l and index j . The edges include both the hierarchical edges from $\mathcal{T}$ and the newly introduced edges generated by the KNN process. For each edge (j_1, j_2) in the graph, the weight is,

$$\mathbf{w}(j_1, j_2) = \begin{cases} \|\mathbf{w}_{j_1} - \mathbf{w}_{j_2}\|_2, & j_1 = j_2 \\ \|\mathbf{w}_{j_1} - \mathbf{w}_{j_2}\|_2 + P^{L-l}, & j_1 \neq j_2, \end{cases} \quad (8)$$

where P^{L-l} is a penalty term that increases the weight of edges connecting nodes at different depths, ensuring that the developmental trajectories follow the hierarchical structure.

3.4.2 Trajectory Inference

Using the constructed graph $\mathcal{G}$, developmental trajectories are inferred by identifying the shortest path between a predefined origin $\mathbf{w}_{\text{start}}$ and a destination $\mathbf{w}_{\text{end}}$. The shortest path is computed by minimizing the total edge weights along the trajectory,

$$\text{Path}_{\text{development}} = \arg \min_{\text{Path} \subseteq \mathcal{G}} \sum_{e \in \text{Path}} \mathbf{w}(j_1, j_2). \quad (9)$$

The inferred developmental trajectories provide a comprehensive representation of the underlying hierarchical relationships in the data, capturing the transitions between different cell states and the progression of cell differentiation processes. The KNN is used only as an auxiliary augmentation to enrich local connectivity and does not alter the global hierarchy captured by $\mathcal{T}$. Empirically, our results are stable across a wide range of k values (see Appendix for sensitivity analysis), indicating that the primary performance gain comes from the tree-structured representation itself.

4 EXPERIMENTS

4.1 Datasets & Baseline Methods

To provide a comprehensive comparison of different methods, we use two types of datasets, general tabular, image, and text datasets (Mnist, Fashion-Mnist, 20news-groups, Cifar10) and single-cell datasets (Limb(Zhang et al., 2023), LHCO(He et al., 2022b), Weinreb(Weinreb et al., 2020b), ECL(Qiu et al., 2024b)). The scale and features of these datasets are detailed in Table. 1 and 2. Baseline methods include traditional approaches, such as Agglomerative Clustering (Agg) (Müllner, 2011), t-SNE (Linderman & Steinerberger, 2019), and UMAP (Dalmia & Sia, 2021), as well as state-of-the-art (SOTA) deep learning methods, including VAE (Doersch, 2016; Lim et al., 2020), LadderVAE (Sønderby et al., 2016), DeepECT (Mautz et al., 2020), and TreeVAE (Manduchi et al., 2024). Additionally, specialized models (Geneformer (Theodoris et al., 2023), LangCell (Zhao et al., 2024), CellPLM (Wen et al., 2024)) tailored for the single-cell domain are incorporated to ensure a thorough evaluation across diverse tasks. More details are provided in the Appendix.

4.2 Evaluation Metrics

To comprehensively evaluate HDTree and baseline methods, the testing protocol is divided into three parts: clustering performance, tree structure performance, and reconstruction performance. *Clustering performance* is measured using *Clustering Accuracy (ACC)* (Nazeer et al., 2009) and *Normalized Mutual Information (NMI)*(Estévez et al., 2009). To obtain these metrics, the input data is first mapped into a latent space using the respective method. The clustering results are then derived directly from this latent representation. For methods that do not inherently produce clustering results, hierarchical clustering is applied to the latent space to generate cluster labels. This ensures a fair and consistent comparison across all evaluated methods.

Table 1: **Comparison of tree performance, clustering performance, and reconstruction performance (Rec. Performance) on four generall image and text datasets.** The ^A means directly use agglomerative clustering on the embeddings to gat the tree performance. The -RL and LL are the reconstruction loss and negative log-likelihood. The best results are highlighted in **bold**. The number after/before $\pm$ shows the mean/standard deviation with 10 different random seeds. ‘NG’ indicates these methods do not have the generation ability.

Dataset	Method	Tree Performance		Clustering Performance		Rec. Performance		Average
		DP($\uparrow$)	LP($\uparrow$)	ACC($\uparrow$)	NMI($\uparrow$)	-RL($\uparrow$)	LL($\uparrow$)	
Mnist (image, 70k $\times$ 784)	Agg	63.7 $\pm$ 0.0	78.6 $\pm$ 0.0	69.5 $\pm$ 0.0	71.1 $\pm$ 0.0	NG	NG	NG
	VAE ^A	79.9 $\pm$ 2.2	90.8 $\pm$ 1.4	86.6 $\pm$ 4.9	81.6 $\pm$ 2.0	-84.7 $\pm$ 2.6	-87.2 $\pm$ 2.0	27.8 $\pm$ 2.5
	LadderVAE ^A	81.6 $\pm$ 3.9	90.9 $\pm$ 2.5	80.3 $\pm$ 5.6	82.0 $\pm$ 2.1	-87.8 $\pm$ 0.7	-99.9 $\pm$ 0.3	24.5 $\pm$ 2.5
	DeepECT	74.6 $\pm$ 5.9	90.7 $\pm$ 3.2	74.9 $\pm$ 6.2	76.7 $\pm$ 4.2	NG	NG	NG
	TreeVAE	87.9 $\pm$ 4.9	96.0 $\pm$ 1.9	90.2 $\pm$ 7.5	90.0 $\pm$ 4.6	-80.3 $\pm$ 0.2	-92.9 $\pm$ 0.2	31.8 $\pm$ 3.2
	HDTTree ^A	92.7$\pm$0.3	97.1$\pm$1.2	97.1$\pm$0.1	92.8$\pm$0.2	NG	NG	NG
	HDTTree	91.9$\pm$2.8	96.6$\pm$1.4	96.6$\pm$1.4	92.4$\pm$1.3	-77.9$\pm$1.2	-85.4$\pm$1.4	35.7$\pm$1.6 ($\uparrow$3.9)
Fashion-Mnist (image, 70k $\times$ 784)	Agg	45.0 $\pm$ 0.0	67.6 $\pm$ 0.0	51.3 $\pm$ 0.0	52.6 $\pm$ 0.0	NG	NG	NG
	VAE ^A	44.3 $\pm$ 2.5	65.9 $\pm$ 2.3	54.9 $\pm$ 4.4	56.1 $\pm$ 3.2	-231 $\pm$ 3.2	-242 $\pm$ 3.2	-32.1 $\pm$ 3.1
	LadderVAE ^A	49.5 $\pm$ 2.3	67.6 $\pm$ 1.2	55.9 $\pm$ 3.0	60.7 $\pm$ 1.4	-231 $\pm$ 1.4	-239 $\pm$ 1.4	-39.5 $\pm$ 1.8
	DeepECT	44.9 $\pm$ 3.3	67.8 $\pm$ 1.4	51.8 $\pm$ 5.7	57.7 $\pm$ 3.7	NG	NG	NG
	TreeVAE	53.4 $\pm$ 2.4	70.4 $\pm$ 2.0	60.6 $\pm$ 3.3	64.7 $\pm$ 1.4	-226 $\pm$ 1.4	-234 $\pm$ 1.4	-35.4 $\pm$ 2.0
	HDTTree ^A	47.7$\pm$1.6	67.1$\pm$1.5	64.6$\pm$1.9	67.4$\pm$1.2	NG	NG	NG
	HDTTree	57.4$\pm$0.3	71.8$\pm$0.3	71.1$\pm$0.2	68.7$\pm$0.2	-219$\pm$0.1	-228$\pm$0.1	-29.9$\pm$0.2 ($\uparrow$5.5)
20news-groups (text, 19k $\times$ 2000)	Agg	13.1 $\pm$ 0.0	30.8 $\pm$ 0.0	26.1 $\pm$ 0.0	27.5 $\pm$ 0.0	NG	NG	NG
	VAE ^A	7.1 $\pm$ 0.3	18.1 $\pm$ 0.5	15.2 $\pm$ 0.4	11.6 $\pm$ 0.3	-45.5 $\pm$ 0.1	-44.2 $\pm$ 0.3	-6.3 $\pm$ 0.3
	LadderVAE ^A	9.0 $\pm$ 0.2	20.0 $\pm$ 0.7	17.4 $\pm$ 0.9	17.8 $\pm$ 0.6	-43.5 $\pm$ 0.1	-44.3 $\pm$ 0.6	-3.9 $\pm$ 0.5
	DeepECT	9.3 $\pm$ 1.8	17.2 $\pm$ 3.8	15.6 $\pm$ 3.0	18.1 $\pm$ 4.1	NG	NG	NG
	TreeVAE	17.5 $\pm$ 1.5	38.4 $\pm$ 1.6	32.8 $\pm$ 2.3	34.4 $\pm$ 1.5	-34.4 $\pm$ 1.5	-34.4 $\pm$ 1.5	9.1 $\pm$ 1.7
	HDTTree ^A	22.0$\pm$0.1	45.5$\pm$0.4	44.6$\pm$0.4	43.7$\pm$0.2	NG	NG	NG
	HDTTree	23.7$\pm$0.1	44.0$\pm$0.2	41.8$\pm$0.2	42.6$\pm$0.2	-31.1$\pm$0.3	-34.1$\pm$1.5	19.0$\pm$0.4 ($\uparrow$9.9)
Cifar10 (image, 50k $\times$ 32 $\times$ 32)	VAE ^A	10.5 $\pm$ 2.3	16.3 $\pm$ 2.3	16.3 $\pm$ 1.6	1.86 $\pm$ 4.2	-31.7$\pm$2.9	-39.2$\pm$2.9	-4.3 $\pm$ 2.7
	LadderVAE ^A	12.8 $\pm$ 3.9	25.3 $\pm$ 3.9	25.3 $\pm$ 2.0	7.41 $\pm$ 4.9	-41.8 $\pm$ 4.7	-40.2 $\pm$ 3.7	-1.9 $\pm$ 3.9
	DeepECT	10.5 $\pm$ 2.5	10.3 $\pm$ 2.5	10.3 $\pm$ 2.8	0.18 $\pm$ 4.2	NG	NG	NG
	TreeVAE	35.3 $\pm$ 4.0	53.8 $\pm$ 3.9	52.9 $\pm$ 7.0	41.4 $\pm$ 5.9	-47.0 $\pm$ 5.9	-48.3 $\pm$ 2.4	14.7 $\pm$ 4.9
	HDTTree ^A	44.2$\pm$1.5	55.2$\pm$1.8	75.9$\pm$4.3	55.3$\pm$2.5	NG	NG	NG
	HDTTree	43.8$\pm$1.7	55.1$\pm$1.4	73.2$\pm$2.7	53.9$\pm$2.0	-34.7 $\pm$ 1.9	-40.3 $\pm$ 3.6	25.2$\pm$2.2 ($\uparrow$10.5)

Tree structure performance (Tree performance) is evaluated using *Leaf Purity (LP)* (Schütze et al., 2008) and *Dendrogram Purity (DP)* (Rokach & Maimon, 2005). We predict the tree structures with different methods. **Reconstruction performance** is assessed using *Reconstruction Loss (RL)* and *Log-Likelihood (LL)*. These metrics quantify the ability of a method to recover the original input data from its latent space representation. Details on evaluation metrics are provided in the Appendix.

4.3 Testing Protocol & Implementation

For all experiments, the data is split into training, validation, and testing sets with an 8:1:1 ratio, ensuring unbiased evaluation. In testing, if the number of points in the dataset is greater than 10,000, we randomly sample 10,000 points from testing dataset. **It is important to clarify that downsampling is purely an evaluation strategy to accelerate metric computation (particularly for clustering metrics like ACC/NMI which require expensive assignment operations), not a model limitation. Our model performs training and inference on complete datasets.** Details on downsampling and its rationale are provided in the Appendix. We implemented HDTTree using PyTorch and trained the model on a single NVIDIA A100 GPU. The model is trained using the AdamW optimizer with a learning rate of 1e-4 and a batch size of 128. The number of diffusion steps T is set to 1000, and the tree depth L is set to 10. More details on the implementation are provided in the Appendix and code.

4.4 Comparisons on General Datasets [better stability/accuracy/generativity]

The proposed HDTTree is both a tree generation and data generation method. To ensure a fair comparison of clustering, tree construction, and generation performance across different methods, we adopted the benchmarking strategy described in the benchmark of (Manduchi et al., 2024). The results are shown in Table 1. The ^A means the methods directly use the agglomerative clustering method on the embeddings to calculate the Tree Performance. **Analysis:** (1) HDTTree achieves superior performance across all evaluated metrics, outperforming traditional and SOTA. This advantage stems from HDTTree’s explicit consideration of hierarchical tree structures, which enhances its ability to capture underlying data relationships. (2) Unlike TreeVAE, HDTTree uses a unified tree

Table 2: **Comparison of tree performance, clustering performance on three single cell datasets.** Since most of the methods are not generative models, we did not compare generative performance.

Dataset	Method	Year	Tree Performance		Clustering Performance		Average($\uparrow$)
			DP($\uparrow$)	LP($\uparrow$)	ACC($\uparrow$)	NMI($\uparrow$)	
Limb (cell lineage, 66,633 cells, celltype:10)	Geneformer ^A	2023	25.6 $\pm$ 5.4	35.9 $\pm$ 0.1	34.1 $\pm$ 0.1	34.9 $\pm$ 0.1	32.6 $\pm$ 1.4
	CellPLM ^A	2024	25.6 $\pm$ 0.1	39.9 $\pm$ 0.1	34.1 $\pm$ 0.2	32.9 $\pm$ 0.2	33.1 $\pm$ 0.2
	LangCell ^A	2024	25.3 $\pm$ 0.1	37.5 $\pm$ 0.1	33.9 $\pm$ 0.1	35.1 $\pm$ 0.1	33.0 $\pm$ 0.1
	TreeVAE ^A	2024	34.7 $\pm$ 1.7	55.6 $\pm$ 1.0	49.8 $\pm$ 0.1	50.0 $\pm$ 0.0	47.5 $\pm$ 0.7
	HDTree ^A	Ours	38.9$\pm$1.3	57.9$\pm$1.0	52.8$\pm$1.0	49.0$\pm$0.1	49.7$\pm$0.9
	HDTree	Ours	41.0$\pm$0.4	57.2$\pm$1.4	55.0$\pm$1.4	46.6$\pm$0.4	50.0$\pm$0.9 ($\uparrow$2.5)
LHCO (cell lineage, 10,628 cells, celltype:7)	CellPLM ^A	2024	27.0 $\pm$ 1.1	35.8 $\pm$ 2.7	16.8 $\pm$ 3.4	1.65 $\pm$ 5.2	20.3 $\pm$ 3.1
	LangCell ^A	2024	26.5 $\pm$ 1.2	35.2 $\pm$ 0.8	35.2 $\pm$ 0.6	0.02 $\pm$ 0.9	24.2 $\pm$ 0.9
	TreeVAE ^A	2024	38.3 $\pm$ 2.0	52.2 $\pm$ 0.1	37.9 $\pm$ 0.1	31.6 $\pm$ 0.0	40.0 $\pm$ 0.6
	HDTree ^A	Ours	38.8$\pm$0.3	52.1$\pm$0.4	46.4$\pm$0.3	34.7$\pm$0.5	43.0$\pm$0.4
	HDTree	Ours	42.7$\pm$0.4	54.0$\pm$0.3	49.4$\pm$0.3	34.5$\pm$0.4	45.2$\pm$0.3 ($\uparrow$2.2)
Weinreb (cell lineage, 130,887 cells, celltype:11)	LangCell ^A	2024	47.4 $\pm$ 0.1	54.8 $\pm$ 0.0	14.3 $\pm$ 0.5	34.3 $\pm$ 0.0	37.7 $\pm$ 0.2
	Geneformer ^A	2024	45.1 $\pm$ 0.4	55.3 $\pm$ 0.1	21.4 $\pm$ 0.1	32.3 $\pm$ 0.1	38.5 $\pm$ 0.2
	TreeVAE ^A	2024	60.4 $\pm$ 2.6	61.4 $\pm$ 0.5	41.0 $\pm$ 0.1	35.2 $\pm$ 0.0	49.5 $\pm$ 0.8
	HDTree ^A	Ours	63.3$\pm$2.6	78.2$\pm$1.1	50.6$\pm$1.0	45.2$\pm$1.2	59.3$\pm$1.5 ($\uparrow$7.5)
	HDTree	Ours	61.0$\pm$0.4	67.0$\pm$0.3	62.6$\pm$0.3	42.6$\pm$0.3	58.3$\pm$0.4

Table 3: **Comparisons on Lineage Ground Truth.** The ratio of observed time points (Lineage Ground Truth) in the k -neighborhood ($k=30$). Top: LineageVAE dataset. Bottom: *C. elegans* dataset.

	Time	Waddington OT	LineageVAE (semi-supervised)	scVI+Agg (unsupervised)	TreeVAE (unsupervised)	HDTree (unsupervised)
LineageVAE Dataset	Day 2	22.1%	2.2%	16.1%	12.1%	23.2% ($\uparrow$ +1.1%)
	Day 4	21.4%	37.4%	28.2%	30.4%	38.4% ($\uparrow$ +1.0%)
	Day 6	56.6%	60.3%	53.2%	56.4%	62.0% ($\uparrow$ +1.7%)
<i>C. Elegans</i> Dataset	100–300	–	15.4%	10.8%	13.8%	15.2% ($\uparrow$ –0.2%)
	300–500	–	41.5%	40.6%	32.5%	45.8% ($\uparrow$ +4.3%)
	500–750	–	62.3%	51.8%	62.8%	66.3% ($\uparrow$ +4.0%)

representation framework, leading to lower standard variance and enhanced stability. (3) The HDTree exhibits smaller variance and generally better performance than HDTree^A, demonstrating that its hierarchical tree representation effectively enhances modeling and generation capabilities. (4) HDTree’s advantages become more pronounced with dataset complexity increases, highlighting its robustness and effectiveness.

4.5 Comparisons on Single Cell Datasets [better stability/accuracy]

Due to the lack of established benchmarks in the single-cell domain, we incorporated data from high-impact published studies into the TreeVAE benchmark. The results are shown in Table 2. **Analysis:** (1) Similar to the general dataset, HDTree consistently demonstrates superior performance across all evaluated metrics, including tree structure quality, clustering accuracy, and hierarchical integrity. (2) We observed that the zero-shot capabilities of single-cell large language models are often unsatisfactory and, in some cases, fail to surpass basic single-cell methods. This conclusion has also been validated in recent studies (Lan et al., 2024; He et al., 2024). In comparison with foundational single-cell models, traditional single-cell tree analysis methods, and TreeVAE, HDTree shows relative advantages in performance and achieves better stability. (3) These results establish HDTree as a robust and reliable approach for single-cell data analysis.

4.6 Comparisons on Lineage Ground Truth [better stability/accuracy]

We evaluate the alignment between latent space structure and true developmental progression using the ratio of observed time points (Lineage Ground Truth) in the k -nearest neighborhood ($k = 30$). As shown in Table 3, HDTree consistently outperforms both classical and recent unsupervised methods across two benchmark datasets. On the LineageVAE dataset, HDTree achieves the highest local temporal consistency on all time points, even surpassing the semi-supervised LineageVAE by +1.7% at Day 6. This highlights the advantage of our diffusion-based hierarchical modeling in capturing temporal lineage progression without relying on labeled supervision. **Analysis:** On the *C. elegans* dataset, HDTree maintains strong performance across early, mid, and late developmental stages. It improves over TreeVAE by +4.3% and +4.0% in the 300-500 and 500-750 windows,

Table 4: **Ablation Study on MNIST & ECL Datasets.** The best performance is highlighted in **bold**.

Ablation Setups	MNIST (General Dataset)				ECL (Single-Cell Dataset)			
	DP($\uparrow$)	LP($\uparrow$)	ACC($\uparrow$)	NMI($\uparrow$)	DP($\uparrow$)	LP($\uparrow$)	ACC($\uparrow$)	NMI($\uparrow$)
A1. Full Model (Ours)	92.7	97.1	96.6	92.4	69.0	83.2	83.2	79.0
A2. w/o HTC	87.4	85.3	84.1	75.2	58.7	71.4	70.8	66.5
A3. w/o SCL ($\mathcal{L}_{SCL}$)	78.9	82.1	81.5	73.1	55.6	68.9	68.3	63.1
A4. w/o HQL ($\mathcal{L}_{HQL}$)	84.1	89.3	86.8	81.7	61.4	74.2	73.7	69.4

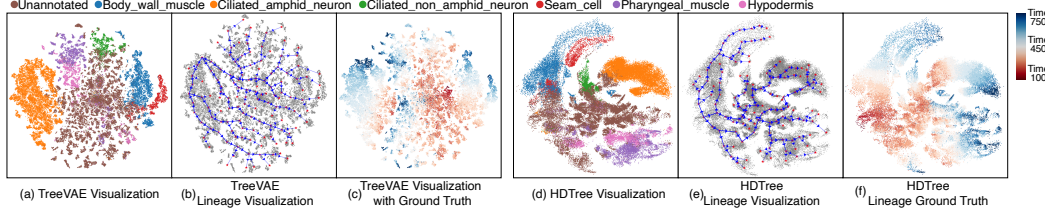


Figure 3: **Comparison of TreeVAE and HDTree methods for visualization and lineage inference.** (a) and (d) are the latent space visualization of TreeVAE and HDTree, color shows the cell type information. (b) and (e) are the lineage structure inferred by TreeVAE and HDTree, overlaid on the data distribution. The blue arrows indicate the inferred lineage relationships. (c) and (f) are the ground truth lineage visualization for comparison, the color shows the real-time information (from blue to red). The results show that HDTree captures more accurate lineage relationships and generates more realistic data than TreeVAE.

respectively, and even slightly outperforms the supervised LineageVAE in the early stage. These results demonstrate that HDTree’s hierarchical latent structure provides a more faithful reflection of biological differentiation dynamics, offering robust generalization across varying levels of trajectory complexity.

4.7 Case Study on HDTree Data Generation [better generativity]

The diffusion generation of HDTree can generate transformation processes between different tree branches according to the tree structure, which is very important in phylogenetic analysis because often people are interested in how different cell types are transformed under natural conditions (for example, from stem cells to somatic cells). We demonstrate how HDTree solves the above problem based on two datasets (MNIST and C. elegans).

The results are shown in Fig. 4. **Analysis:** (1) The results on the generic MNIST data show the visualization of the result from the number 6 to three different numbers (3, 1, 9). The result shows that the model can complete the generation process well, and the data changes slowly throughout the process. (2) The results on C. elegans data show the visualization of the result from the stem cell to the somatic cell. Although we cannot visually display every gene, we have selected three iconic genes and then used pie charts to display the trend of data changes.

4.8 Case Study on C. elegans Lineage [better accuracy]

To evaluate the performance of HDTree on real lineage labels, we utilized labeled data provided by (Packer et al., 2019). The C. elegans dataset not only labels the type of cell but also the relative time at which the cell is collected, which can be regarded as the gold label for our analysis of the cell’s differentiation lineage. HDTree demonstrates superior performance compared to TreeVAE in capturing lineage relationships and generating biologically meaningful results. A detailed introduction is in the caption of Fig. 3. **Analysis:** (1) By analyzing Fig. 3(a) and Fig. 3(d), both TreeVAE and HDTree can distinguish different cell types well because they map cells of the same type to similar locations. (2) TreeVAE cannot accurately model the differentiation process of cells. This is because

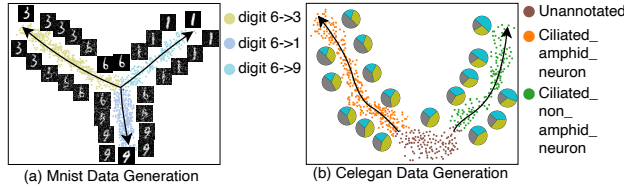


Figure 4: **Data generation of HDTree on MNIST and C. elegans.** Each scatter is the generated data visualized by tSNE. Color indicates the label. For MNIST, data is generated from digit 6 to 3, 1, and 9. For C. elegans, from stem cell to somatic cell (pie charts: **odr-10**, **osm-6**, **elt-5**.)

the differentiation lineage visualization (Fig. 3(b)) based on the TreeVAE representation does not match the real-time gold label (Fig. 3(c)). In contrast, the differentiation lineage inferred by HDTree basically (Fig. 3(e)) overlaps with the time gold label (Fig. 3(f)).

4.9 Comparisons on Computational Cost [better effectively]

To evaluate the computational efficiency of HDTree, we compared the training time of HDTree with TreeVAE and TreeVAE^A on four dataset (in Table 5). We observe that traditional methods do have an advantage when dealing with small datasets. However, when the dataset size becomes large, traditional methods will become slow due to the their complexity.

Table 5: Training time comparison on general and single-cell datasets. **Bold** denotes the best result. (mm:ss)

	tSNE+Agg	UMAP+Agg	TreeVAE	HDTree
MNIST	854:10	2:09	192:09	42:23
F-MNIST	915:13	2:22	206:13	45:02
LHCO	1708:28	13:51	246:20	53:23
Weinreb	5879:27	340:18	361:12	53:47

4.10 Ablation Study [better accuracy]

To evaluate the contributions of HDTree’s components, we conducted ablation experiments on MNIST and ECL datasets. The setups included (A1) Full Model (HDTree), (A2) without the HCL and directly use vanilla codebook, (A3) without the SCL ($\mathcal{L}_{SCL}$) and directly use the contrastive learning loss, and (A4) without the HQL loss and directly use the VQ Loss ($\mathcal{L}_{hq}$). Performance is measured using tree structure (DP , LP) and clustering metrics (ACC , NMI). The results are shown in Table 4. **Analysis:** The full model consistently achieved the best performance. Removing the HCL (A2) caused the most significant performance drop across both datasets, highlighting its role in structural and clustering performance. The SCL (A3) is essential for maintaining tree depth and clustering interpretability. All components contribute to HDTree’s success, with the HCL and contrastive loss being the most critical for optimal performance.

4.11 Parameter Sensitivity Analysis [better stability]

To evaluate the impact of latent dimensionality on model performance, we conducted a sensitivity analysis using tree performance (DP) as the primary metric. The baseline and SOTA methods are evaluated on the ECL dataset with varying latent dimensionalities. The results are presented in Fig. 6. **Analysis:** (1) HDTree achieves its best performance at a latent dimensionality of 64-256, beyond which no further improvements are observed.

This observation suggests that HDTree effectively captures the essential structural information at moderate dimensionalities, avoiding over-parameterization. (2) Moreover, regardless of the choice of dimensionality, HDTree maintains a consistent advantage over competing methods, underscoring its superiority in preserving tree structures and hierarchical relationships. The sensitivity of HDTree to key hyperparameters is examined.

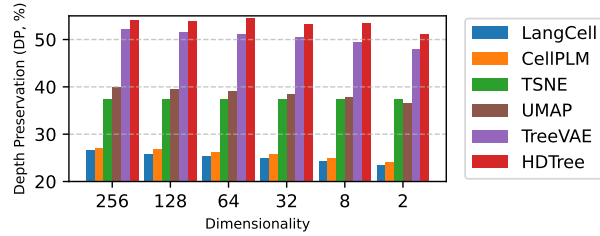


Table 6: **Sensitivity Analysis of Dimensionality on Dendrogram Purity (DP) Across Methods.** The performance of various methods on the ECL dataset with different latent dimensionalities.

5 CONCLUSION

We introduce HDTree, a unified diffusion-based framework for hierarchical representation and data generation. By combining a quantized diffusion process with a hierarchical codebook, HDTree captures tree-structured relationships without relying on branch-specific modules, leading to enhanced stability, generative quality, and interpretability. Experimental results demonstrate consistent improvements in clustering accuracy, tree structure fidelity, and lineage alignment. **Limitations.** Although HDTree achieves high-quality hierarchical generation, its diffusion-based decoder remains computationally expensive during sampling, especially for large-scale datasets. Future work will focus on accelerating generation via fast-sampling strategies and more efficient latent diffusion schemes.

REFERENCES

- Parveen Ali and Ahtisham Younas. Understanding and interpreting regression analysis. *Evidence-Based Nursing*, 24(4):116–118, 2021.
- Athman Bouguettaya, Qi Yu, Xumin Liu, Xiangmin Zhou, and Andy Song. Efficient agglomerative hierarchical clustering. *Expert Systems with Applications*, 42(5):2785–2797, 2015.
- Morteza Haghiri Chehreghani and Mostafa Haghiri Chehreghani. Hierarchical correlation clustering and tree preserving embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23083–23093, 2024.
- Ayush Dalmia and Suzanna Sia. Clustering with umap: Why and how connectivity matters. *arXiv preprint arXiv:2108.05525*, 2021.
- Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.
- Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. Normalized mutual information feature selection. *IEEE Transactions on neural networks*, 20(2):189–201, 2009.
- Zoubin Ghahramani, Michael Jordan, and Ryan P Adams. Tree-structured stick breaking for hierarchical data. *Advances in neural information processing systems*, 23, 2010.
- Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10696–10706, 2022.
- Zhiye Guo, Jian Liu, Yanli Wang, Mengrui Chen, Duolin Wang, Dong Xu, and Jianlin Cheng. Diffusion models in bioinformatics and computational biology. *Nature reviews bioengineering*, 2(2):136–154, 2024.
- Croix Gyurek, Niloy Talukder, and Mohammad Al Hasan. Binder: Hierarchical concept representation through order embedding of binary vectors. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 980–991, 2024.
- Laleh Haghverdi et al. Diffusion pseudotime robustly reconstructs lineage branching. *Nature Methods*, 13(10):845–848, 2016.
- Fei He, Ruixin Fei, Mingyue Gao, Li Su, Xinyu Zhang, and Dong Xu. Parameter-efficient fine-tuning enhances adaptation of single cell large language model for cell type identification. *bioRxiv*, 2024.
- Z. He, A. Maynard, A. Jain, et al. Lineage recording in human cerebral organoids. *Nat Methods*, 19: 90–99, 2022a. doi: 10.1038/s41592-021-01344-8.
- Zhisong He, Ashley Maynard, Akanksha Jain, Tobias Gerber, Rebecca Petri, Hsiu-Chuan Lin, Malgorzata Santel, Kevin Ly, Jean-Samuel Dupré, Leila Sidow, et al. Lineage recording in human cerebral organoids. *Nature methods*, 19(1):90–99, 2022b.
- Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *NeurIPS*, 2020a. URL <https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html>.
- Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020b.
- Lun Huang, Qiang Qiu, and Guillermo Sapiro. Pq-vae: Learning hierarchical discrete representations with progressive quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7550–7558, 2024.
- Weikuan Jia, Meili Sun, Jian Lian, and Sujuan Hou. Feature dimensionality reduction: a review. *Complex & Intelligent Systems*, 8(3):2663–2693, 2022.
- Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

- Jogendra Nath Kundu, Maharshi Gor, Dakshit Agrawal, and R Venkatesh Babu. Gan-tree: An incrementally learned hierarchical generative framework for multi-modal data distributions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8191–8200, 2019.
- Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests for large-scale regression when uncertainty matters. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.
- Wei Lan, Guohang He, Mingyang Liu, Qingfeng Chen, Junyue Cao, and Wei Peng. Transformer-based single-cell language model: A survey. *Big Data Mining and Analytics*, 7(4):1169–1186, 2024.
- Kart-Leong Lim, Xudong Jiang, and Chenyu Yi. Deep clustering with variational autoencoder. *IEEE Signal Processing Letters*, 27:231–235, 2020.
- George C Linderman and Stefan Steinerberger. Clustering with t-sne, provably. *SIAM journal on mathematics of data science*, 1(2):313–332, 2019.
- Qijiong Liu, Xiaoyu Dong, Jiaren Xiao, Nuo Chen, Hengchang Hu, Jieming Zhu, Chenxu Zhu, Tetsuya Sakai, and Xiao-Ming Wu. Vector quantization for recommender systems: A review and outlook. *arXiv preprint arXiv:2405.03110*, 2024.
- Koichiro Majima, Yasuhiro Kojima, Kodai Minoura, Ko Abe, Haruka Hirose, and Teppei Shimamura. Lineagevae: reconstructing historical cell states and transcriptomes toward unobserved progenitors. *Bioinformatics*, 40(10):btac520, 2024.
- Laura Manduchi, Moritz Vandenhirtz, Alain Ryser, and Julia Vogt. Tree variational autoencoders. *Advances in Neural Information Processing Systems*, 36:54952–54986, 2023.
- Laura Manduchi, Moritz Vandenhirtz, Alain Ryser, and Julia Vogt. Tree variational autoencoders. *Advances in Neural Information Processing Systems*, 36:54952–54986, 2024.
- Dominik Mautz, Claudia Plant, and Christian Böhm. Deepect: The deep embedded cluster tree. *Data Science and Engineering*, 5:419–432, 2020.
- Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.
- Fionn Murtagh and Pierre Legendre. Ward’s hierarchical agglomerative clustering method: which algorithms implement ward’s criterion? *Journal of classification*, 31:274–295, 2014.
- KA Abdul Nazeer, MP Sebastian, et al. Improving the accuracy and efficiency of the k-means clustering algorithm. In *Proceedings of the world congress on engineering*, volume 1, pp. 1–3. Association of Engineers London, UK, 2009.
- Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.
- E Oti and M Olusola. Overview of agglomerative hierarchical clustering methods. *British Journal of Computer, Networking and Information Technology*, 7:14–23, 2024.
- Jonathan S Packer, Qin Zhu, Chau Huynh, Priya Sivaramakrishnan, Elicia Preston, Hannah Dueck, Derek Stefanik, Kai Tan, Cole Trapnell, Junhyong Kim, et al. A lineage-resolved molecular atlas of c. elegans embryogenesis at single-cell resolution. *Science*, 365(6459):eaax1971, 2019.
- C. Qiu, B.K. Martin, I.C. Welsh, et al. A single-cell time-lapse of mouse prenatal development from gastrula to birth. *Nature*, 626:1084–1093, 2024a. doi: 10.1038/s41586-024-07069-w.
- Chengxiang Qiu, Beth K Martin, Ian C Welsh, Riza M Daza, Truc-Mai Le, Xingfan Huang, Eva K Nichols, Megan L Taylor, Olivia Fulton, Diana R O’Day, et al. A single-cell time-lapse of mouse prenatal development from gastrula to birth. *Nature*, 626(8001):1084–1093, 2024b.
- Lior Rokach and Oded Maimon. Clustering methods. *Data mining and knowledge discovery handbook*, pp. 321–352, 2005.

- Vladimir Rokhlin and Mark Tygert. Hierarchical clustering for data sets and networks: Practical issues and adaptive algorithms. *Proceedings of the National Academy of Sciences*, 114(29): 7535–7540, 2017.
- Geoffrey Schiebinger et al. Optimal transport for developmental trajectories. *Proceedings of NeurIPS*, 2021.
- Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008.
- Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. *Advances in neural information processing systems*, 29, 2016.
- Kelly Street, Davide Risso, Robert B Fletcher, et al. Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics*, 19:477, 2018.
- Christina V Theodoris, Ling Xiao, Anant Chopra, Mark D Chaffin, Zeina R Al Sayed, Matthew C Hill, Helene Mantineo, Elizabeth M Brydon, Zexian Zeng, X Shirley Liu, et al. Transfer learning enables predictions in network biology. *Nature*, 618(7965):616–624, 2023.
- Feng Tian, Sen Lei, Yingbo Zhou, Jialin Cheng, Guohao Liang, Zhengxia Zou, Heng-Chao Li, and Zhenwei Shi. Hirennet: Hierarchical-relation network for few-shot remote sensing image scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 2024.
- Alex Tong and Xin Huang. Trajectorynet: Continuous modeling of cell trajectories with neural networks. *Proceedings of ISMB*, 2020.
- Cole Trapnell, Davide Cacchiarelli, James Grimsby, et al. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature Biotechnology*, 32(4): 381–386, 2014.
- Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 33:19667–19679, 2020.
- Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.
- Caleb Weinreb, Alejo Rodriguez-Fraticelli, Fernando D. Camargo, and Allon M. Klein. Lineage tracing on transcriptional landscapes links state to fate during differentiation. *Science*, 367(6479): eaaw3381, 2020a. doi: 10.1126/science.aaw3381. URL <https://www.science.org/doi/abs/10.1126/science.aaw3381>.
- Caleb Weinreb, Alejo Rodriguez-Fraticelli, Fernando D Camargo, and Allon M Klein. Lineage tracing on transcriptional landscapes links state to fate during differentiation. *Science*, 367(6479): eaaw3381, 2020b.
- Hongzhi Wen, Wenzhuo Tang, Xinnan Dai, Jiayuan Ding, Wei Jin, Yuying Xie, and Jiliang Tang. CellPLM: Pre-training of cell language model beyond single cells. In *ICLR*, 2024. URL <https://openreview.net/forum?id=BKXvPDekud>.
- Junxi Xiao and Qinliang Su. Treevi: Reparameterizable tree-structured variational inference for instance-level correlation capturing. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- Zelin Zang, Yuhao Wang, Jinlin Wu, Hong Liu, Yue Shen, Stan Li, Zhen Lei, et al. Dmt-hi: Moe-based hyperbolic interpretable deep manifold transformation for unsupervised dimensionality reduction. *arXiv preprint arXiv:2410.19504*, 2024a.
- Zelin Zang, Yongjie Xu, Chenrui Duan, Jinlin Wu, Stan Z Li, and Zhen Lei. A review of artificial intelligence based biological-tree construction: Priorities, methods, applications and trends. *arXiv preprint arXiv:2410.04815*, 2024b.

- Andy GX Zeng, Suraj Bansal, Liqing Jin, Amanda Mitchell, Weihsu Claire Chen, Hussein A Abbas, Michelle Chan-Seng-Yue, Veronique Voisin, Peter van Galen, Anne Tierens, et al. A cellular hierarchy framework for understanding heterogeneity and predicting drug response in acute myeloid leukemia. *Nature medicine*, 28(6):1212–1223, 2022.
- B. Zhang, P. He, J.E.G. Lawrence, et al. A human embryonic limb cell atlas resolved in space and time. *Nature*, 635:668–678, 2024. doi: 10.1038/s41586-023-06806-x.
- Bao Zhang, Peng He, John EG Lawrence, Shuaiyu Wang, Elizabeth Tuck, Brian A Williams, Kenny Roberts, Vitalii Kleshchevnikov, Lira Mamanova, Liam Bolt, et al. A human embryonic limb cell atlas resolved in space and time. *Nature*, pp. 1–11, 2023.
- Suyuan Zhao, Jiahuan Zhang, Yushuai Wu, YIZHEN LUO, and Zaiqing Nie. Langcell: Language-cell pre-training for cell identity understanding. In *Forty-first International Conference on Machine Learning*, 2024.
- Min Zhou, Menglin Yang, Bo Xiong, Hui Xiong, and Irwin King. Hyperbolic graph neural networks: A tutorial on methods and applications. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5843–5844, 2023.

A APPENDIX: DETAILS OF RELATED WORK

Tree-Structured Representation & Generative Models. Tree-structured representations are essential for modeling hierarchical relationships within data (Zang et al., 2024b). Traditional approaches, such as hierarchical clustering (Müllner, 2011) and distance-based methods (Kaufman & Rousseeuw, 2009; Bouguettaya et al., 2015), use predefined metrics to construct trees and have been foundational in many applications. Recent advancements in deep learning have introduced adaptive techniques for tree construction, such as the Nouveau VAE (NVAE)(Vahdat & Kautz, 2020), which captures hierarchical semantics through recursive structures, and the Tree Variational Autoencoder (Tree-VAE)(Manduchi et al., 2024), which encodes hierarchical latent structures in generative models. Furthermore, hyperbolic geometry has emerged as a powerful framework for representing hierarchical relationships, with methods like Poincaré Embeddings(Nickel & Kiela, 2017) and Hyperbolic Graph Neural Networks (HGNNs)(Zhou et al., 2023) offering efficient and expressive representations of such structures. TreeVI (Xiao & Su, 2024) extends variational inference by using a tree structure to efficiently capture correlations among latent variables in the posterior, enabling scalable reparameterization and training while improving performance in tasks like constrained clustering, user matching, and link prediction. Tree-based generative models offer powerful solutions for modeling hierarchical relationships and multi-modal data distributions. GAN-Tree(Kundu et al., 2019) introduces a hierarchical divisive strategy with a mode-splitting algorithm for unsupervised clustering, effectively addressing mode-collapse and discontinuities in data, while enabling incremental updates by modifying only specific tree branches.

Deep Learning Based Cell Lineage Analysis. Cell lineage analysis is a vital task in single-cell genomics, aiming to reconstruct developmental trajectories of cells. Traditional methods like *Monocle*(Trapnell et al., 2014) and *Slingshot*(Street et al., 2018) infer pseudotime trajectories but are limited by reliance on predefined metrics and inability to model unobserved progenitor states. Recent advances, such as *LineageVAE*(Majima et al., 2024) and *Waddington-OT*(Schiebinger et al., 2021), address these limitations using probabilistic models and optimal transport, yet often face challenges with the high dimensionality and sparsity of single-cell data. Generative models like *Diffusion Pseudotime Models*(Haghverdi et al., 2016) and *TrajectoryNet*(Tong & Huang, 2020) enhance scalability and interpretability but typically lack mechanisms to model hierarchical relationships in differentiation. Our proposed HDTree integrates hierarchical tree structures into the generative process, enabling accurate reconstruction of both observed and unobserved cell states while improving the interpretability of cell lineage trajectories.

B APPENDIX: ADDITIONAL ILLUSTRATION OF LATENT INDEXING AND OUTPUT TREE

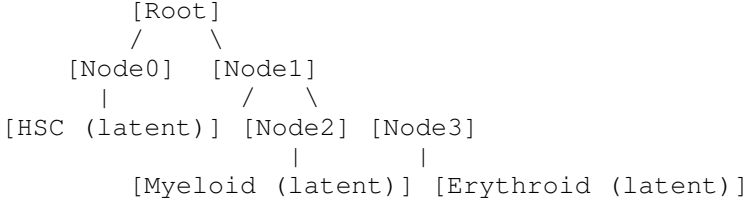
The binary structure of the hierarchical codebook $\mathcal{C}_W$ is used solely as an efficient indexing mechanism for latent representations and does not impose a binary topology on the generated tree. Each sample follows a unique path in the latent codebook, and shared prefixes across samples naturally form branching points. When aggregated over the entire dataset, these paths induce an n -ary and possibly unbalanced output tree that better reflects the underlying semantic hierarchy. Figure B.1 provides a textual illustration: the top panel shows binary latent paths labeled with their associated cell types, while the bottom panel shows how they merge into the expected hematopoietic tri-branch lineage (lymphoid, myeloid, erythroid).

C APPENDIX: SENSITIVITY ANALYSIS ON k IN THE AUXILIARY K-NN GRAPH

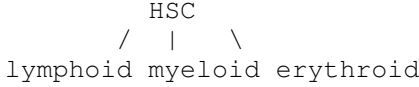
To verify that the auxiliary k-NN graph does not dominate the trajectory inference performance, we evaluate HDTree with different k values when constructing the within-level connectivity graph ($k \in \{5, 10, 20, 30\}$). Table C.1 reports the performance on the lineage inference task using the same metrics as in the main paper.

As shown in Table C.1, the performance varies only slightly ($<1\%$) when changing k from 5 to 30, indicating that the k-NN graph acts mainly as a local smoothing component while the global hierarchy is determined by the learned tree structure.

(a) Binary latent codebook (indexing only)



(b) Output lineage (multi-branch)



Mapping: paths ending at HSC(latent), Myeloid(latent), Erythroid(latent) map to the corresponding cell fates in the output tree.

Figure B.1: **Binary latent indexing vs. multi-branch output (textual illustration).** *Top:* the hierarchical codebook is a binary tree where leaf nodes store latent codewords associated with cell types (e.g., HSC, myeloid, erythroid). *Bottom:* when aggregating latent paths across samples, these leaves naturally form a multi-branch lineage tree that matches biological differentiation (e.g., hematopoietic tri-furcation). This clarifies that the binary structure is used only for latent indexing and does not constrain the number of output branches.

Table C.1: **Sensitivity of HDTree to the choice of k .** Results are averaged over three runs. The performance remains stable across a wide range of k , showing that the tree structure provides the main contribution.

k	5	10	20	30
ARI $\uparrow$	0.842	0.845	0.847	0.843
NMI $\uparrow$	0.792	0.794	0.795	0.791

D APPENDIX: ANY ROOTED TREE CAN BE REPRESENTED AS A BINARY TREE

We show that any finite rooted tree (optionally ordered) can be encoded as a binary tree via the *left-child/right-sibling (LCRS)* transformation, and that this encoding is bijective up to isomorphism. Hence a binary hierarchical codebook can index arbitrary n -ary branching structures without loss of information.

Setting. Let $\mathcal{T}$ be a finite rooted tree with node set V , root r , and (optional) left-to-right order among the children of each node. The standard *binary tree* has at most two pointers per node: *left child* and *right child*.

LCRS Encoding (Rooted Tree $\rightarrow$ Binary Tree). Construct a binary tree $\mathcal{B}$ on the same node set V by:

- For each node $u \in V$, if u has children $c_1, \dots, c_k$ in left-to-right order (possibly $k = 0$), set the **left child** of u in $\mathcal{B}$ to c_1 (or NULL if $k = 0$).
- For each sibling pair c_i, c_{i+1} ($1 \leq i < k$), set the **right child** of c_i in $\mathcal{B}$ to c_{i+1} ; if u has no $(i + 1)$ -th child, set the right child to NULL.

Intuitively: “left child = first child” and “right child = next sibling.”

Table E.2: Dataset-wise Hyperparameter Settings.

Dataset	ν (t-dist)	k (k-NN)	P	λ_{Recon}	λ_{SCL}	λ_{HQL}	Hierarchy Depth	LR
MNIST	0.2	5	1.0	1.0	0.5	1.0	10	0.005
FMNIST	0.5	5	1.0	1.0	0.5	1.0	10	0.005
20News	0.2	5	1.0	1.0	0.5	1.0	10	0.005
CIFAR-10	0.5	5	1.0	1.0	0.5	1.0	10	0.005
LIMB	0.5	5	1.0	1.0	0.5	1.0	10	0.005
LHCO	0.2	5	1.0	1.0	0.5	1.0	10	0.005
Weinreb	0.2	5	1.0	1.0	0.5	1.0	10	0.005

Decoding (Binary Tree $\rightarrow$ Rooted Tree). Given $\mathcal{B}$ produced by the above rules, recover $\mathcal{T}$ by:

1. For each node u , its (ordered) children in $\mathcal{T}$ are exactly the nodes reachable by starting at the *left child* of u in $\mathcal{B}$ and repeatedly following *right child* links ($c_1 = \text{left}(u)$, $c_{i+1} = \text{right}(c_i)$ until NULL).
2. The root is the unique node not appearing as a right child of any other node.

Theorem. For any finite rooted tree $\mathcal{T}$ (optionally ordered), the LCRS encoding produces a binary tree $\mathcal{B}$ such that decoding $\mathcal{B}$ recovers a tree isomorphic to $\mathcal{T}$. Moreover, the transformation preserves ancestor/descendant relations and sibling order.

Proof. (Well-defined & injective) By construction, every node has at most one left and one right child in $\mathcal{B}$; thus $\mathcal{B}$ is a binary tree. The children of u in $\mathcal{T}$ become a right-linked list starting at $\text{left}(u)$ in $\mathcal{B}$, preserving order. Different $\mathcal{T}$ yield different sets of right-linked lists, hence distinct $\mathcal{B}$ up to isomorphism.

(Surjective onto image & decoding correctness) The decoding procedure inverts the encoding by reading off, for each u , exactly the right-linked list rooted at $\text{left}(u)$ as the children sequence of u . Therefore $\text{decoding}(\text{encoding}(\mathcal{T}))$ returns a tree isomorphic to $\mathcal{T}$.

(Preserved relations) For any nodes u, v , u is an ancestor of v in $\mathcal{T}$ iff there exists a path in $\mathcal{B}$ that alternates: zero or more “right” edges within a sibling list to reach the correct child, followed by a “left” edge to descend to the first child at the next level, and so on. Hence ancestor/descendant and sibling order are preserved. $\square$

Complexity and Size. The transformation is linear time $O(|V|)$ and space $O(|V|)$. No dummy nodes are required; the node set is unchanged. Degree- k branching in $\mathcal{T}$ becomes a length- k right-sibling chain under the left child of the parent in $\mathcal{B}$.

What Is Preserved (and Not). The transformation preserves node identities, parent-child and ancestor relations, sibling order (if present), the number of nodes, and the lowest common ancestor (LCA) up to isomorphism. It does not preserve raw edge lengths or exact out-degree counts; the depth of a root-to-leaf path may shift by the number of right-sibling steps between successive left-child descents, though this mapping remains deterministic and fully reversible.

Consequence for Modeling. Using a binary hierarchical codebook for latent *indexing* (left = first child, right = next sibling) does *not* constrain the *output* branching factor: aggregating decoded paths over samples recovers arbitrary n -ary (possibly unbalanced) hierarchies. Thus a binary latent tree suffices to represent any rooted tree topology without loss of structural information.

E APPENDIX: DETAILS OF EXPERIMENTAL SETUP

Table E.2 summarizes the dataset-wise hyperparameter settings used in all experiments. All hyperparameters, including the weights in Eq. (8), are fixed across datasets except the t-distribution parameter ν , which is tuned via a small grid search $\nu \in \{0.05, 0.1, 0.2, 0.5\}$.

F APPENDIX: DETAILS OF DATASET

F.1 DATASETS

In this section, we provide an overview of the datasets used in our evaluation. We consider a diverse set of datasets, including image, text, and single-cell data, to assess the performance of hierarchical clustering methods across different domains. The datasets are selected based on their popularity, complexity, and relevance to real-world applications. We provide a brief description of each dataset, along with key statistics and preprocessing steps.

MNIST: A widely-used dataset consisting of 70,000 grayscale images of handwritten digits, each of size 28×28 . Each image is flattened into a 784-dimensional vector. This dataset is primarily used for benchmarking tree structure modeling and hierarchical clustering methods. More details can be found at ¹.

Fashion-MNIST: A dataset of 70,000 grayscale images representing 10 categories of clothing items, each with a resolution of 28×28 . The images are flattened into 784-dimensional vectors for analysis. This dataset is used to evaluate the robustness of methods on visual data with more complex patterns than MNIST. Dataset details are available at ².

20News-Groups: A text dataset with 18,846 newsgroup posts across 20 categories. The features are represented as TF-IDF vectors with dimensionality reduced to 2,000 features for computational feasibility. This dataset is employed to test methods on high-dimensional text data and hierarchical categorization tasks. More details can be found at ³.

CIFAR-10: A dataset of 60,000 color images spanning 10 classes, with each image sized at 32×32 . The pixel intensity values are flattened into a 3,072-dimensional vector. This dataset is utilized for evaluating hierarchical modeling methods on high-dimensional image data. Details are available at ⁴.

Limb: A single-cell RNA-seq dataset collected from limb bud development experiments. It contains approximately 10,000 cells with 20,000 genes per cell after preprocessing. This dataset is used to study developmental trajectories in biological processes. Dataset details are available in Zhang et al. (2024).

LHCO: Single-cell transcriptomics data derived from lung and heart cell ontogeny studies. The features represent gene expression profiles across 15,000 cells with dimensionality reduced to 10,000 genes. This dataset is used to explore differentiation patterns in complex organ systems. Dataset details can be found in He et al. (2022a).

Weinreb: A lineage-specific dataset focused on Darwinian lineage inference from single-cell data. It contains 8,000 cells with high-dimensional transcriptomic data preprocessed to 12,000 genes. This dataset is employed for benchmarking methods for lineage tracing tasks. Dataset details are provided in Weinreb et al. (2020a).

ECL: Single-cell data from embryonic cell lineages, designed for studying early developmental processes. It comprises 12,000 cells with 15,000 genes per cell after quality filtering. This dataset is used to test hierarchical modeling on datasets with complex lineage structures. Details are available in Qiu et al. (2024a).

The key characteristics of the datasets are summarized in Table F.3.

F.2 DATASET ORGANIZATION

In our experiment, the all dataset after download from our offered source link should be organized as follows: For Image data, the organization is:

¹<http://yann.lecun.com/exdb/mnist/>

²<https://github.com/zalandoresearch/fashion-mnist>

³<http://qwone.com/~jason/20Newsgroups/>

⁴<https://www.cs.toronto.edu/~kriz/cifar.html>

Table F.3: Summary of Dataset Statistics.

Dataset	Type	Class	Samples	Features	Source URL or Reference
MNIST	Image	10	70,000	784	http://yann.lecun.com/exdb/mnist/
Fashion-MNIST	Image	10	70,000	784	https://github.com/zalandoresearch/fashion-mnist
20News-Groups	Text	20	18,846	2,000	http://qwone.com/~jason/20Newsgroups/
CIFAR-10	Image	10	60,000	3,072	https://www.cs.toronto.edu/~kriz/cifar.html
Limb	Single-cell	10	66,633	500	https://limb-dev.cellgeni.sanger.ac.uk/
LHCO	Single-cell	7	10,628	500	https://www.ebi.ac.uk/biostudies/arrayexpress/studies/E-MTAB-10973
Weinreb	Single-cell	11	130,887	500	https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM4185642
ECL	Single-cell	10	838,000	500	https://cellxgene.cziscience.com/collections/45d5d2c3-bc28-4814-aed6-0bb6f0e11c82

Image Datasets Directory Structure

```

datasets/
|-- MNIST/
|   |-- train-images-idx3-ubyte
|   |-- train-labels-idx1-ubyte
|   |-- t10k-images-idx3-ubyte
|   \-- t10k-labels-idx1-ubyte
|-- FashionMNIST/
|   |-- train-images-idx3-ubyte
|   |-- train-labels-idx1-ubyte
|   |-- t10k-images-idx3-ubyte
|   \-- t10k-labels-idx1-ubyte
|-- 20news/
|   |-- alt.atheism/
|   |   |-- 12345.txt
|   |   |-- 67890.txt
|   |   \-- ...
|   |-- comp.graphics/
|   |   |-- 12346.txt
|   |   |-- 67891.txt
|   |   \-- ...
|   \-- ...
\-- cifar-10-batches-py/
    |-- data_batch_1
    |-- data_batch_2
    |-- data_batch_3
    |-- data_batch_4
    |-- data_batch_5
    |-- test_batch
    \-- batches.meta

```

For biology data, the organization is:

Biology Datasets Directory Structure

```

datasets_bio/
|- original/
| |- EpitheliaCell.h5ad
| |- LimbFilter.h5ad
| |- He_2022_NatureMethods_Day15.h5ad
| |- Weinreb_inVitro_clone_matrix.mtx
| |- Weinreb_inVitro_gene_names.txt
| |- Weinreb_inVitro_metadata.txt
| - Weinreb_inVitro_normed_counts.mtx
- processed/ (exists once the process is run)
|- EpitheliaCell_data_n.npy
|- EpitheliaCell_label.npy
|- LimbFilter_data_n.npy
|- LimbFilter_label.npy
|- LHCO.h5ad
- Weinreb.h5ad

```

F.3 PREPROCESSING

Before training, datasets need to be preprocessed. Preprocessing steps differ depending on the type of dataset. Below are the detailed guidelines:

F.3.1 IMAGE DATA

For image datasets (e.g., MNIST, FMINST), preprocessing is straightforward and can leverage the code from TreeVAE. The steps include: Initially, downloading and organizing the data ensures that all necessary dataset files, including images and corresponding labels, are retrieved from their sources and systematically placed into the appropriate directories within the project structure. This step is essential for maintaining data integrity and facilitating efficient access during subsequent processing stages. Next, converting raw data into NumPy arrays involves using provided scripts to load the raw image and label data. These scripts parse the binary or structured data formats and convert them into NumPy arrays, which are optimized for numerical computations in Python. This conversion facilitates further data manipulation and model training processes by providing a standardized format for handling large-scale datasets. Finally, normalization and formatting of the pixel values are performed. This typically includes scaling the pixel intensities to a range of [0, 1] to standardize the input features across different images. Additionally, any necessary adjustments are made to ensure the data format aligns with the requirements of the HDTree model, such as ensuring correct dimensionality and data type consistency. Proper normalization enhances model convergence and stability during training.

F.3.2 BIOLOGICAL DATA

For biological datasets (LHCO, Limb, Weinreb, ECL), preprocessing is more complex and tailored to each dataset. Below are the detailed preprocessing steps for each:

LHCO: Initially, data cleaning is performed to enhance the quality of the dataset. This includes the removal of duplicate entries and invalid samples that could introduce bias or noise into subsequent analyses. Additionally, strategies for handling missing values are implemented, which may involve imputation techniques or filtering out rows with an excessive amount of missing data. Following data cleaning, feature extraction is conducted to identify and extract relevant features from the raw data. For LHCO datasets, these features often include particle kinematics or event-level characteristics that are critical for analysis. Once extracted, feature scaling is applied to normalize or standardize the data, ensuring consistency across different scales and facilitating more efficient model training. Finally, the dataset is split into training, validation, and test sets according to predefined configurations.

This is the key code of our preprocessed methods:

```
1 adata = sc.read(f"{input_path}/He_2022_NatureMethods_Day15.h5ad")
```

```

1080 2 sc.pp.highly_variable_genes(adata, n_top_genes=500)
1081 3 adata = adata[:, adata.var['highly_variable']]
1082 4 data = adata.X
1083 5 data = adata.X.toarray()
1084 6 data = np.array(data).astype(np.float32)
1085 7 mean = data.mean(axis=0)
1086 8 std = data.std(axis=0)
1087 9 data = (data - mean) / std

```

Listing 1: Preprocess of Lhco

Limb: It begins with data loading, where the dataset is read from the provided files—typically in formats such as CSV or HDF5—into a structured representation suitable for further processing. Next, the data undergoes filtering and cleaning to enhance its quality and reliability. This includes the removal of noise or artifacts that may have been introduced during data acquisition, as well as deduplication to eliminate redundant entries. Missing values are addressed through appropriate strategies, such as imputation or selective removal of incomplete records. Following this, feature engineering is performed to transform raw biological signals into more interpretable and informative features. For instance, limb motion patterns or other domain-specific characteristics may be extracted to better capture the underlying structure of the data. These features are then normalized to ensure uniformity in scale, which is essential for many machine learning algorithms. Finally, the dataset is stratified and split into training, validation, and test subsets. This is the key code of our preprocessed methods:

```

1101 1 adata = sc.read(f"{input_path}/LimbFilter.h5ad")
1102 2 data_all = adata.X.toarray().astype(np.float32)
1103 3 label_celltype = adata.obs['celltype'].to_list()
1104 4 vars = np.var(data_all, axis=0) # HVG
1105 5 mask_gene = np.argsort(vars)[-500:]
1106 6 data_hvg = data_all[:, mask_gene]
1107 7 label_count = {}
1108 8 for i in list(set(label_celltype)):
1109 9     label_count[i] = label_celltype.count(i)
1110 10 label_count = sorted(label_count.items(), key=lambda x: x[1], reverse=
1111 11 True)
1112 11 label_count = label_count[:10]
1113 12 mask_top10 = np.zeros(len(label_celltype)).astype(np.bool_)
1114 13 for str_label in label_count:
1115 14     mask_top10[str_label[0]] = np.array(label_celltype) == 1
1116 15 data_n = np.array(data_hvg).astype(np.float32)[mask_top10]
1117 16 mean = data_n.mean(axis=0)
1118 17 std = data_n.std(axis=0)
1119 18 data = (data_n - mean) / std

```

Listing 2: Preprocess of Limb

Weinreb: Initially, data transformation is performed to normalize the raw gene expression counts. This typically includes log-transformation or conversion into normalized values such as Counts Per Million (CPM), Transcripts Per Million (TPM), or Fragments Per Kilobase of transcript per Million mapped reads (FPKM). Low-expression genes, as well as cells with insufficient sequencing depth or missing data, are filtered out to improve signal-to-noise ratio and computational efficiency. Following normalization, dimensionality reduction techniques—such as PCA—are applied to reduce the feature space while preserving the major sources of variation in the data, which can improve model performance and reduce computational burden. When the dataset originates from multiple experimental batches or sources, batch effect correction is employed to mitigate technical variability that could confound biological signal detection. Various statistical or machine learning-based approaches may be used depending on the nature of the data and experimental design. Finally, the dataset is stratified and partitioned into training, validation, and test sets.

This is the key code of our preprocessed methods:

```

1133 1 matrix_file = f"{input_path}Weinreb_inVitro_normed_counts.mtx"
1134 2 genes_file = f"{input_path}Weinreb_inVitro_gene_names.txt"

```

```

1134 3 metadata_file = f"{input_path}Weinreb_inVitro_metadata.txt"
1135 4 mtx = mmread(matrix_file).tocsr()
1136 5 genes = pd.read_csv(genes_file, header=None, names=['genes'])
1137 6 adata = sc.AnnData(mtx, var=genes)
1138 7 metadata = pd.read_csv(metadata_file, sep='\t')
1139 8 adata.obs = metadata.set_index(adata.obs.index)
1140 9 adata.write(f'{output_path}Weinreb.h5ad')
1141 10 sc.pp.loglp(adata)
1142 11 adata.obs['celltype']=adata.obs['Cell type annotation']
1143 12 adata = adata[~adata.obs['celltype'].isna()]
1144 13 sc.pp.highly_variable_genes(adata, n_top_genes=500)
1145 14 adata = adata[:, adata.var['highly_variable']]
1146 15 data = adata.X.toarray()
1147 16 data = np.array(data).astype(np.float32)
1148 17 mean = data.mean(axis=0)
1149 18 std = data.std(axis=0)
1150 19 data = (data - mean) / std

```

Listing 3: Preprocess of Weinreb

ECL: Initially, raw data files are parsed and converted into structured tabular formats that facilitate further computational processing. This is followed by a preprocessing stage that includes normalization—commonly achieved through z-score transformation or Min-Max scaling—and the handling of missing values, which may involve either imputation techniques or the removal of incomplete samples. Subsequently, feature selection is performed with an emphasis on retaining biologically meaningful attributes, often guided by domain-specific knowledge such as known molecular signatures. Finally, the dataset is partitioned into training, validation, and test subsets, ensuring that class distributions are preserved across splits to support unbiased model evaluation and generalization.

This is the key code of our preprocessed methods:

```

1161 1 adata = sc.read(f"{input_path}/EpitheliaCell.h5ad")
1162 2 adata.obs['celltype']=adata.obs['cell_type']
1163 3 label_celltype = adata.obs['celltype'].to_list()
1164 4 adata_sub = adata.copy()
1165 5 sc.pp.subsample(adata_sub, fraction=0.1)
1166 6 data_all = adata_sub.X.toarray().astype(np.float32)
1167 7 vars = np.var(data_all, axis=0)
1168 8 mask_gene = np.argsort(vars)[-500:]
1169 9 adata = adata[:, mask_gene]
1170 10 data = adata.X.toarray().astype(np.float32)
1171 11 label_count = {}
1172 12 for i in list(set(label_celltype)):
1173 13     label_count[i] = label_celltype.count(i)
1174 14 label_count = sorted(label_count.items(), key=lambda x: x[1], reverse=
1175 15     True)
1176 16 label_count = label_count[:10]
1177 17 mask_top10 = np.zeros(len(label_celltype)).astype(np.bool_)
1178 18 for str_label in label_count:
1179 19     mask_top10[str_label[0]] = np.array(label_celltype) == 1
1180 20 data_n = np.array(data).astype(np.float32)[mask_top10]
1181 21 label_train_str = np.array(list(np.squeeze(label_celltype)))[mask_top10]
1182 22 # downsample the 10k data for every cell type
1183 23 mask = np.zeros(len(label_train_str)).astype(np.bool_)
1184 24 for i in range(10):
1185 25     # random select 10k data for each cell type
1186 26     random_index = np.random.choice(
1187 27         np.where(label_train_str == label_count[i][0])[0],
1188 28         10000, replace=False)
1189 29     mask[random_index] = 1
1190 30 data_n = data_n[mask]
1191 31 mean = data_n.mean(axis=0)
1192 32 std = data_n.std(axis=0)
1193 33 data = (data_n - mean) / std

```

Listing 4: Preprocess of ECL

G APPENDIX: DETAILS OF BASELINE METHODS

We used the TreeVAE, CellPLM, LangCell, Geneformer as the reference methods. The detail use should follow:

1. **TreeVAE** You should clone the project "treevae" from <https://github.com/lauramanduchi/treevae.git>, and install necessary package followed by "minimal_requirements.txt".
2. **CellPLM** You should clone the project "CellPLM" from <https://github.com/OmicsML/CellPLM.git>, and install necessary package followed by "requirements.txt". The use of CellPLM is shown in the official tutorial in https://github.com/OmicsML/CellPLM/blob/main/tutorials/cell_embedding.ipynb
3. **LangCell** You should clone the project "LangCell" from [gitclonehttps://github.com/PharMolix/LangCell.git](https://github.com/PharMolix/LangCell.git), and install necessary package followed by "requirements.txt". Then you should install the "geneformer_001". The use of LangCell is shown in the official tutorial in <https://github.com/PharMolix/LangCell/blob/main/LangCell-annotation-zeroshot/zero-shot.ipynb>
4. **Geneformer** After you install LangCell, the Geneformer package has been installed. The use of Geneformer is shown in the official tutorial in https://github.com/jkobject/geneformer/blob/main/examples/extract_and_plot_cell_embeddings.ipynb

H APPENDIX: DETAILS OF TESTING PROTOCOL

To ensure a comprehensive and reproducible evaluation, we report clustering quality, hierarchical structure quality, and generative/reconstruction quality. Unless noted, metrics are computed on the test split and averaged over multiple runs (with fixed random seeds).

Clustering Accuracy (ACC). ACC is computed via an optimal one-to-one relabeling using the Hungarian algorithm. Let y_i be the ground-truth label and $\hat{y}_i$ the predicted cluster.

$$\text{ACC} = \frac{1}{n} \max_{\pi \in \mathcal{S}} \sum_{i=1}^n \mathbf{1}\{y_i = \pi(\hat{y}_i)\},$$

where $\mathcal{S}$ is the set of all label permutations. We use the standard Hungarian implementation to obtain π .

Normalized Mutual Information (NMI). Given predicted clustering C and ground-truth clustering G ,

$$\text{NMI}(C, G) = \frac{2 I(C; G)}{H(C) + H(G)},$$

where $I(\cdot; \cdot)$ is mutual information and $H(\cdot)$ is entropy. We use the symmetric NMI with natural logarithms. $\text{NMI} \in [0, 1]$ (higher is better).

Leaf Purity (LP). Let the learned tree T have leaf nodes $\{L_1, \dots, L_k\}$. For leaf L_i , define $L_i^y = \{x \in L_i : \text{label}(x) = y\}$. We report the macro-average over non-empty leaves:

$$\text{LP} = \frac{1}{k} \sum_{i=1}^k \frac{\max_y |L_i^y|}{|L_i|}.$$

Empty leaves (no assigned samples) are excluded from the average.

Dendrogram Purity (DP). Throughout the paper, DP denotes **Dendrogram Purity**, consistent with hierarchical clustering literature and prior work. For each class c , consider all pairs (i, j) with $y_i = y_j = c$. Let $\text{LCA}(i, j)$ be the lowest common ancestor cluster of x_i and x_j in the dendrogram, and let S_{ij} be the set of samples contained in that cluster. The pairwise purity is

$$\text{pur}(i, j) = \frac{|\{p \in S_{ij} : y_p = c\}|}{|S_{ij}|}.$$

DP is the average of $\text{pur}(i, j)$ over all intra-class pairs across all classes. This metric increases when intra-class pairs meet early in the tree (at purer LCA nodes).

Reconstruction Loss (RL). Given inputs $X = \{x_i\}$ and reconstructions $\hat{X} = \{\hat{x}_i\}$, we compute MSE:

$$\text{RL} = \frac{1}{n} \sum_{i=1}^n \|x_i - \hat{x}_i\|_2^2.$$

To align the “higher-is-better” convention across metrics, we report $-\text{RL}$ in tables (i.e., larger is better). Reconstructions for diffusion models are obtained by conditioning on the learned latent path and running the standard deterministic denoising trajectory at evaluation time.

Log-Likelihood (LL) for Diffusion Models. Exact log-likelihood is intractable for DDPMs; we report the negative ELBO (variational lower bound) following standard practice. Concretely, we sum the per-timestep KL (or reweighted MSE) terms under the chosen $\{\beta_t\}$ schedule and include the analytic prior and decoder terms as in Ho et al. (2020). We report the per-sample LL (higher is better). Implementation matches our training loss with the appropriate constants added back.

Fréchet Inception Distance (FID). *Images:* we compute FID in the 2048-D Inception-V3 pool3 feature space, matching the number of generated and real samples and using the same preprocessing. *Single-cell (scRNA-seq):* we compute FID in a biologically meaningful feature space: (i) select HVGs (e.g., top-1,000 by variance) on the training set; (ii) normalize real and generated matrices identically; (iii) optionally correct batch effects (e.g., Harmony/Scanorama) *before* feature extraction; (iv) run PCA to retain $> 90\%$ variance (typically ~ 50 PCs); (v) estimate Gaussians in the PC space and compute FID via covariance square roots (with a small diagonal regularizer if needed). We fix random seeds and average FID over multiple generations.

Ratio of Observed Time Points (ROP) for Lineage Consistency. For time-resolved single-cell datasets, we quantify local temporal coherence by measuring, for each cell, the fraction of its k -nearest neighbors (in the learned representation) whose time stamps are consistent with its developmental order; we then average over all cells and report by time window as in the main paper. Ablations show ROP strongly correlates (negatively) with tree-edit distance, supporting its biological relevance (Appendix §??).

Implementation Notes (All Metrics). (i) ACC relabeling uses the Hungarian algorithm; ties are broken deterministically. (ii) Empty leaves are excluded from LP; singleton leaves contribute 1.0. (iii) All metrics are averaged across r runs (defaults given in code) with fixed seeds and identical preprocessing. (iv) For diffusion metrics (RL/LL/FID), generation uses the same $\{\beta_t\}$ schedule and evaluation pipeline across methods.

I APPENDIX: DETAILS OF IMPLEMENTATION

For all experiments, the data is split into training, validation, and testing sets with an 8:1:1 ratio, ensuring unbiased evaluation. In testing, if the number of points in the dataset is greater than 10,000, we randomly sample 10,000 points from testing dataset. Details on downsampling and its rationale are provided in the Appendix. We implemented HDTree using PyTorch and trained the model on a single NVIDIA A100 GPU. The model is trained using the AdamW optimizer with a learning rate of $1e-4$ and a batch size of 128. The number of diffusion steps T is set to 1000, and the tree depth L is set to 10. The loss weights λ_{tree} and λ_{vq} were set to 1.0 and 0.25, respectively. The encoder and diffusion model are implemented using the multi-multilayer perceptron (MLP).

Table J.4: Time Efficiency Comparison Across Methods and Data Sizes: We conducted an empirical evaluation by selecting varying numbers of highly expressed channels to assess the computational time cost of our model under different data sizes. For this experiment, we set the output dimensionality of the Embedding method to 512 dimensions and employed UMAP for dimensionality reduction to facilitate visualization. The resulting features were then subjected to clustering analysis. As the dataset size increased from 10,000 to 100,000 samples, we observed a dramatic increase in the computational time required for both the dimensionality reduction and clustering processes. Notably, the clustering process escalated from a response time measured in minutes (ranging from 30 to 41 seconds) to one that took several hours (ranging from 2201 to 20429 seconds). Such prolonged processing times are impractical for model evaluation purposes. Consequently, we opted to downsample the data to 10,000 samples for further processing.

Metod	PCA	TSNE	UMAP	CellPLM	LangCell	GeneFormer
Cin=100, C=512, N=10000						
Embedding	None	None	820s	6s	13s	27s
DR	None	None	120s	129s	133s	150s
Cluster	None	None	41s	37s	30s	25s
Cin=1000, C=512, N=10000						
Embedding	140s	None	800s	7s	13s	25s
Dimentional Reduction	140s	None	120s	130s	130s	105s
Cluster	45s	None	40s	37s	34s	30s
Cin=1000, C=512, N=100000						
Embedding	224s	None	3153s	13s	11s	220s
Dimentional Reduction	413s	None	825s	416s	430s	420s
Cluster	9637s	None	7083s	5037s	2032s	3285s
Using K-means to 5000 clusters						
Embedding	221s	None	1834s	14s	123s	110s
Dimentional Reduction	458s	None	411s	350s	427s	410s
Cluster	2925s	None	20429s	5500s	3432s	2200s

J APPENDIX: DETAILS OF DOWNSAMPLING IN TESTING

As shown in Table J.4, the computational time required for encoding, dimensionality reduction (LowDim), and clustering significantly increases with larger data sizes. For instance, when the input size increases to N=100,000, the encoding time for UMAP rises from 820s to 3153s, and the clustering time for GeneFormer increases from 25s to 3285s. These trends are consistent across all tested methods.

Such exponential growth in computational overhead makes the evaluation process infeasible for large-scale datasets. To address this challenge, we uniformly downsample the data to 10,000 points for all methods during metric computation. This ensures consistent and fair comparisons while significantly improving testing efficiency. Additionally, the downsampling procedure preserves the overall data distribution and class proportions, ensuring that the evaluation results remain representative of the original dataset.

The detailed justification for selecting 10,000 points as the downsampling target is discussed in the Appendix.

K APPENDIX: DETAILS OF EXPERIMENTAL ENVIRONMENT

The experiments were conducted on a high-performance computing system with robust hardware and software configurations to ensure efficient handling of large datasets and complex computations. The hardware setup included NVIDIA A100 GPUs with 40GB memory for accelerated computation, Intel Xeon Platinum 8260 CPUs with 24 cores operating at 2.40GHz for efficient multi-threaded processing, 512GB of RAM, and 10TB of NVMe SSD storage for fast input/output operations.

The software environment was carefully configured for compatibility and reproducibility. The operating system used was Ubuntu 20.04 LTS (64-bit), and the primary deep learning framework was

Table L.5: **Compare on different sample size:** HDTree on different sample size of ECL. ECL-N k means we use ECL dataset and downsample it into N*1000 data point for training, while keeping the same test dataset.

	ECL-10k	ECL-20k	ECL-30k	ECL-50k	ECL-100k	ECL-200k	ECL-300k
ACC	82.2 $\pm$ 1.5	82.2 $\pm$ 1.5	82.5 $\pm$ 0.3	82.6 $\pm$ 0.5	82.9 $\pm$ 0.3	83.1 $\pm$ 0.4	83.1 $\pm$ 0.3
DP	67.0 $\pm$ 1.2	67.6 $\pm$ 1.5	68.3 $\pm$ 0.9	69.1 $\pm$ 0.7	68.5 $\pm$ 0.9	68.9 $\pm$ 0.8	67.0 $\pm$ 0.8

PyTorch (version 2.4.1), supplemented by TorchVision and Torchaudio extensions. The experiments were conducted using Python 3.11.6, with Conda (version 24.9.2) employed as the package manager to handle dependencies and virtual environments.

Key Python packages essential for the experiments included NumPy (1.26.4), SciPy (1.14.1), and pandas (2.2.3) for data preprocessing and analysis. Visualization tasks were performed using Matplotlib (3.9.2), Seaborn (0.13.2), and Plotly (5.24.1). For deep learning tasks, PyTorch (2.4.1) served as the primary frameworks. Dimensionality reduction and clustering were supported by UMAP-learn (0.5.6) and scikit-learn (1.5.2). Single-cell data analysis relied on specialized packages like Scanpy (1.10.3) and anndata (0.10.9).

A comprehensive list of installed Python packages is available upon request, capturing all dependencies required for reproducing the experiments. This configuration ensures the reported results are reproducible and highlights the environment’s compatibility with the experimental setup.

L APPENDIX: DETAILS OF EXPERIMENTS ON SINGLE CELL

L.1 DETAILED COMPARISONS ON MORE SINGLE CELL DATASETS.

We selected four single-cell datasets: LHCO, Limb, Weinreb, and ECL. For LHCO, Limb, and Weinreb, we used the full data. For ECL, to reduce computational cost in the metrics computation, we downsampled the testing dataset to 100,000 cells.

We evaluated both traditional (t-SNE, UMAP) and deep learning-based methods (Geneformer, LangCell, CellPLM, TreeVAE). For traditional models, we first selected 500 highly variable genes (HVGs), applied log_{1p} transformation where needed, and normalized the data using Z-score. For deep learning-based models, the preprocessing was handled automatically by their built-in pipelines. Thus, for fairness, we directly read the h5ad files, selected HVGs, and fed the results into the respective model pipelines to obtain cell embeddings.

After obtaining cell embeddings from all models, we performed unsupervised clustering using K-Means. Cluster assignments were compared with true cell type labels to compute clustering and tree-structure-based performance metrics. Due to the lack of established benchmarks in the single cell domain, we incorporated data from high-impact published studies into the TreeVAE benchmark. The results are shown in Table 2.

Analysis: (1) Similar to the general dataset, HDTree consistently demonstrates superior performance across all evaluated metrics, including tree structure quality, clustering accuracy, and hierarchical integrity. (2) We observed that the zero-shot capabilities of single-cell large language models are often unsatisfactory and, in some cases, fail to surpass basic single-cell methods. This conclusion has also been validated in recent studies (Lan et al., 2024; He et al., 2024). In comparison with foundational single-cell models, traditional single-cell tree analysis methods, and TreeVAE, HDTree shows relative advantages in performance and achieves better stability. (3) These results establish HDTree as a robust and reliable approach for single-cell data analysis.

L.2 STABILITY OF MODEL PERFORMANCE ACROSS VARYING TRAINING DATA SIZES

To verify that downsampling did not significantly affect model performance, we conducted additional experiments on subsampled versions of the ECL dataset at varying scales. The results are shown in Table L.5

Table L.6: **Comparison of tree performance, clustering performance on four single cell datasets.** Since most of the methods are not generative models, we did not compare generative performance.

Dataset	Method	Year	Tree Performance		Clustering Performance		Average($\uparrow$)
			DP($\uparrow$)	LP($\uparrow$)	ACC($\uparrow$)	NMI($\uparrow$)	
Limb (cell lineage, 66,633 points, celltype:10)	tSNE+Agg ^A	2014	34.9 $\pm$ 1.4	55.4 $\pm$ 1.1	48.9 $\pm$ 0.7	47.5 $\pm$ 1.0	46.7 $\pm$ 1.0
	UMAP+Agg ^A	2018	30.9 $\pm$ 2.0	50.1 $\pm$ 1.0	49.1 $\pm$ 1.0	41.4 $\pm$ 1.4	42.9 $\pm$ 1.4
	Geneformer ^A	2023	25.6 $\pm$ 5.4	35.9 $\pm$ 0.1	34.1 $\pm$ 0.1	34.9 $\pm$ 0.1	32.6 $\pm$ 1.4
	CellPLM ^A	2024	25.6 $\pm$ 0.1	39.9 $\pm$ 0.1	34.1 $\pm$ 0.2	32.9 $\pm$ 0.2	33.1 $\pm$ 0.2
	LangCell ^A	2024	25.3 $\pm$ 0.1	37.5 $\pm$ 0.1	33.9 $\pm$ 0.1	35.1 $\pm$ 0.1	33.0 $\pm$ 0.1
	TreeVAE ^A	2024	34.7 $\pm$ 1.7	55.6 $\pm$ 1.0	49.8 $\pm$ 0.1	50.0 $\pm$ 0.0	47.5 $\pm$ 0.7
	HDTree ^A	Ours	38.9$\pm$1.3	57.9$\pm$1.0	52.8$\pm$1.0	49.0$\pm$0.1	49.7$\pm$0.9
	HDTree	Ours	41.0$\pm$0.4	57.2$\pm$1.4	55.0$\pm$1.4	46.6$\pm$0.4	50.0$\pm$0.9 ($\uparrow$2.5)
LHCO (cell lineage, 10,628 points, celltype:7)	tSNE+Agg ^A	2014	37.4 $\pm$ 1.6	52.8 $\pm$ 0.8	43.6 $\pm$ 1.0	29.8 $\pm$ 0.5	40.9 $\pm$ 1.0
	UMAP+Agg ^A	2018	40.0 $\pm$ 1.4	50.6 $\pm$ 0.2	46.2 $\pm$ 0.2	34.2 $\pm$ 0.2	43.0 $\pm$ 0.5
	CellPLM ^A	2024	27.0 $\pm$ 1.1	35.8 $\pm$ 2.7	16.8 $\pm$ 3.4	1.65 $\pm$ 5.2	20.3 $\pm$ 3.1
	LangCell ^A	2024	26.5 $\pm$ 1.2	35.2 $\pm$ 0.8	35.2 $\pm$ 0.6	0.02 $\pm$ 0.9	24.2 $\pm$ 0.9
	TreeVAE ^A	2024	38.3 $\pm$ 2.0	52.2 $\pm$ 0.1	37.9 $\pm$ 0.1	31.6 $\pm$ 0.0	40.0 $\pm$ 0.6
	HDTree ^A	Ours	38.8$\pm$0.3	52.1$\pm$0.4	46.4$\pm$0.3	34.7$\pm$0.5	43.0$\pm$0.4
	HDTree	Ours	42.7$\pm$0.4	54.0$\pm$0.3	49.4$\pm$0.3	34.5$\pm$0.4	45.2$\pm$0.3 ($\uparrow$2.2)
Weinreb (cell lineage, 130,887 points, celltype:11)	tSNE+Agg ^A	2014	57.9 $\pm$ 1.5	63.3 $\pm$ 1.1	35.3 $\pm$ 1.0	38.5 $\pm$ 0.6	48.8 $\pm$ 1.1
	UMAP+Agg ^A	2018	51.8 $\pm$ 0.1	62.1 $\pm$ 0.8	47.2 $\pm$ 4.9	46.1 $\pm$ 1.2	51.8 $\pm$ 1.8
	LangCell ^A	2024	47.4 $\pm$ 0.1	54.8 $\pm$ 0.0	14.3 $\pm$ 0.5	34.3 $\pm$ 0.0	37.7 $\pm$ 0.2
	Geneformer ^A	2024	45.1 $\pm$ 0.4	55.3 $\pm$ 0.1	21.4 $\pm$ 0.1	32.3 $\pm$ 0.1	38.5 $\pm$ 0.2
	TreeVAE ^A	2024	60.4 $\pm$ 2.6	61.4 $\pm$ 0.5	41.0 $\pm$ 0.1	35.2 $\pm$ 0.0	49.5 $\pm$ 0.8
	HDTree ^A	Ours	63.3$\pm$2.6	78.2$\pm$1.1	50.6$\pm$1.0	45.2$\pm$1.2	59.3$\pm$1.5 ($\uparrow$7.5)
	HDTree	Ours	61.0$\pm$0.4	67.0$\pm$0.3	62.6$\pm$0.3	42.6$\pm$0.3	58.3$\pm$0.4
ECL (cell lineage, 838k points, celltype:10)	tSNE+Agg ^A	2014	55.5 $\pm$ 5.4	73.7 $\pm$ 4.2	73.1 $\pm$ 5.4	70.9 $\pm$ 3.8	68.3 $\pm$ 4.7
	UMAP+Agg ^A	2018	53.2 $\pm$ 1.4	73.6 $\pm$ 1.0	71.2 $\pm$ 1.5	71.8 $\pm$ 0.8	67.4 $\pm$
	TreeVAE ^A	2024	41.86 $\pm$ 1.9	60.7 $\pm$ 2.0	57.0 $\pm$ 3.0	61.8 $\pm$ 1.8	55.3 $\pm$ 2.2
	HDTree ^A	Ours	60.1$\pm$0.1	74.7$\pm$0.5	70.9$\pm$0.4	78.9$\pm$0.4	71.2$\pm$0.4
	HDTree	Ours	69.0$\pm$0.7	83.2$\pm$0.3	83.2$\pm$0.3	79.1$\pm$0.3	78.6$\pm$0.4 ($\uparrow$10.3)

The experimental results showed that the model performance did not change significantly with variations in the size of the training data. This indicates that the model’s performance remained relatively stable regardless of the dataset size, without notable improvements or declines. This finding suggests that the current amount of data is sufficient for the model to learn robust feature representations, or that data quantity is not the key factor influencing model performance in this task.