Rigel3D: Rig-aware Latents for Animation-Ready 3D Asset Generation

Nikitas Chatzis[1], Marios Loizou[1,2], Evangelos Kalogerakis[1,2,3]

[1]Technical University of Crete [2]CYENS Center of Excellence
[3]University of Massachusetts Amherst

Abstract

Recent 3D generative models can synthesize high-quality assets, but their outputs are typically static: they lack the skeletal rigs, joint hierarchies, and skinning weights required for animation. This limits their use in games, film, simulation, virtual agents, and embodied AI, where assets must not only look plausible but also move plausibly. We introduce Rigel3D, a generative method for animation-ready 3D assets represented as rigged meshes. Unlike post-hoc auto-rigging methods that attach rigs to completed shapes, our method jointly models geometry and rig structure through coupled surface and skeleton structured latent representations. A rig-aware autoencoder decodes these representations into mesh geometry, skeleton topology, joint coordinates, and skinning weights, while a two-stage latent generative model synthesizes both surface and skeleton representations for image-conditioned generation. To support downstream animation workflows, we further introduce an open-vocabulary joint labeling module that embeds generated joints into a shared vision-language space, enabling correspondence to arbitrary retargeting templates. Experiments on large-scale rigged asset datasets demonstrate that our method generates diverse, high-quality animation-ready assets and outperforms existing rigging baselines across multiple metrics.

This work is based on the Arxiv preprint *Rigel3D: Rig-aware Latents for Animation-Ready 3D Asset Generation* (Chatzis et al. [2026]). See `https://arxiv.org/abs/2605.13129`.

Keywords: Other, Vision and Learning

1 Introduction

Recent advances in 3D generative modeling have made it possible to synthesize increasingly detailed 3D assets from images, text, and learned latent distributions Zhang et al. [2023, 2024], Xiang et al. [2025b,a]. Yet most generated assets remain static: they lack the skeletal rigs, joint hierarchies, and skinning weights required by standard animation pipelines. Consequently, a generated mesh may look plausible but still require substantial post-processing before it can be posed, animated, or retargeted to existing motion data. Automatic rigging has long aimed to bridge this gap by predicting skeletons and skinning weights for input meshes. Classical and learning-based methods such as Pinocchio Baran and Popović [2007], RigNet Xu et al. [2020], TARig Ma and Zhang [2023], and recent template-free approaches Deng et al. [2025], Song et al. [2025b,a] have made significant progress in turning static shapes into animation-ready assets. However, most of these methods treat rigging as a post-processing step applied after shape generation. This separation is limiting for generative outputs, where geometry, topology, pose, and part structure may differ substantially from

† In accordance with the ERC Open Access mandate, the authors have made the Author Accepted Manuscript (AAM) publicly available under the Creative Commons Attribution (CC-BY 4.0) license.

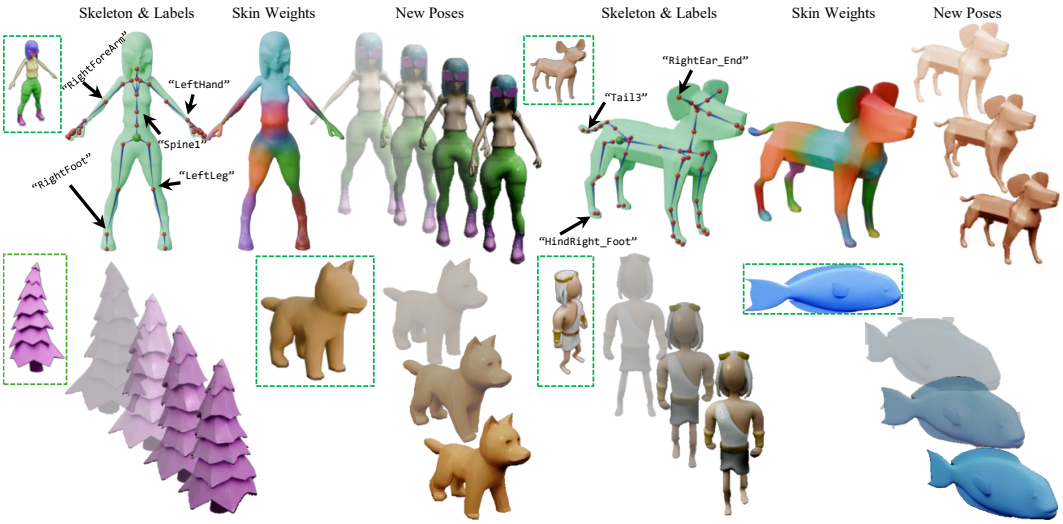

Figure 1: Given input images (in green boxes), Rigel3D generates diverse rigged 3D assets with meshes, textures, skeletons, labeled joints, and skinning weights (top row), making them directly usable in standard animation pipelines. Bottom row shows animated poses only.

the training distribution of a downstream rigger. Instead of first generating a static mesh and then attaching a rig, we seek to generate geometry and rigging together.

This joint generation problem is challenging because the relevant structures are heterogeneous but tightly coupled. Surface geometry is naturally represented by local features near the visible surface, while skeletons are sparse hierarchical structures that predominantly lie inside the object. A model must preserve consistency between these two domains, generate valid variable-size kinematic trees, and predict continuous skinning weights that bind surface vertices to bones. Another challenge is that practical animation workflows often require semantic joint labels for motion retargeting, but template-free generated skeletons typically provide only coordinates and connectivity. Thus, animation-ready asset generation requires not only high-quality shape synthesis, but also coherent skeleton generation, skinning prediction, and semantic compatibility with existing animation tools.

We introduce Rigel3D, a generative method for animation-ready 3D asset synthesis. Our method builds on TRELLIS' Structured LATent representation (SLATs) Xiang et al. [2025b], that represent 3D assets using sparse voxel locations and local latent codes attached to those voxels. We extend this surface-centric representation to rig-aware generation by learning two coupled structured latent representations: a surface SLat capturing geometry and appearance, and a skeleton SLat capturing articulation structure. A rig-aware autoencoder maps rigged meshes into these latent representations and decodes them into mesh geometry, skeleton topology, joint coordinates, and skinning weights. The decoder combines a skeleton-conditioned mesh decoder, an autoregressive skeleton decoder, and an attention-based skinning decoder, allowing shape and rig structure to inform one another. We then train a generative model over the rig-aware SLats. The generated latent representations are decoded into a mesh, optional appearance, an animation skeleton, and skinning weights, yielding complete rigged 3D assets rather than static ones. To support downstream motion retargeting, we also introduce an open-vocabulary joint labeling module that embeds generated joints into a shared vision-language space. Unlike closed-set classifiers tied to a fixed template, this module allows a generated skeleton to be matched to arbitrary candidate label sets. In summary, our contributions are:

- We introduce Rigel3D, a generative end-to-end framework for animation-ready 3D assets that produces meshes, optional appearance, skeletons, and skinning weights.
- We propose a rig-aware autoencoder with coupled surface and skeleton SLats, a skeleton-conditioned mesh decoder, an autoregressive skeleton decoder, and an attention-based skinning decoder, along with two-stage SLat generation for both surface and skeleton latent representations, enabling joint generation of geometry and rig structure.
- We introduce an open-vocabulary joint labeling module that supports motion retargeting by matching generated joints to text labels in a shared vision-language space.

- We demonstrate state-of-the-art performance over prior auto-rigging baselines across several metrics on two datasets: Anymate Deng et al. [2025] and ModelsResource Xu et al. [2020].

2 Related Work

3D and 4D generation. Recent 3D generative models synthesize high-quality static assets from images, text, or latent distributions using neural fields, Gaussian splats, vector-set representations, and sparse structured latents Mildenhall et al. [2020], Kerbl et al. [2023], Zhang et al. [2023, 2024], Xiang et al. [2025b]. TRELLIS Xiang et al. [2025b] introduced SLats for scalable two-stage 3D generation, with follow-up work improving compactness and fidelity Xiang et al. [2025a]. Other methods generate dynamic content or articulated objects Ren et al. [2023], Wu et al. [2025], Chen et al. [2025], Li et al. [2025], but typically represent motion, rigid-only articulation, or deformation sequences rather than complete skeletal rigs with skinning weights. Our method instead generates rigged meshes that can be controlled through standard skeletal animation pipelines.

Automatic rigging. Automatic rigging aims to convert a static 3D shape into an animation-ready asset by predicting an internal skeleton and skinning weights. Classical methods such as Pinocchio Baran and Popović [2007] embed template skeletons into input meshes, while early learning-based rigging methods such as volumetric skeleton prediction Xu et al. [2019] and RigNet Xu et al. [2020] predict skeleton structure from 3D geometry. Several works study humanoid, character-specific, or template-aware rigging: Neural Blend Shapes Li et al. [2021] assumes a prescribed skeleton structure and learns pose-dependent corrective deformations, NeuroSkinning Liu et al. [2019] predicts skinning weights for production characters with known skeletons, TARig Ma and Zhang [2023] performs template-aware humanoid rigging, and MoRig Xu et al. [2022] uses motion cues from point-cloud sequences to infer rigs. Recent humanoid-oriented systems such as HumanRig Chu et al. [2024], DRiVE Sun et al. [2025a], and Make-It-Animatable Guo et al. [2025b] produce animation-ready characters, but are primarily designed for humanoid inputs or post-hoc rigging of given shapes. CANOR He et al. [2025] takes a different approach by predicting editable control blobs instead of an explicit skeleton and skinning representation.

More recent template-free auto-rigging methods improve generality by modeling skeletons autoregressively or decomposing rigging into joint, connectivity, and skinning stages. Anymate Deng et al. [2025] introduces a large rigged-object dataset and strong modular baselines for joint prediction, connectivity, and skinning. MagicArticulate Song et al. [2025b] and Puppeteer Song et al. [2025a] use transformer-based skeleton generation conditioned on shape features, while UniRig Zhang et al. [2025], RigAnything Liu et al. [2025], ARMO Sun et al. [2025b], Auto-Connect Guo et al. [2025a] and Sun et al. [2026] explore autoregressive skeleton tokenization, connectivity modeling, and skinning prediction for diverse assets. These methods generally assume a completed input shape and infer a rig as a post-processing step. In contrast, our method jointly models the interdependent shape geometry and rig structure in an end-to-end generative framework.

Generative animation-ready assets. Closest to our work are methods that synthesize assets together with animation structure. SKDream Xu et al. [2025] conditions multiview and 3D generation on arbitrary skeletons, but assumes a skeleton input rather than generating a complete rig from scratch. AnimaX Huang et al. [2025], AnimaMimic Xie et al. [2025], Make-It-Poseable Guo et al. [2025c], and AnimateAnyMesh Wu et al. [2025] animate existing meshes or generate motion-conditioned deformations, but do not primarily focus on jointly generating a new mesh, skeleton, and skinning weights as a complete rigged asset. AnyTop Gat et al. [2025] generates motions for arbitrary skeleton topologies through textual joint descriptions. AniGen Huang et al. [2026] is concurrent work that directly generates animatable 3D assets by representing shape, skeleton, and skinning as continuous S^3 fields over a shared spatial domain. They produce a discrete skeleton through a post-processing clustering step. Our approach instead learns explicit surface and skeleton SLats, decodes skeleton topology autoregressively, and predicts skinning through point–bone attention in an end-to-end network. We also introduce open-vocabulary joint labeling for easier motion retargeting.

3 Method

Overview. Rigel3D builds on the Structured LATent representation of TRELLIS Xiang et al. [2025b], which represents 3D assets with sparse voxel locations and local latent codes. We extend this representation with two coupled latent sets: surface SLats for geometry and appearance, and

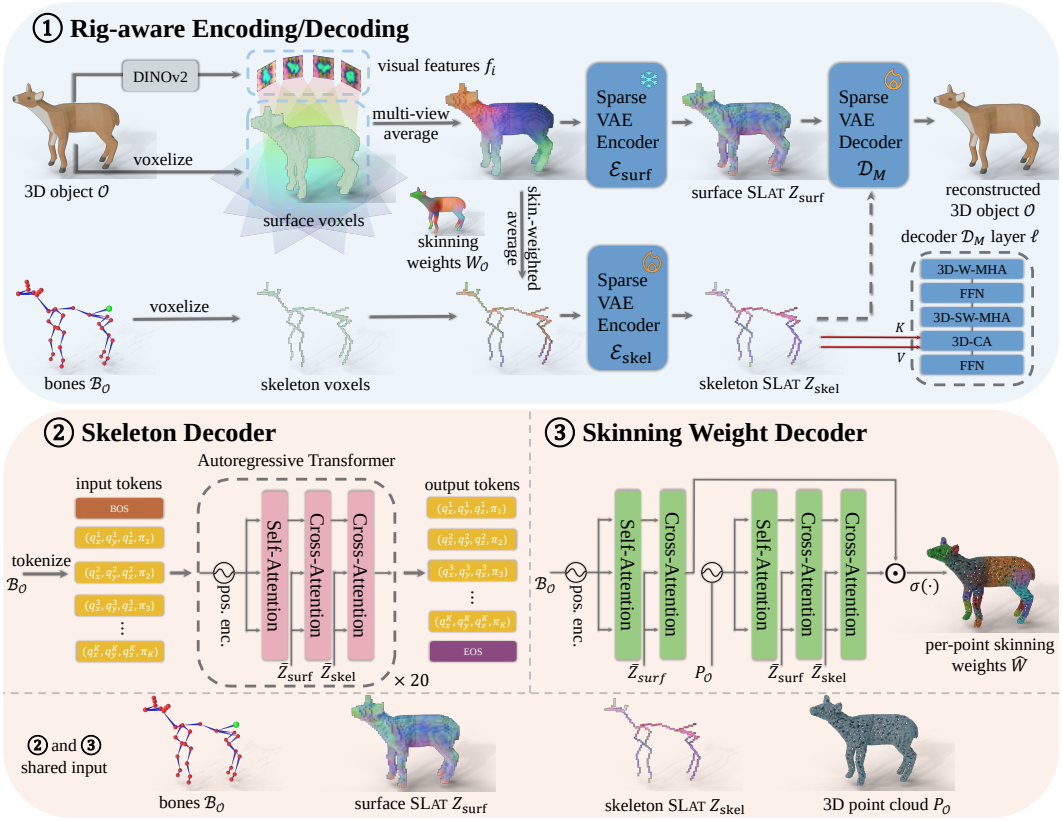

Figure 2: Overview of the rig-aware autoencoder. A surface encoder produces surface SLats from multiview visual features attached to occupied surface voxels, while a skeleton encoder produces skeleton SLats from rig-aware features attached to voxels intersecting the bones. The two latent representations are jointly decoded into mesh geometry, skeleton structure, and skinning weights.

skeleton SLats for internal articulation. A rig-aware autoencoder (Fig. 2) maps rigged meshes to these latents and decodes them into mesh geometry, skeleton, and skinning weights; a generative model then synthesizes both latent sets for image-conditioned rigged asset generation (Fig. 3).

3.1 Rig-Aware Autoencoder

Surface encoder. We adopt the TRELLIS surface encoder Xiang et al. [2025b]. The input mesh is voxelized at resolution 64^3, and voxels intersecting the surface are marked active. For each active voxel, we aggregate DINOv2 features Oquab et al. [2023] from multiview renderings, and process the resulting sparse feature grid with a sparse VAE encoder to obtain surface SLats:

$$\boldsymbol{Z}_{\text{surf}} = \{(\boldsymbol{c}_i, \boldsymbol{z}_i^{\text{surf}})\}_{i=1}^{L_{\text{surf}}}, \tag{1}$$

where $\boldsymbol{c}_i \in \{0, \ldots, 63\}^3$ is an active voxel coordinate and $\boldsymbol{z}_i^{\text{surf}}$ is the latent code attached to that voxel. These surface SLats provide a local representation of geometry and appearance.

Rig encoder. Each training asset contains a skeleton and skinning weights. We denote an asset by $\mathcal{O}$, its mesh vertices by $V_\mathcal{O}$, and its bones by $\mathcal{B}_\mathcal{O} = \{(\boldsymbol{j}_b^{\text{head}}, \boldsymbol{j}_b^{\text{tail}})\}_{b=1}^{|\mathcal{B}_\mathcal{O}|}$,, where $\boldsymbol{j}_b^{\text{head}}, \boldsymbol{j}_b^{\text{tail}} \in \mathbb{R}^3$ are the head and tail joint coordinates of bone b. The skinning weight matrix is $\boldsymbol{W}_\mathcal{O} \in \mathbb{R}^{|V_\mathcal{O}| \times |\mathcal{B}_\mathcal{O}|}$,, where $w_\mathcal{O}^{(v,b)}$ is the influence of bone b on vertex v.

To attach surface-aware context to the skeleton, we construct a feature for each bone by pooling the multiview DINOv2 features of its influenced vertices. Let $\boldsymbol{f}_v$ be the multiview feature associated with vertex v, obtained by projecting the vertex into the rendered views and averaging the corresponding DINOv2 features. For each bone b, let $V_b = \{v \in V_\mathcal{O} \mid w_\mathcal{O}^{(v,b)} > 0\}$ be the set of vertices influenced

by that bone. We define the bone feature as the skinning-weighted average:

$$\boldsymbol{f}_b = \frac{\sum_{v \in V_b} w_{\mathcal{O}}^{(v,b)} \boldsymbol{f}_v}{\sum_{v \in V_b} w_{\mathcal{O}}^{(v,b)} + \epsilon}, \tag{2}$$

where ϵ is a small constant for numerical stability.

We then rasterize each bone segment $(\boldsymbol{j}_b^{\text{head}}, \boldsymbol{j}_b^{\text{tail}})$ into the same 64^3 voxel grid and attach $\boldsymbol{f}_b$ to all voxels intersected by the segment, including the voxels containing the head and tail joints. If multiple bones intersect the same voxel, their features are averaged. Because skeleton voxels are sparse and mostly lie inside the mesh rather than on the surface, we process them with a separate sparse encoder $\mathcal{E}_{\text{skel}}$. This produces skeleton structured latents:

$$\boldsymbol{Z}_{\text{skel}} = \{(\boldsymbol{c}_i, \boldsymbol{z}_i^{\text{skel}})\}_{i=1}^{L_{\text{skel}}}, \tag{3}$$

which encode the internal articulation structure of the object.

Mesh decoder. Our mesh decoder extends the TRELLIS mesh decoder Xiang et al. [2025b]. Given surface SLats $\boldsymbol{Z}_{\text{surf}}$, the decoder predicts local FlexiCubes parameters Shen et al. [2023] for each active surface voxel and upsamples the sparse representation to a higher resolution before extracting the final mesh. Inspired by structure-aware reconstruction methods that use skeletal or medial information to improve surface reconstruction Petrov et al. [2024], we make geometry reconstruction aware of the underlying rig by augmenting each decoder block with a cross-attention layer from surface tokens to skeleton tokens. Let $\boldsymbol{H}_{\text{surf}}^{\ell}$ be the surface token features at decoder layer ℓ, and let $\boldsymbol{H}_{\text{skel}}$ be the skeleton SLats after projection, positional encoding, and decoder-side processing, used as keys and values in the cross-attention (CA) layer:

$$\widetilde{\boldsymbol{H}}_{\text{surf}}^{\ell} = \text{CA}\left(\boldsymbol{H}_{\text{surf}}^{\ell}, \boldsymbol{H}_{\text{skel}}, \boldsymbol{H}_{\text{skel}}\right), \tag{4}$$

followed by the standard TRELLIS decoder operations. This conditions surface reconstruction on the internal articulation structure. For each active surface voxel, the decoder predicts FlexiCubes parameters and signed distance values,

$$\mathcal{D}_M(\boldsymbol{Z}_{\text{surf}}, \boldsymbol{Z}_{\text{skel}}) = \{(\boldsymbol{\theta}_i, \boldsymbol{d}_i)\}_{i=1}^{L_{\text{surf}}}, \tag{5}$$

where $\boldsymbol{\theta}_i$ contains the local FlexiCubes parameters , and $\boldsymbol{d}_i$ denotes SDF values at voxel vertices. The final mesh is extracted from the predicted implicit field following TRELLIS and FlexiCubes.

Skeleton decoder. We decode the skeleton using an autoregressive transformer conditioned on both surface and skeleton SLats. Following recent auto-rigging methods Song et al. [2025a,b], we represent a skeleton as a sequence of discrete tokens encoding joint coordinates and parent connectivity. We first convert each skeleton into a breadth-first-search ordering starting from the root joint. Joints at the same depth are sorted deterministically by their spatial coordinates, using z-y-x order. Each joint token contains a discretized joint coordinate in a 128^3 grid and an index pointing to its parent joint. We quantize skeleton coordinates at a higher resolution than the SLat voxel grid to reduce joint localization error: $\boldsymbol{t}_k = (q_x^k, q_y^k, q_z^k, \pi_k)$, where (q_x^k, q_y^k, q_z^k) are discretized coordinates and π_k is the joint's parent index. For non-root joints, $\pi_k < k$; for the root joint, we set $\pi_k = k$. Unlike prior work that conditions skeleton generation on a single global shape feature, we condition on structured latent representations. We first project the surface and skeleton SLats to a common dimension D_s and add positional encodings based on their voxel coordinates: $\bar{\boldsymbol{z}}_i^{\text{surf}} = \phi_{\text{surf}}(\boldsymbol{z}_i^{\text{surf}}) + \gamma(\boldsymbol{c}_i)$, $\bar{\boldsymbol{z}}_i^{\text{skel}} = \phi_{\text{skel}}(\boldsymbol{z}_i^{\text{skel}}) + \gamma(\boldsymbol{c}_i)$, where ϕ_{surf} and ϕ_{skel} are learned linear projections and γ is a positional encoding function. At transformer layer ℓ, token features are updated by causal self-attention (CausalSA) followed by cross-attention to the surface and skeleton SLats:

$$\boldsymbol{h}_t^{\ell} = \text{CausalSA}\left(\boldsymbol{h}_t^{\ell-1}\right), \tilde{\boldsymbol{h}}_t^{\ell} = \text{CA}\left(\boldsymbol{h}_t^{\ell}, \bar{\boldsymbol{Z}}_{\text{surf}}, \bar{\boldsymbol{Z}}_{\text{surf}}\right), \boldsymbol{h}_t^{\ell+1} = \text{CA}\left(\tilde{\boldsymbol{h}}_t^{\ell}, \bar{\boldsymbol{Z}}_{\text{skel}}, \bar{\boldsymbol{Z}}_{\text{skel}}\right). \tag{6}$$

where $\bar{\boldsymbol{Z}}_{\text{surf}} = \{\bar{\boldsymbol{z}}_i^{\text{surf}}\}_{i=1}^{L_{\text{surf}}}$ and $\bar{\boldsymbol{Z}}_{\text{skel}} = \{\bar{\boldsymbol{z}}_i^{\text{skel}}\}_{i=1}^{L_{\text{skel}}}$ are the projected and positionally encoded SLat tokens used as keys and values in the cross-attention layers. The transformer outputs categorical distributions over coordinate tokens and valid parent indices. During training, the model receives the ground-truth token sequence and is optimized with teacher forcing. During inference, generation starts from a BOS token and proceeds until an EOS token is produced or a maximum joint count is reached. Parent logits are masked so that non-root token k can only select indices $< k$.

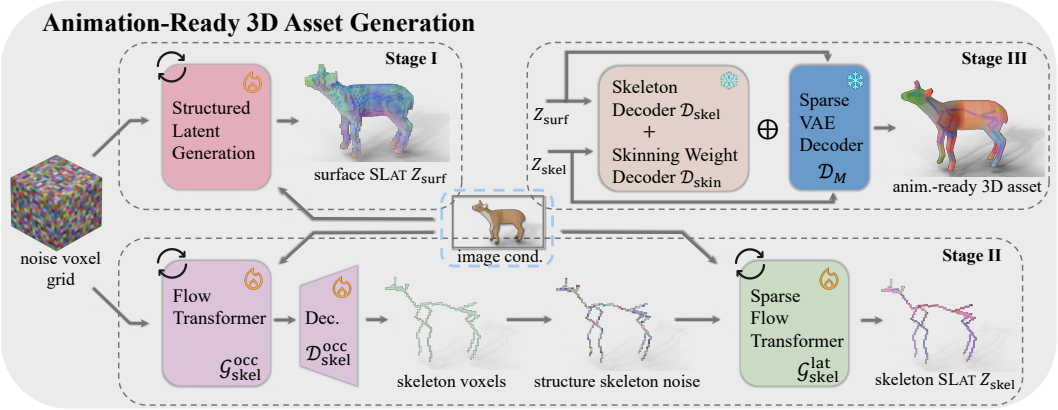

Figure 3: Animation-ready asset generation. Our generation pipeline yields both surface and skeleton SLats, which then are decoded into mesh geometry, skeleton structure, and skinning weights.

Skinning weight decoder. Given a generated or reconstructed mesh and skeleton, we predict skinning weights with an attention-based module. During training, the input consists of a fixed-size point cloud sampled from the mesh surface: $\boldsymbol{P} = \{(\boldsymbol{p}_i, \boldsymbol{n}_i)\}_{i=1}^{N}$, where $\boldsymbol{p}_i \in \mathbb{R}^3$ is a point and $\boldsymbol{n}_i \in \mathbb{R}^3$ is its normal, together with the decoded bones $\mathcal{B}_{\mathcal{O}}$ and the two latent representations $\boldsymbol{Z}_{\text{surf}}$ and $\boldsymbol{Z}_{\text{skel}}$. We embed points and bones into a shared feature dimension D_p. Each bone is represented by its head and tail coordinates: $\boldsymbol{h}^0_{\boldsymbol{p}_i} = \phi_P([\boldsymbol{p}_i, \boldsymbol{n}_i]) + \gamma(\boldsymbol{p}_i)$, $\boldsymbol{h}^0_b = \phi_B([\boldsymbol{j}^{\text{head}}_b, \boldsymbol{j}^{\text{tail}}_b]) + \gamma(\boldsymbol{j}^{\text{head}}_b) + \gamma(\boldsymbol{j}^{\text{tail}}_b)$. Point and bone tokens are refined through self-attention and cross-attention to surface and skeleton SLats.

The skinning logits are computed by point–bone similarity, then normalized with softmax:

$$a_{i,b} = \frac{\langle \psi_P(\tilde{\boldsymbol{h}}_{\boldsymbol{p}_i}), \psi_B(\boldsymbol{h}_b) \rangle}{\sqrt{D_p}}, \widehat{w}_{i,b} = \frac{\exp(a_{i,b})}{\sum_{b'=1}^{|\mathcal{B}_{\mathcal{O}}|} \exp(a_{i,b'})}. \tag{7}$$

where $\psi_P(\tilde{\boldsymbol{h}}_{\boldsymbol{p}_i})$ and $\psi_B(\boldsymbol{h}_b)$ are the output token embeddings corresponding to point i and bone b, respectively. At inference time, the same decoder can be evaluated on the vertices of the extracted mesh to obtain per-vertex skinning weights. Since attention and the softmax operate over the decoded bone tokens, the skinning decoder naturally handles skeletons with variable numbers of bones.

Autoencoder training. Training jointly supervises mesh reconstruction, skeleton reconstruction, skinning prediction, and latent regularization:

$$\mathcal{L}_{\text{AE}} = \mathcal{L}_{\text{mesh}} + \lambda_{\text{skel}} \mathcal{L}_{\text{skel}} + \lambda_{\text{skin}} \mathcal{L}_{\text{skin}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}. \tag{8}$$

Here $\mathcal{L}_{\text{mesh}}$ is the TRELLIS mesh reconstruction loss, $\mathcal{L}_{\text{skel}}$ is cross-entropy over skeleton coordinate and parent tokens, $\mathcal{L}_{\text{skin}}$ is soft cross-entropy over skinning distributions, and $\mathcal{L}_{\text{KL}}$ regularizes the surface and skeleton latent distributions.

We note that the surface encoder and mesh decoder are initialized from TRELLIS Xiang et al. [2025b], then fine-tuned during the training of our autoencoder.

3.2 Animation-Ready Asset Generation

We generate rigged assets by learning generative models over the latent representations produced by the rig-aware autoencoder. We follow the two-stage generation strategy of TRELLIS Xiang et al. [2025b], which first generates the sparse voxel structure and then generates the local latent codes attached to the active voxels. In our setting, we apply this two-stage process to the surface and skeleton SLats. For each latent type $r \in \{\text{surf}, \text{skel}\}$, we represent the active voxel set as a binary occupancy grid $\boldsymbol{O}_r \in \{0,1\}^{64\times64\times64}$. A 3D convolutional VAE compresses this grid into a low-resolution continuous feature grid $\boldsymbol{y}_r \in \mathbb{R}^{16\times16\times16\times C_r}$. A rectified-flow transformer $\mathcal{G}^{\text{occ}}_r$ learns to generate $\boldsymbol{y}_r$ conditioned on image features extracted from input renders. The VAE decoder then maps $\boldsymbol{y}_r$ back to a 64^3 active voxel grid. Given the generated active voxels, a second rectified-flow transformer $\mathcal{G}^{\text{lat}}_r$ generates the latent codes attached to those voxels, $\boldsymbol{Z}_r = \mathcal{G}^{\text{lat}}_r(\boldsymbol{O}_r, \boldsymbol{I})$, where $\boldsymbol{I}$ denotes the image condition. As in TRELLIS, image features are injected into the flow transformers through cross-attention. In total, our generative model contains four flow models: occupancy and

latent generators for the surface SLats, and occupancy and latent generators for the skeleton SLats. The latents $\boldsymbol{Z}_{\text{surf}}, \boldsymbol{Z}_{\text{skel}}$ are passed to the mesh decoder, skeleton decoder, and skinning decoder:

$$\widehat{M} = \mathcal{D}_M(\boldsymbol{Z}_{\text{surf}}, \boldsymbol{Z}_{\text{skel}}), \widehat{\mathcal{B}} = \mathcal{D}_{\text{skel}}(\boldsymbol{Z}_{\text{surf}}, \boldsymbol{Z}_{\text{skel}}), \widehat{\boldsymbol{W}} = \mathcal{D}_{\text{skin}}(\widehat{M}, \widehat{\mathcal{B}}, \boldsymbol{Z}_{\text{surf}}, \boldsymbol{Z}_{\text{skel}}). \tag{9}$$

4 Open-Vocabulary Joint Label Assignment

Many animation pipelines require semantic joint names to establish correspondences between source and target skeletons for motion retargeting. However, template-free skeleton generation methods typically output only joint coordinates and connectivity, or generic identifiers such as `Joint1` and `Bone003`, which do not encode anatomical, semantic, or functional correspondence. A closed-set classifier would restrict generated rigs to a predefined label vocabulary or skeleton template, which is undesirable for assets with non-standard parts, different naming conventions, or object-specific articulations. We therefore use an open-vocabulary formulation: generated joints are embedded into a shared vision-language space and can be queried using labels from arbitrary templates.

Given a generated skeleton and the corresponding surface and skeleton SLats, our labeling module predicts a normalized embedding $\boldsymbol{e}_{J_k}$ for each joint k in the embedding space of a frozen OpenCLIP text encoder Cherti et al. [2023]. The module conditions on joint coordinates, skeleton hierarchy, and the global SLat context. Since joint labels are highly dependent along the kinematic tree, we use an autoregressive transformer following the BFS ordering of the skeleton decoder: each joint embedding attends to the surface and skeleton SLats, causally attends to previously generated joints, and cross-attends to embeddings of previous labels. This autoregressive variant performs best in our experiments and helps disambiguate repeated or symmetric structures such as left/right limbs, fingers, fins, and tails. We train the joint embedding model using cleaned joint labels from Anymate Deng et al. [2025]. Because artist-provided labels are heterogeneous and often contain armature prefixes, namespaces, duplicated identifiers, or uninformative strings, we preprocess them with an LLM-based cleanup pipeline using QWEN3-8B Yang et al. [2025]; details and examples are provided in the appendix. For each cleaned label ℓ_k, we compute a normalized text embedding $\boldsymbol{e}_{\ell_k} = \text{CLIP}_{\text{text}}(\ell_k)$ and optimize an InfoNCE objective over joint-label pairs:

$$\mathcal{L}_{\text{label}} = -\frac{1}{B}\sum_{k=1}^{B} \log \frac{\exp(\boldsymbol{e}_{J_k}^{\top} \boldsymbol{e}_{\ell_k}/\tau)}{\sum_{m=1}^{B} \exp(\boldsymbol{e}_{J_k}^{\top} \boldsymbol{e}_{\ell_m}/\tau)}, \tag{10}$$

where τ is a temperature. Other labels in the minibatch serve as negatives. At inference time, each generated joint is labeled by nearest-neighbor retrieval over any candidate vocabulary, including joint names of a retargeting template. Thus, the learned joint representation is decoupled from the choice of label set, allowing the same rig to be matched to different animation templates without retraining.

5 Experiments

We now discuss our training and evaluation protocol, along with comparisons, results, and ablations.

Training dataset. We train Rigel3D on the Anymate dataset Deng et al. [2025], the largest publicly available dataset of rigged 3D assets. We use its training split, which contains approximately 225K rigged models with mesh geometry, skeletons, and skinning weights. To prepare the images used for the extraction of the multi-view Dino features as well as the rig-features, we follow the rendering procedure of TRELLIS Xiang et al. [2025b].

Competing methods. We compare against two representative auto-rigging baselines, Anymate Deng et al. [2025] and Puppeteer Song et al. [2025a], as well as the *concurrent* image-to-rigged-asset generation method AniGen Huang et al. [2026]. Anymate and Puppeteer assume an input mesh rather than generating one. For a fair comparison in the image-conditioned setting, we provide both methods with meshes generated by TRELLIS Xiang et al. [2025b] finetuned on the Anymate training split, matching the training data used by Rigel3D. For Anymate, we use the publicly released checkpoints trained on the same dataset as ours. For Puppeteer, we train the model from scratch on our training data, since the released checkpoints are trained on different datasets. For AniGen, we use the publicly available checkpoints provided by the authors.

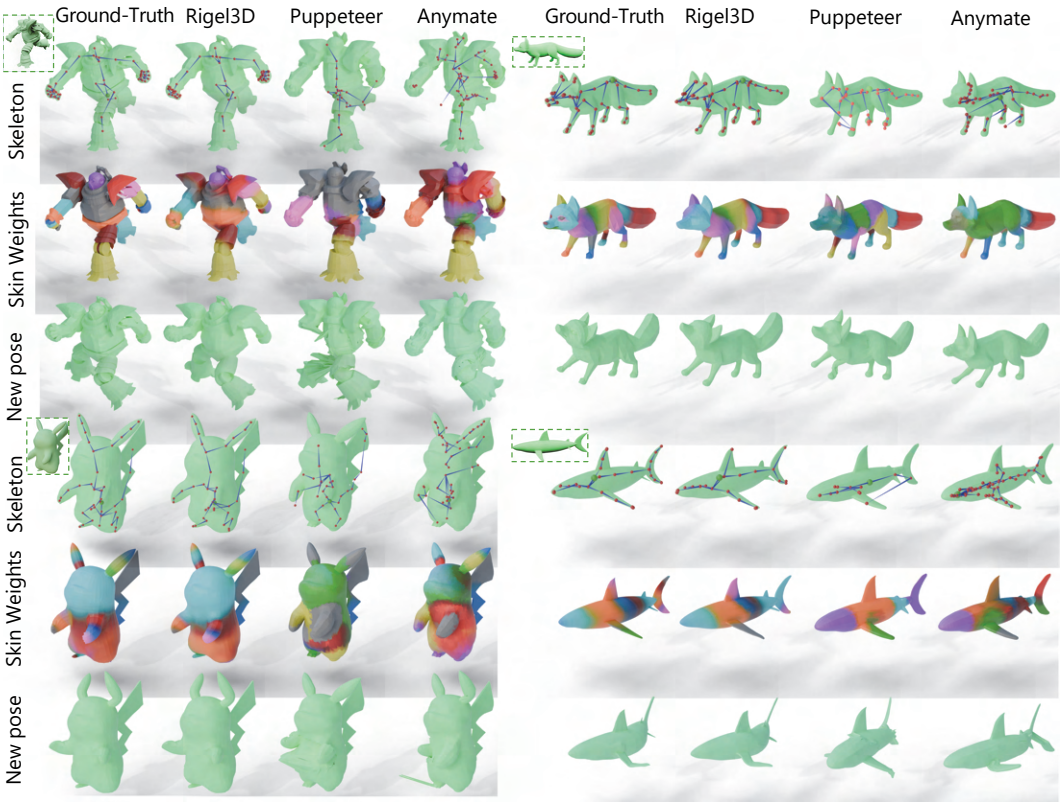

Figure 4: **Qualitative comparison with Anymate and Puppeteer.** Compared to auto-rigging baselines, Rigel3D produces rigs that better match the reference ones in joint placement, connectivity, and skinning while yielding more coherent novel poses. Shapes are shown without texture to emphasize skeleton and pose geometry rather than appearance. Skinning colors indicate bone influence, with smooth transitions corresponding to blended weights. Green insets show input images.

Evaluation protocol. We evaluate on the Anymate test split, containing approximately 5.6K assets Deng et al. [2025], and on the ModelsResource test split Xu et al. [2020], containing 270 diverse rigged assets. For each asset, we render 4 views and use the rendered images as conditioning inputs. Our rendering setup and camera parameters follow TRELLIS Xiang et al. [2025b]. We report metrics averaged over all rendered views and all test assets. For comparisons with AniGen Huang et al. [2026], we evaluate on the ArticulationXL test split used in their work, and the ModelsResource test split. We note that our ArticulationXL numbers differ from those reported by AniGen because, at the time of submission, their test images, rendering parameters, and evaluation code were not publicly available. We thus evaluate AniGen checkpoints using our rendering setup and protocol.

Evaluation metrics. We use the standard skeleton and skinning metrics introduced in RigNet Xu et al. [2020] and Anymate Deng et al. [2025]. For skeleton evaluation, *J2J* measures the symmetric Chamfer distance between predicted and reference joint sets. *J2B* measures the average of two distances: from predicted joints to the nearest point on the reference bones, and from reference joints to the nearest point on the predicted bones. *B2B* measures the Chamfer distance between predicted and reference bones, treated as line segments. Note that all geometric distances are computed after normalizing each test asset to a unit bounding box. For skinning evaluation, we report the per-vertex ℓ_1 and ℓ_2 distances between predicted and reference skinning weight vectors. In addition, we report the the KL divergence measuring how much the predicted skinning weight distribution differs from the reference skinning weight distribution. Since the generated meshes do not share vertex correspondence with the reference mesh, we follow Anigen Huang et al. [2026] and align predicted skinning weights to the reference skeleton using optimal transport induced by the Wasserstein distance between predicted and reference joints. We report per-point ℓ_1, ℓ_2, and KL divergence between the aligned predicted skinning distributions and the reference skinning weights. This avoids assuming a fixed joint ordering and amount or shared mesh topology across methods.

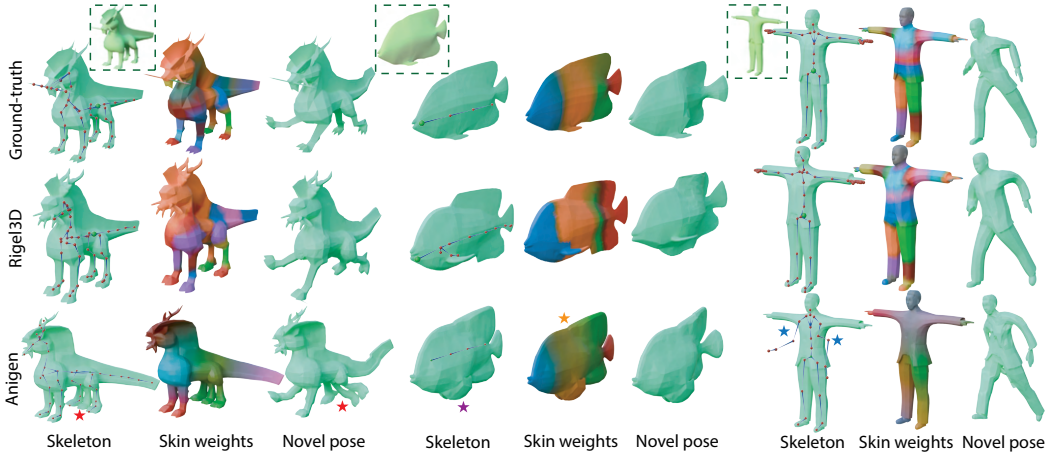

Figure 5: **Qualitative comparison with AniGen.** Here we compare generated shapes, skeletons, skinning, and novel-pose deformations. Green insets show input images. Rigel3D better preserves the articulation structure and produces more coherent skinning weights across diverse assets. Stars highlight issues: ⋆ an extra generated spurious limb, ⋆ incomplete fin geometry, ⋆ skinning weights that differ substantially from the reference distribution, and ⋆ erroneous joint placement.

Table 1: Image-conditioned rig generation comparisons (skeleton distances are reported $\times 100$).

	Anymate						ModelsResource					
	J2J↓	J2B↓	B2B↓	ℓ_1↓	ℓ_2↓	KL↓	J2J↓	J2B↓	B2B↓	ℓ_1↓	ℓ_2↓	KL↓
Anymate	7.834	5.964	5.363	0.0449	0.0180	2.589	5.834	**4.184**	4.322	0.0661	0.0285	2.215
Puppeteer	8.645	6.739	5.710	0.0457	0.0183	2.611	6.797	5.049	4.808	0.0673	0.0294	2.249
Ours	**7.364**	**5.736**	**4.902**	**0.0440**	**0.0176**	**2.478**	**5.765**	4.260	**4.200**	**0.0654**	**0.0283**	**2.155**

Table 2: Comparisons with Anigen (skeleton distances are reported $\times 100$).

	Articulation XL						ModelsResource					
	J2J↓	J2B↓	B2B↓	ℓ_1↓	ℓ_2↓	KL↓	J2J↓	J2B↓	B2B↓	ℓ_1↓	ℓ_2↓	KL↓
AniGen	10.667	9.715	8.601	**0.0609**	**0.0249**	**2.293**	9.730	7.740	6.852	0.0669	0.0293	2.330
Ours	**7.168**	**6.036**	**5.976**	0.0637	0.0267	2.320	**5.765**	**4.260**	**4.200**	**0.0654**	**0.0283**	**2.155**

Comparisons with prior auto-rigging methods. Table 1 compares Rigel3D against Anymate and Puppeteer on the Anymate and ModelsResource test splits. Both baselines are post-hoc auto-rigging methods that require an input mesh; in this image-conditioned setting, we provide them with meshes generated by the same TRELLIS model used in our pipeline. On Anymate, Rigel3D outperforms both baselines across all reported skeleton and skinning metrics. On ModelsResource, Rigel3D obtains the best score on J2J, B2B, and all skinning metrics, while Anymate is slightly better on J2B. The gains are most pronounced for B2B, suggesting that our skeletons better capture bone-level structure and connectivity, and for KL, indicating improved alignment of the predicted skinning distributions. Overall, these results support our design choice of jointly modeling surface geometry and rig structure, rather than applying rigging as a post-processing step to a generated mesh. Qualitative comparisons in Figure 4 further illustrate that Rigel3D produces more accurate joint placement, bone connectivity, and skinning weight distributions on complex shapes.

Quantitative comparisons with AniGen. Table 2 compares Rigel3D with the concurrent image-to-rigged-asset method AniGen Huang et al. [2026] on ArticulationXL and ModelsResource. On ArticulationXL, Rigel3D substantially improves all skeleton metrics over AniGen, reducing J2J from 10.667 to 7.168 (32.8% relative reduction), J2B from 9.715 to 6.036 (37.9%), and B2B from 8.601 to 5.976 (30.5%). AniGen obtains slightly lower skinning errors on this split, suggesting that its continuous field representation can produce competitive skinning distributions once the skeleton is aligned. On ModelsResource, Rigel3D outperforms AniGen across all reported skeleton and skinning metrics, with especially large gains in skeleton accuracy. These results indicate that explicitly modeling skeleton structure with skeleton SLats and autoregressive topology decoding improves the quality of generated rigs, while remaining competitive on skinning prediction. Qualitative comparisons in Fig. 5 further illustrates that Rigel3D better preserves shape structure, places joints more consistently, and produces skinning weights that more closely match the reference distribution.

Joint labeling. Fig. 1 illustrates open-vocabulary joint labeling on representative generated characters. The predicted labels identify semantically meaningful joints across different body structures, enabling correspondences to downstream motion-retargeting templates without assuming a fixed skeleton vocabulary (please see the supplement for motion retargeting examples). We provide a detailed comparison of alternative joint-labeling strategies in the Appendix.

Ablation studies and additional comparisons. We provide ablation studies in the Appendix analyzing the contribution of our main architectural choices, including the benefit of factorizing the latent space into distinct surface and skeleton SLats and coupling them during decoding. We also compare the shape generation quality of Rigel3D against TRELLIS Xiang et al. [2025b].

6 Discussion and Limitations

We introduced Rigel3D, a generative framework for producing animation-ready 3D assets with geometry, skeletons, skinning weights, and joint labels. By jointly modeling surface and skeleton SLats, our method narrows the gap between static 3D generation and rig-based animation workflows.

Limitations. Generated rigs may still contain missing joints, spurious branches, or imperfect connectivity. Because skinning is not supervised by long motion sequences, deformations can be less natural under extreme poses or specific animation intents. Open-vocabulary labels may be ambiguous for repeated structures such as fingers, tails, or decorative appendages. Finally, our method inherits limitations of image-conditioned 3D generation when input views are ambiguous or occluded.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the Horizon Research and Innovation Programme (Grant agreement No. 101124742).

References

Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM TOG*, 26(3), 2007.

Nikitas Chatzis, Marios Loizou, and Evangelos Kalogerakis. Rigel3d: Rig-aware latents for animation-ready 3d asset generation, 2026. URL `https://arxiv.org/abs/2605.13129`.

Honghua Chen, Yushi Lan, Yongwei Chen, and Xingang Pan. ArtiLatent: Realistic Articulated 3D Object Generation via Structured Latents. In *SIGGRAPH Asia*, 2025.

Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. In *CVPR*, 2023.

Zedong Chu, Feng Xiong, Meiduo Liu, Jinzhi Zhang, Mingqi Shao, Zhaoxu Sun, Di Wang, and Mu Xu. HumanRig: Learning Automatic Rigging for Humanoid Character in a Large Scale Dataset, 2024.

Yufan Deng, Yuhao Zhang, Chen Geng, Shangzhe Wu, and Jiajun Wu. Anymate: A Dataset and Baselines for Learning 3D Object Rigging. In *SIGGRAPH*, 2025.

Inbar Gat, Sigal Raab, Guy Tevet, Yuval Reshef, Amit H. Bermano, and Daniel Cohen-Or. AnyTop: Character Animation Diffusion with Any Topology. *arXiv preprint arXiv:2502.17327*, 2025.

Jingfeng Guo, Jian Liu, Jinnan Chen, Shiwei Mao, Changrong Hu, Puhua Jiang, Junlin Yu, Jing Xu, Qi Liu, Lixin Xu, Zhuo Chen, and Chunchao Guo. Auto-Connect: Connectivity-Preserving RigFormer with Direct Preference Optimization. *arXiv preprint arXiv:2506.11430*, 2025a.

Zhiyang Guo, Jinxu Xiang, Kai Ma, Wengang Zhou, Houqiang Li, and Ran Zhang. Make-It-Animatable: An Efficient Framework for Authoring Animation-Ready 3D Characters. In *CVPR*, 2025b.

Zhiyang Guo, Jinxu Xiang, Kai Ma, Wengang Zhou, Houqiang Li, and Ran Zhang. Make-It-Poseable: Feed-forward Latent Posing Model for 3D Humanoid Characters. *arXiv preprint arXiv:2512.16767*, 2025c.

Guangzhao He, Chen Geng, Shangzhe Wu, and Jiajun Wu. Category-Agnostic Neural Object Rigging. *arXiv preprint arXiv:2505.20283*, 2025.

Yi-Hua Huang, Zi-Xin Zhou, Yuting He, Chirui Chang, Cheng-Feng Pu, Ziyi Yang, Yuan-Chen Guo, Yan-Pei Cao, and Xiaojuan Qi. AniGen: Unified S^3 Fields for Animatable 3D Asset Generation. *SIGGRAPH (to appear), arxiv version DOI: 2604.08746 (14 Apr 2026)*, 2026.

Zehuan Huang, Haoran Feng, Yangtian Sun, Yuanchen Guo, Yanpei Cao, and Lu Sheng. AnimaX: Animating the Inanimate in 3D with Joint Video-Pose Diffusion Models. *arXiv preprint arXiv:2506.19851*, 2025.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM TOG*, 42(4), 2023.

Peizhuo Li, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. Learning Skeletal Articulations with Neural Blend Shapes. *ACM TOG*, 40(4), 2021.

Ruining Li, Yuxin Yao, Chuanxia Zheng, Christian Rupprecht, Joan Lasenby, Shangzhe Wu, and Andrea Vedaldi. Particulate: Feed-Forward 3D Object Articulation. *arXiv preprint arXiv:2512.11798*, 2025.

Isabella Liu, Zhan Xu, Yifan Wang, Hao Tan, Zexiang Xu, Xiaolong Wang, Hao Su, and Zifan Shi. Riganything: Template-free autoregressive rigging for diverse 3d assets. *ACM TOG*, 44, 2025.

Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. NeuroSkinning: Automatic Skin Binding for Production Characters with Deep Graph Networks. *ACM TOG*, 38(4), 2019.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Jing Ma and Dongliang Zhang. TARig: Adaptive Template-Aware Neural Rigging for Humanoid Characters. *Comput. Graph.*, 114, 2023.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning Robust Visual Features without Supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Dmitry Petrov, Pradyumn Goyal, Vikas Thamizharasan, Vladimir Kim, Matheus Gadelha, Melinos Averkiou, Siddhartha Chaudhuri, and Evangelos Kalogerakis. GEM3D: GEnerative Medial Abstractions for 3D Shape Synthesis. In *SIGGRAPH*, 2024.

Jiawei Ren, Liang Pan, Jiaxiang Tang, Chi Zhang, Ang Cao, Gang Zeng, and Ziwei Liu. DreamGaussian4D: Generative 4D Gaussian Splatting. *arXiv preprint arXiv:2312.17142*, 2023.

Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible Isosurface Extraction for Gradient-Based Mesh Optimization. *ACM TOG*, 42(4), 2023.

Chaoyue Song, Xiu Li, Fan Yang, Zhongcong Xu, Jiacheng Wei, Fayao Liu, Jiashi Feng, Guosheng Lin, and Jianfeng Zhang. Puppeteer: Rig and Animate Your 3D Models. In *NeurIPS*, 2025a.

Chaoyue Song, Jianfeng Zhang, Xiu Li, Fan Yang, Yiwen Chen, Zhongcong Xu, Jun Hao Liew, Xiaoyang Guo, Fayao Liu, Jiashi Feng, and Guosheng Lin. MagicArticulate: Make Your 3D Models Articulation-Ready. In *CVPR*, 2025b.

Mingze Sun, Junhao Chen, Junting Dong, Yurun Chen, Xinyu Jiang, Shiwei Mao, Puhua Jiang, Jingbo Wang, Bo Dai, and Ruqi Huang. DRiVE: Diffusion-based Rigging Empowers Generation of Versatile and Expressive Characters. In *CVPR*, 2025a.

Mingze Sun, Shiwei Mao, Keyi Chen, Yurun Chen, Shunlin Lu, Jingbo Wang, Junting Dong, and Ruqi Huang. ARMO: Autoregressive Rigging for Multi-Category Objects. *arXiv preprint arXiv:2503.20663*, 2025b.

Mingze Sun, Cheng Zeng, Jiansong Pei, Junhao Chen, Chaoyue Song, Shaohui Wang, Tianyuan Chang, Bin Huang, Zijiao Zeng, and Ruqi Huang. Animator-Centric Skeleton Generation on Objects with Fine-Grained Details. *arXiv preprint arXiv:2604.20539*, 2026.

Zijie Wu, Chaohui Yu, Fan Wang, and Xiang Bai. AnimateAnyMesh: A Feed-Forward 4D Foundation Model for Text-Driven Universal Mesh Animation. *arXiv preprint arXiv:2506.09982*, 2025.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and Compact Structured Latents for 3D Generation. *arXiv preprint arXiv:2512.14692*, 2025a.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3D Latents for Scalable and Versatile 3D Generation. In *CVPR*, 2025b.

Tianyi Xie, Yunuo Chen, Yaowei Guo, Yin Yang, Bolei Zhou, Demetri Terzopoulos, Ying Jiang, and Chenfanfu Jiang. AnimaMimic: Imitating 3D Animation from Video Priors. *arXiv preprint arXiv:2512.14133*, 2025.

Yuanyou Xu, Zongxin Yang, and Yi Yang. SKDream: Controllable Multi-View and 3D Generation with Arbitrary Skeletons. In *CVPR*, 2025.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets. In *3DV*, 2019.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. Rignet: Neural rigging for articulated characters. *ACM TOG*, 39(4), 2020.

Zhan Xu, , Yang Zhou, Li Yi, and Evangelos Kalogerakis. MoRig: Motion-Aware Rigging of Character Meshes from Point Clouds. In *SIGGRAPH Asia*, 2022.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3DShape2VecSet: A 3D Shape Representation for Neural Fields and Generative Diffusion Models. *ACM TOG*, 42(4), 2023.

Jia-Peng Zhang, Cheng-Feng Pu, Meng-Hao Guo, Yan-Pei Cao, and Shi-Min Hu. One Model to Rig Them All: Diverse Skeleton Rigging with UniRig. *ACM TOG*, 44(4), 2025.

Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. CLAY: A Controllable Large-Scale Generative Model for Creating High-Quality 3D Assets. *ACM TOG*, 43(4), 2024.

Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, Bin Fu, Tao Chen, Gang Yu, and Shenghua Gao. Michelangelo: Conditional 3D Shape Generation Based on Shape-Image-Text Aligned Latent Representation. In *NeurIPS*, 2023.

A Detailed description of Open-Vocabulary Joint Label Assignment

Many downstream animation pipelines assume that the joints of an input skeleton are labeled. For example, motion retargeting methods often require correspondences between source and target joints, such as matching `LeftArm`, `Spine`, or `Head` across different characters. However, template-free skeleton generation methods typically output only joint coordinates and connectivity, without semantic joint names. This limits their compatibility with existing animation tools and retargeting pipelines, where joint labels provide the semantic bridge between different skeletons.

A closed-set joint classifier would restrict retargeting to a predefined label vocabulary or skeleton template, which is undesirable for generated assets whose rigs may contain non-standard parts, different naming conventions, or object-specific articulations. Instead, we use an open-vocabulary formulation so that joints can be queried using labels from arbitrary source templates, including human, animal, creature, and object rigs. This enables retargeting pipelines to establish correspondences to a wider range of templates without retraining the labeler for each label set.

We therefore introduce an open-vocabulary joint labeling module as a post-processing component for generated rigs. Given a generated skeleton and the corresponding surface and skeleton SLats, the module embeds each joint into a shared vision-language space and retrieves semantic labels using text queries. This design allows the model to assign labels beyond a fixed closed vocabulary, while still exploiting the geometry, hierarchy, and rig-aware latent context produced by our method. Traditional artist-created rigs usually include manually assigned joint or bone labels. These labels often encode the body part or object part associated with the joint, such as `Torso`, `Arm`, `Tail`, or `Handle`, as well as relative location descriptors such as `Left`, `Right`, `Upper`, or `Lower`. In contrast, prior template-free rigging methods Xu et al. [2020], Song et al. [2025b,a], Liu et al. [2025], Deng et al. [2025], Sun et al. [2025b], Zhang et al. [2025], Guo et al. [2025a] typically generate structural skeletons only. Their output joints are either unlabeled or assigned generic identifiers such as `Joint1` or `Bone003`, which do not encode anatomical, semantic, or functional correspondences needed for retargeting.

Label preprocessing. The Anymate dataset Deng et al. [2025] provides text labels for many joints, but these labels are collected from heterogeneous artist-created assets and are not standardized. They may contain armature prefixes, namespace strings, duplicated identifiers, inconsistent left/right conventions, or non-informative names. We clean these labels using an LLM-based preprocessing pipeline with QWEN3-8B Yang et al. [2025]. The preprocessing removes asset-specific prefixes and suffixes, normalizes common side and part descriptors, and filters samples whose labels are uninformative or ambiguous. Examples are shown in Tab. 7; the full prompt and filtering rules are provided in Section D.

Joint-text embedding model. We train a model that maps each joint to the embedding space of a frozen OpenCLIP text encoder Cherti et al. [2023]. Let $\boldsymbol{j}_k \in \mathbb{R}^3$ denote the coordinate of joint k. The model receives the joint coordinates, the skeleton hierarchy, and the surface and skeleton SLats. We first embed joints with positional encodings:

$$\boldsymbol{h}^0_{\boldsymbol{j}_k} = \phi_J(\boldsymbol{j}_k) + \gamma(\boldsymbol{j}_k). \tag{11}$$

The joint embeddings are processed by transformer blocks that combine self-attention over joints with cross-attention to the structured latent representations:

$$\boldsymbol{H}'_J = \mathrm{SelfAttn}(\boldsymbol{H}^0_J), \tag{12}$$

$$\boldsymbol{H}''_J = \mathrm{CrossAttn}\left(\boldsymbol{H}'_J, \bar{\boldsymbol{Z}}_{\mathrm{surf}}, \bar{\boldsymbol{Z}}_{\mathrm{surf}}\right), \tag{13}$$

$$\boldsymbol{H}'''_J = \mathrm{CrossAttn}\left(\boldsymbol{H}''_J, \bar{\boldsymbol{Z}}_{\mathrm{skel}}, \bar{\boldsymbol{Z}}_{\mathrm{skel}}\right), \tag{14}$$

$$\boldsymbol{e}_J = \mathrm{MLP}(\boldsymbol{H}'''_J). \tag{15}$$

The output $\boldsymbol{e}_J$ is projected to the CLIP text embedding dimension and ℓ_2-normalized. For each cleaned joint label ℓ_k, we compute a normalized text embedding:

$$\boldsymbol{e}_{\ell_k} = \mathrm{CLIP}_{\mathrm{text}}(\ell_k). \tag{16}$$

We train the joint embedding model with an InfoNCE contrastive objective over joints and labels in a minibatch:

$$\mathcal{L}_{\mathrm{label}} = -\frac{1}{B}\sum_{k=1}^{B} \log \frac{\exp(\boldsymbol{e}_{J_k}^\top \boldsymbol{e}_{\ell_k}/\tau)}{\sum_{m=1}^{B}\exp(\boldsymbol{e}_{J_k}^\top \boldsymbol{e}_{\ell_m}/\tau)}, \tag{17}$$

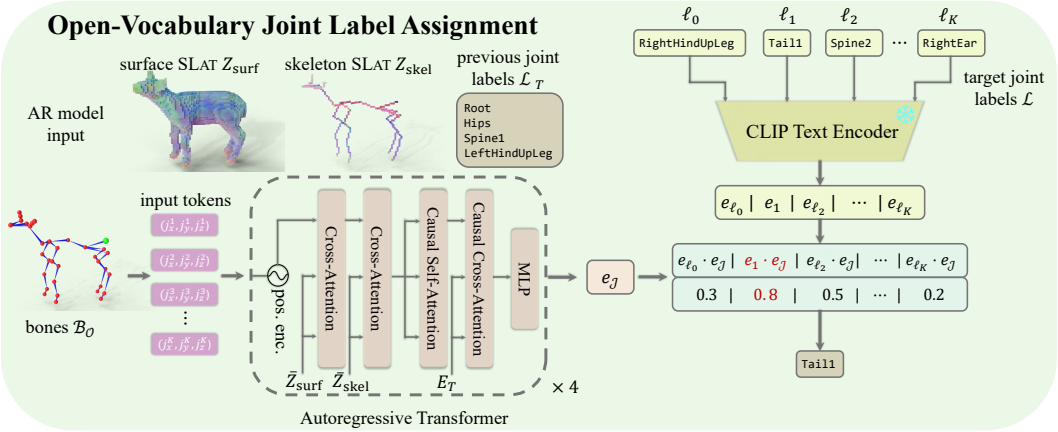

Figure 6: Open-vocabulary joint label assignment. Our model coembeds skeleton geometry and text-captions, allowing for label querying and further compatibility with downstream animation pipelines.

where τ is a learned or fixed temperature. Labels from other joints in the minibatch serve as negatives. At inference time, each generated joint can be labeled by nearest-neighbor retrieval among any candidate vocabulary, including the joint names of a target retargeting template, or queried directly with arbitrary text prompts such as `left wrist`, `tail base`, or `front wheel`. This formulation separates the learned joint representation from the choice of label vocabulary, allowing the same generated rig to be matched against different retargeting templates at inference time. Note that because left–right and repeated structures can be ambiguous from local geometry alone, the model conditions on both the full skeleton and the global SLat context when predicting joint embeddings.

Autoregressive labeling variant. We also evaluate an autoregressive variant that predicts joint embeddings while conditioning on previously labeled joints. This is useful because joint labels are not independent: for example, the presence of `LeftUpperArm` increases the likelihood of nearby descendants such as `LeftForeArm` and `LeftHand`. Following the BFS ordering used by the skeleton decoder, each transformer block performs cross-attention to the latent grids, causal self-attention over previous joints, and causal cross-attention to the embeddings of previous labels:

$$\boldsymbol{H}'_J = \text{CrossAttn}\left(\boldsymbol{H}^0_J, \bar{\boldsymbol{Z}}_{\text{surf}}, \bar{\boldsymbol{Z}}_{\text{surf}}\right), \tag{18}$$

$$\boldsymbol{H}''_J = \text{CrossAttn}\left(\boldsymbol{H}'_J, \bar{\boldsymbol{Z}}_{\text{skel}}, \bar{\boldsymbol{Z}}_{\text{skel}}\right), \tag{19}$$

$$\boldsymbol{H}'''_J = \text{CausalSelfAttn}(\boldsymbol{H}''_J), \tag{20}$$

$$\boldsymbol{H}^*_J = \text{CausalCrossAttn}(\boldsymbol{H}'''_J, \boldsymbol{E}_T, \boldsymbol{E}_T), \tag{21}$$

$$\boldsymbol{e}_J = \text{MLP}(\boldsymbol{H}^*_J), \tag{22}$$

where $\boldsymbol{E}_T$ denotes the sequence of CLIP embeddings of previous labels, using ground-truth labels during training and generated labels during inference. The model is trained with the same contrastive objective as Eq. (10), using teacher forcing over the ground-truth label sequence.

B Training and Implementation Details

In this section we provide details regarding hyperparameters and the training and hardware setups.

We finetune $\mathcal{E}_{surf}$ and $\mathcal{D}_M$ on the Anymate Deng et al. [2025] training split for 1 day on 4 RTX 6000 Ada 48GB GPUs starting from the provided TRELLIS Xiang et al. [2025b] checkpoints and retaining the same loss weighting parameters. The joint training procedure of $\mathcal{E}_{skel}$, $\mathcal{D}_M$ (including the extra cross-attention layer), skeleton generation and skinning models is done under the same hardware setup. We randomly sample 5k assets from the Anymate training split to use as a validation set and train for $\sim$ 10 days until convergence, with a batch size of 4 per GPU. We set $\lambda_{skel} = 1$ and $\lambda_{skin} = 1$. As mentioned in the main text, due to memory constraints we freeze $\mathcal{E}_{surf}$ and pre-extract the surface SLats for each item in the training split.

We train the surface and skeleton Sparse Structure VAEs for 24 hours on 4 NVIDIA A5000 24GB GPUs with a batch size of 4 per GPU. We finetune $\mathcal{G}^{occ}_{skel}, \mathcal{G}^{occ}_{surf}, \mathcal{G}^{lat}_{skel}, \mathcal{G}^{lat}_{surf}$ for 2 days each on 8

Table 3: Geometric reconstruction and Image-to-3D comparisons with TRELLIS

	PSNR↑	LPIPS↓	CD↓	F-score↑	CLIP↑	FD_{dinov2}↓	KD_{dinov2}↓
TRELLIS	31.077	**0.0373**	**0.3677**	98.64	**89.30**	34.14	12.30
Ours	**31.130**	0.0382	0.3704	98.64	88.73	**30.83**	**9.35**

NVIDIA A5000s with a batch size of 2 per GPU, this time starting from the pretrained TRELLIS checkpoints. We found that the provided TRELLIS checkpoints slightly improve performance compared to training from scratch even for the skeleton flow models.

The skeleton labeling transformer is trained on 4 NVIDIA A5000s GPUs with a batch size of 16 per GPU for the AR variant and 32 per GPU for the regular variant. Training takes ~ 3 days until convergence, with a randomly sampled validation set of 5k items.

For all training procedures, we use the AdamW optimizer Loshchilov and Hutter [2017] with a learning rate of 1×10^{-4}. For the joint training of the rig-aware autoencoder, skeleton and skinning models specifically, we use a cosine learning rate scheduler with 5000 warmup steps.

All parameters regarding the noise scheduler and samplers of the 4 latent flow models are the same as Xiang et al. [2025b]. For all experiments we use 25 sampling steps and set the classifier-free guidance weight to 5.0.

C Additional Comparisons and Ablation Studies

Comparison with Trellis. To assess geometric reconstruction quality, generation quality and the effect of the skeleton-aware latent space and joint training scheme, we evaluate our method against TRELLIS finetuned on the Anymate dataset. We compare on a selection of metrics curated from Xiang et al. [2025b] to quantitatively measure the quality of the results. Table 3 contains quantitative results for geometry reconstruction and image-to-3D generation. PSNR, LPIPS, Chamfer Distance and F-score measure the reconstruction fidelity of our rig-aware autoencoder. The first two are computed between renders of the reconstructed and ground truth meshes, whereas for CD and F-score we unproject the depth maps of 100 renders from uniformly sampled views into a pointcloud and sample 100k points for computing the metrics. For FD_{dinov2} and KD_{dinov2} we render 4 images of the GT and generated asset with yaw angles every $90°$ and a pitch of $30°$. For CLIP similarity we use 8 images rendered with yaw at every $45°$ and the same pitch used previously. Rigel3D remains competitive with TRELLIS on geometric reconstruction and image-to-3D quality despite jointly training for geometry, skeletons, and skinning. While TRELLIS is slightly better on LPIPS, CD, and CLIP, our method matches its F-score, slightly improves PSNR, and yields significantly better distributional feature metrics, reducing FD_{DINOv2} from 34.14 to 30.83 (9.7% relative reduction) and KD_{DINOv2} from 12.30 to 9.35 (24.0% relative reduction).

Ablation study. We conduct an ablation study on some of the architecture choices for our skeleton generation and skinning models. Since training our method requires access to the ground-truth rigging information, it is unfair to evaluate on the standard geometry conditioned (mesh input) autorigging setting where such information is unavailable during test time. However, because the standard setting is faster to train and evaluate since it does not require the additional training of the flow models or averaging across multiple views, we utilize this setting to perform an ablation study. Table 4 shows the mesh-conditioned skeleton generation results for the ablation variants of our method. *Ours (Queries)* refers to a variant of our method that compresses the variable length surface SLats into a fixed-sized shape condition for the autoregressive skeleton generator via a learnable Query set, similarly to how Michelangelo Zhao et al. [2023] is used to obtain fixed-size global shape information in Song et al. [2025b,a]. *Ours (joint SLat grid)* refers to our method with the encoder input consisting of both the surface DinoV2 features as well as the rig-aware features within the *same* 3D grid. Finally, in *Ours (Skeleton SLats)* we perform a single cross-attention operation with just the rig slats inside each transformer block. Our method, which uses cross-attention with both the surface and skeleton SLats, showcases the effectiveness of the coarse-to-fine, shape-to-skeleton attention based block. For skinning we include the results for the combined SLat grid. *Separating the two SLat types for the cross-attention operations significantly improves the model performance.*

Joint Labeling. We evaluate the accuracy of our labeling model on the Anymate test-split. Since there is no relevant method for this task, we compare our model against the non-autoregressive

Table 4: Ablation of our method for Mesh-to-Skeleton generation results (values reported $\times 100$).

	Anymate			ModelsResource		
	J2J↓	J2B↓	B2B↓	J2J↓	J2B↓	B2B↓
Ours (Queries)	3.826	2.787	2.837	4.105	2.835	3.165
Ours (Joint SLat grid)	3.471	2.472	2.672	3.873	2.587	3.010
Ours (Skeleton SLats)	1.431	1.098	1.909	1.322	**1.038**	2.111
Ours	**1.386**	**1.068**	**1.881**	**1.280**	**1.006**	**2.052**

Table 5: Ablation study for the skinning weight model, evaluated on the standard mesh-conditioned setting (values reported $\times 100$).

	Anymate			ModelsResource		
	L1↓	Precision↑	Recall↑	L1↓	Precision↓	Recall↓
Ours (Joint SLat grid)	1.251	83.48	64.85	1.772	78.25	80.89
Ours	1.013	88.19	65.19	1.280	85.66	83.02

Table 6: Joint label assignment results

	Top1-Acc.↑	Top3-Acc.↑
Ours (TRELLIS SLats)	81.31	91.06
Ours (both SLats)	81.88	91.41
Ours (AR default order)	84.68	91.10
Ours (AR BFS)	**85.90**	**92.06**
Ours (AR DFS)	84.61	90.93
Ours (AR z-y-x)	74.24	86.29

baseline, and report ablations on the same table. Table 6 reports the results of our Open-vocabulary joint labeling method on label assignment accuracy. *Ours (TRELLIS SLats)* refers to the non-autoregressive variant described in Section 4, using only the surface SLats obtained from the finetuned TRELLIS encoder. *Ours (both SLats)* adds an additonal cross-attention layer with our rig-aware SLats, showcasing the importance of skeleton spatial information and the effectiveness of our latent space. Finally, *Ours (AR X)* reports the performance of the autoregressive model that also utilizes the CLIP embeddings of the previously labeled joints. We experiment with various traversals of the skeleton tree. "Default order" refers to no explicit reordering of the joint sequence (i.e. we retain the parent-child order hierarchy of the artist) whereas the other variants are standard tree traversals and spatial ordering.

D Rig Label Dataset Preparation Details

In this section, we provide the full prompt details regarding the LLM based filtering and standardization of the labels provided with the Anymate dataset Deng et al. [2025]. The filtering resulted in approximately 153k samples with informative labels in the training split and $\sim 3.5k$ in the test-split.

The full prompt for processing the raw labels is provided in Listing 1, and examples of labels before and after the processing are shown in Table 7.

E Broader Societal Impact

This work focuses on object-level 3D animation and does not directly involve sensitive personal data or decision-making systems. It may reduce manual effort in animation workflows, with potential misuse limited to generating misleading animated 3D content.

Table 7: Examples of joint label preprocessing.

Unprocessed label	Processed label
metarig-spine.1	Spine1
Armature-mixamorig:LeftForeArm	LeftForeArm
Armature-Mutant:LeftToeBase	LeftToeBase
GLTF_created_0-shoulder.R_Skeleton_14	RightShoulder
Bip001-Bip001_L_UpperArm	LeftUpperArm
group-pasted__Neck	Neck
SK_Female_Rig_V1_MainC-SK_Female_Rig_V1_SH_lShoulderJ	LeftShoulder

Listing 1: Joint label filtering prompt

```
"### ROLE\n"
"You are an intelligent AI assistant for computer graphics,
    specialized in rigging."
"Your goal is to standardize joint labels for professional 3D
    pipelines (Maya/Blender/Unity)."
"Follow the user's requirements carefully and make sure you
    understand them. \n\n"

"### TASK\n"
"Process a list of joint labels separated by ' || '. For each
    label:\n"
"1. Remove technical prefixes (e.g., 'mixamorig:', 'Armature:',
    'Bind_', 'jnt_', 'bn_').\n"
"2. Standardize to PascalCase (e.g., 'lower_arm_L' -> 'LowerArmL').\n"
"3. Translate non-English anatomical terms to English (e.g., 'Cuisse'
    -> 'Thigh', 'Brazo' -> 'Arm'). **Do not guess; if unsure, keep
    the original.**\n"
"4. **PRESERVE** numbers and side indicators (e.g., 'Thigh1',
    'Arm.R', 'LegC'). Do not strip these.\n\n"

"### CRITICAL LOGIC: INFORMATIVE VS UNINFORMATIVE\n"
"- **Informative:** Labels containing recognizable body part or
    object part names (Thigh, Finger, Spine, Wheel, Trigger, etc.).\n"
"- **Uninformative:** Labels that are generic (e.g., 'Joint1',
    'Bone.001', 'Obj_42').\n"
"- :warning: **ACTION:** If **ANY** single label in the provided
    group is 'Uninformative', "
"you must return 'uninformative' as the entire result for that
    group.\n"
"- Use 'uninformative' only for strictly generic names. If a label
    contains a hint of anatomy (e.g., 'Arm_L_003'), treat it as
    informative.\n\n"

"### OUTPUT FORMAT\n"
"- Return ONLY a raw JSON list of strings.\n"
"- No conversational filler, no markdown code blocks, just the
    JSON.\n\n"
"- **IMPORTANT: Your response must start with [ and end with ]. Do
    not include any text before or after the JSON array.**"

"### EXAMPLE\n"
"Input: 'mixamorig:LeftBrazo || Armature:Cuisse_01 || Joint_02'\n"
"Output: [\"uninformative\"]\n"
"Input: 'mixamorig:LeftBrazo || Armature:Cuisse_01'\n"
"Output: [\"LeftArm\", \"Thigh01\"]"
```