DiffuseBot: Breeding Soft Robots With Physics-Augmented Generative Diffusion Models

David S. Hippocampus

Introduction. This paper presents DiffuseBot , a first step toward efficient automatic robotic and virtual creature content creation. We propose using physical simulation to guide the generative process of pretrained large-scale 3D diffusion models. Diffusion models pretrained for 3D shapes provide an expressive base distribution that can effectively propose reasonable candidate geometries for soft robots. In order to sample robots in a physics-aware and performance-driven manner, we first optimize the embeddings that condition the diffusion model, skewing the sampling distribution toward better-performing robots as evaluated by our simulator. Then, we reformulate the sampling process that incorporates co-optimization over structure and control.

Method. Soft robot co-design refers to a joint optimization of the morphology (geometry, body stiffness, and actuator placement) and control of soft robots. Co-design poses challenges in optimization including interplay between body and control variables, see [16] for details.

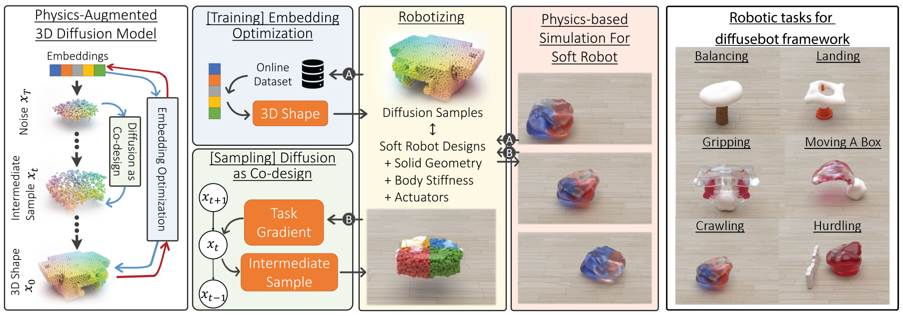

Figure 1: The DiffuseBot framework consists of three modules: (i) *robotizing*, which converts diffusion samples into physically simulatable soft robot designs (ii) *embedding optimization*, which iteratively generate new robots to be evaluated for training the conditional embedding (iii) *diffusion as co-design*, which guides the sampling process with co-design gradients from differentiable simulation.

We adopt Point-E [12] as a pre-trained 3D diffusion model. However, the samples from the diffusion model $\mathbf{x}_t$ are in the form of surface point cloud and are not readily usable as robots to be evaluated in the physics-based simulation, and so the diffusion process must be "robotized" in order to compute the gradient of the objective function.

We use a Material Point Method (MPM)-based simulation [16], which takes solid geometries as inputs. After a mesh reconstruction, the solid geometry Ψ_{geo} represented by a solid interior point cloud is then sampled evenly within the mesh, with sufficient density to support the MPM-based simulation. Finally, to integrate the solidification process into diffusion samplers, we still need its gradient. We adopt Gaussian kernels on point-wise Euclidean distances as gradients between two point clouds: $\frac{\partial \mathbf{u}}{\partial \mathbf{v}} = \frac{\exp(-\alpha\|\mathbf{u}-\mathbf{v}\|^2)}{\sum_{v' \in \hat{\mathbf{x}}_0} \exp(-\alpha\|\mathbf{u}-\mathbf{v}'\|^2)}, \mathbf{u} \in \Psi_{\text{geo}}, \mathbf{v} \in \hat{\mathbf{x}}_0$.

A solid geometry does not make a robot; in order for the robot to behave, its dynamics must be defined. After sampling a solid geometry, we thus need to define material properties and actuator placement. Specifically, we embed actuators in the robot body in the form of muscle fibers that can contract or expand to create deformation; further, we define a stiffness parameterization in order to determine the relationship between deformation and restorative elastic force. We adopt constant stiffness for simplicity since it has been shown to trade off with actuation strength [16]; thus we have

Table 1: Improved physical utility by augmenting physical simulation with diffusion models.

Embed. Optim.	Diffusion as Co-design	Passive Dynamics		Locomotion		Manipulation	
		Balancing	Landing	Crawling	Hurdling	Gripping	Moving a Box
		$0.081^{.164}$	$0.832^{.217}$	$0.011^{.012}$	$0.014^{.020}$	$0.014^{.008}$	$0.019^{.020}$
✓		$0.556^{.127}$	$0.955^{.032}$	$0.048^{.007}$	$0.019^{.014}$	$0.025^{.006}$	$0.040^{.018}$
✓	✓	$\mathbf{0.653}^{.107}$	$\mathbf{0.964}^{.029}$	$\mathbf{0.081}^{.018}$	$\mathbf{0.035}^{.030}$	$\mathbf{0.027}^{.004}$	$\mathbf{0.044}^{.021}$

$\Psi_{\text{st}}(\hat{\mathbf{x}}_0) = \text{const}$ and $\frac{\partial \Psi_{\text{st}}}{\partial \hat{\mathbf{x}}_0} = 0$. Then, we propose to construct actuators based on the robot solid geometry $\Psi_{\text{act}}(\Psi_{\text{geo}}(\hat{\mathbf{x}}_0))$ via clustering; namely, we perform k-means with pre-defined number of clusters on the coordinates of 3D points from the solid geometry Ψ_{geo}. The gradient then becomes $\frac{\partial \Psi_{\text{act}}}{\partial \Psi_{\text{geo}}} \frac{\partial \Psi_{\text{geo}}}{\partial \hat{\mathbf{x}}_0} \frac{\partial \hat{\mathbf{x}}_0}{\partial \mathbf{x}_t}$, where $\frac{\partial \Psi_{\text{act}}}{\partial \Psi_{\text{geo}}} \approx 0$ as empirically we found the clustering is quite stable in terms of label assignment, i.e., with $\Delta \Psi_{\text{geo}}$ being small, $\Delta \Psi_{\text{act}} \to 0$.

To best leverage the diversity of the generation from a large-scale pre-trained diffusion models, we propose to (1) actively generate new data from model and maintain them in a buffer, (2) use physics-based simulation as a certificate of performance, (3) optimize the embeddings conditioned by the diffusion model under a skewed data distribution to improve robotic performance in simulation. Curating a training dataset on its own alleviates the burden of manual effort to propose performant robot designs. Optimizing the conditional embeddings instead of finetuning the diffusion model eliminates the risk of deteriorating the overall generation and saves the cost of storing model weights for each new task (especially with large models). We follow, $\min_{\mathbf{c}} \mathbb{E}_{t\sim[1,T], p_\theta(\mathbf{x}_0|\mathbf{c}), \mathcal{N}(\epsilon;\mathbf{0},\mathbf{I})}[||\epsilon - \epsilon_\theta(\mathbf{x}_t(\mathbf{x}_0, \epsilon, t), t, \mathbf{c})||^2]$

Algorithm 1 Sampling: Diffusion As Co-design

```
Initialize: initial sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
while within maximal number of diffusion steps $t \geq 0$ do
    Perform regular per-step diffusion update: $\mathbf{x}_t \Leftarrow \mathbf{x}_{t+1}$.
    if perform co-design then
        while within $K$ steps do
            Run update in (1) and (2) to overwrite $\mathbf{x}_t$.
        end while
    end if
end while
```

We further improve the performance of individual samples by reformulating the diffusion sampling process into a co-design optimization. There is a synergy between diffusion models and energy-based models [15, 4, 3], which allows a more gradient-descent-like update with Markov Chain Monte Carlo sampling [3]. Incorporating gradient-based co-optimization [16], we have

$$\text{Design Optim.: } \mathbf{x}_t^{(k)} = \mathbf{x}_t^{(k-1)} + \frac{\sigma^2}{2}\left(\epsilon_\theta(\mathbf{x}_t^{(k-1)}, t) - \kappa \nabla_{\mathbf{x}_t^{(k-1)}} \mathcal{L}(\Psi(\mathbf{x}_t^{(k-1)}), \phi_t^{k-1})\right) + \sigma^2 \epsilon \quad (1)$$

$$\text{Control Optim.: } \phi_t^{(k)} = \phi_t^{(k-1)} + \gamma \nabla_{\phi_t^{(k-1)}} \mathcal{L}(\Psi(\mathbf{x}_t^{(k-1)}), \phi_t^{k-1}) \quad (2)$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{x}_t^{(0)} = \mathbf{x}_{t-1}^{(K)}$, κ is the ratio between two types of design gradients, K is the number of MCMC sampling steps at the current diffusion time, γ is the weight for trading off design and control optimization, and $\phi_t^{(0)}$ can be either inherited from the previous diffusion time $\phi_{t-1}^{(K)}$ or reset to the initialization $\phi_T^{(0)}$. We highlight the high resemblance to gradient-based co-optimization with $\mathbf{x}_t$ as the design variable and ϕ_t as the control variable. This procedure is performed once every M diffusion steps (Algorithm 1), where M is a hyperparameter that trade-offs "guidance" strength from physical utility and sampling efficiency. Intuitively, the entire diffusion-as-co-design process is guided by three types of gradients: (i) $\epsilon_\theta(\mathbf{x}_t^{(k-1)}, \cdot)$ provides a direction for the design toward feasible 3D shapes based on the knowledge of pre-training with large-scale datasets (and toward enhanced physical utility with the optimized embeddings via classifier-free guidance using $\hat{\epsilon}_\theta(\mathbf{x}_t^{(k-1)}, \cdot, \mathbf{c})$), (ii) $\nabla_{\mathbf{x}_t} \mathcal{L}(\Psi(\mathbf{x}_t), \cdot)$ provides a direction for the design toward improving co-design objective $\mathcal{L}$ via differentiable simulation, and (iii) $\nabla_{\phi_t} \mathcal{L}(\cdot, \phi_t)$ provides a direction for the controller toward a better adaption to the current design $\mathbf{x}_t$ that allows more accurate evaluation of the robot performance.

Experiments. We cover three types of robotics tasks: passive dynamics, locomotion, and manipulation (Figure 1). In Table 1, we examine the effectiveness of embedding optimization and diffusion as co-design for improving physical utility. For each entry, we draw 100 samples with preset random seeds to provide valid sample-level comparison (i.e., setting the step size of co-design optimization to zero in the third row will produce almost identical samples as the second row). We report the average performance with standard deviation in the superscript. We observe increasing performance across all tasks while incorporating the two proposed techniques, demonstrating the efficacy of DiffuseBot.

1 Ethical Considerations

Automating engineering and creative design workflows has the ability to empower more people to be creators and democratize design and engineering; however, with every potentially large change, even positive, comes complementary risks. (Semi-)automating aspects of the design of virtual characters or real-world cyberphysical machines has the potential to expand the amount of design and engineering work that can be done, but if the demand for that type of work is capped, it could lead to more competitive labor markets or outright career disenfranchisement. Further, our method currently relies on on Point-E as a base model for diffusion of robot forms. Although Point-E is ethically sourced from free-and-open-source datasets, other models may be directly substituted in its place. Those may not be derived from free datasets, and commercial application would exploit individuals' unpaid labor. Further, some models could capture 3D data that may be personal assets (*e.g.* body scans); in that case our method would generate variations of personal information, thus violating user privacy.

References

[1] Moritz Bächer, Espen Knoop, and Christian Schumacher. Design and control of soft robots using differentiable simulation. *Current Robotics Reports*, 2(2):211–221, 2021.

[2] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems*, 2021.

[3] Yilun Du, Conor Durkan, Robin Strudel, Joshua B Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. *arXiv preprint arXiv:2302.11552*, 2023.

[4] Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*, 2019.

[5] Debkalpa Goswami, Shuai Liu, Aniket Pal, Lucas G Silva, and Ramses V Martinez. 3d-architected soft machines with topologically encoded motion. *Advanced Functional Materials*, 29(24):1808713, 2019.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[7] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[8] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[9] Byungchul Kim, Useok Jeong, Brian Byunghyun Kang, and Kyu-Jin Cho. Slider-tendon linear actuator with under-actuation and fast-connection for soft wearable robots. *IEEE/ASME Transactions on Mechatronics*, 26(6):2932–2943, 2021.

[10] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. *arXiv preprint arXiv:2206.01714*, 2022.

[11] Jonàs Martínez, Jérémie Dumas, and Sylvain Lefebvre. Procedural voronoi foams for additive manufacturing. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[12] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022.

[13] Songyou Peng, Chiyu "Max" Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape as points: A differentiable poisson solver. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[14] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[15] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[16] Tsun-Hsuan Wang, Pingchuan Ma, Andrew Everett Spielberg, Zhou Xian, Hao Zhang, Joshua B. Tenenbaum, Daniela Rus, and Chuang Gan. Softzoo: A soft robot co-design benchmark for locomotion in diverse environments. In *The Eleventh International Conference on Learning Representations*, 2023.

Algorithm 2 Training: Embedding Optimization

```
Initialize: $\mathcal{D} \leftarrow \emptyset$, $\mathbf{c} \leftarrow \varnothing$
while within maximal number of epochs do
    Generate data with the diffusion model: $\mathbf{x}_0 \sim p_\theta(\mathbf{x}_0|\mathbf{c})$.
    Evaluate samples with physics-based simulation: $l(\mathbf{x}_0) = \mathcal{L}(\Psi(\mathbf{x}_0), \phi)$.
    Aggregate and update datasets: $\mathcal{D} \leftarrow \text{Filter}(\mathcal{D} \cup \{\mathbf{x}_0, l\})$.
    Optimize the embedding $\mathbf{c}$ on $\mathcal{D}$ using the objective (5).
end while
```

A Background On Point-E

Point-E [12] is a diffusion-based generative model that produces 3D point clouds from text or images. The Point-E pipeline consists of three stages: first, it generates a single synthetic view using a text-to-image diffusion model; second, it produces a coarse, low-resolution 3D point cloud (1024 points) using a second diffusion model which is conditioned on the generated image; third, it upsamples/"densifies" the coarse point cloud to a high-resolution one (4096 points) with a third diffusion model. The two diffusion models operating on point clouds use a permutation invariant transformer architecture with different model sizes. The entire model is trained on Point-E's curated dataset of several million 3D models and associated metadata which captures a generic distribution of common 3D shapes, providing a suitable and sufficiently diverse prior for robot geometry. The diffused data is a set of points, each point possessing 6 feature dimensions: 3 for spatial coordinates and 3 for colors. We ignore the color channels in this work. The conditioning for the synthesized image in the first stage relies on embeddings computed from a pre-trained ViT-L/14 CLIP model; in the embedding optimization of DiffuseBot, the variables to be optimized is exactly the same embedding. Diffusion as co-design is only performed in the second stage (coarse point cloud generation) since the third stage is merely an upsampling which produces only minor modifications to robot designs. We refer the reader to the original paper [12] for more details.

B Theoretical Motivation

B.1 3D Shape Generation with Diffusion-based Models

Diffusion-based generative models [6, 14] aim to model a data distribution by augmenting it with auxiliary variables $\{\mathbf{x}_t\}_{t=1}^T$ defining a Gaussian diffusion process $p(\mathbf{x}_0) = \int p(\mathbf{x}_T) \prod_{t=1}^T p(\mathbf{x}_{t-1}|\mathbf{x}_t) d\mathbf{x}_{1:T}$ with the transition kernel in the forward process $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$ for some $0 < \beta_t < 1$. For sufficiently large T, we have $p(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$. This formulation enables an analytical marginal at any diffusion time $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ based on clean data $\mathbf{x}_0$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\bar{\alpha}_t = \prod_{i=1}^t 1 - \beta_i$. The goal of the diffusion model (or more precisely the denoiser ϵ_θ) is to learn the reverse diffusion process $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ with the loss,

$$\min_\theta \mathbb{E}_{t\sim[1,T], p(\mathbf{x}_0), \mathcal{N}(\epsilon;\mathbf{0},\mathbf{I})}[||\epsilon - \epsilon_\theta(\mathbf{x}_t(\mathbf{x}_0, \epsilon, t), t)||^2] \tag{3}$$

Intuitively, ϵ_θ learns a one-step denoising process that can be used iteratively during sampling to convert random noise $p(\mathbf{x}_T)$ gradually into realistic data $p(\mathbf{x}_0)$.

To achieve controllable generation with conditioning $\mathbf{c}$, the denoising process can be slightly altered via classifier-free guidance [7, 2],

$$\hat{\epsilon}_{\theta,\text{classifier-free}} := \epsilon_\theta(\mathbf{x}_t, t, \varnothing) + s \cdot (\epsilon_\theta(\mathbf{x}_t, t, \mathbf{c}) - \epsilon_\theta(\mathbf{x}_t, t, \varnothing)) \tag{4}$$

where s is the guidance scale, $\varnothing$ is a null vector that represents non-conditioning. **Embedding Optimization.**

To best leverage the diversity of the generation from a large-scale pre-trained diffusion models, we propose to (1) actively generate new data from model and maintain them in a buffer, (2) use physics-based simulation as a certificate of performance, (3) optimize the embeddings conditioned by the diffusion model under a skewed data distribution to improve robotic performance in simulation. Curating a training dataset on its own alleviates the burden of manual effort to propose performant robot designs. Optimizing the conditional embeddings instead of finetuning the diffusion model eliminates the risk of deteriorating the overall generation and saves the cost of storing model weights for each new task (especially with large models). We follow,

$$\min_{\mathbf{c}} \mathbb{E}_{t\sim[1,T],p_\theta(\mathbf{x}_0|\mathbf{c}),\mathcal{N}(\epsilon;\mathbf{0},\mathbf{I})}[||\epsilon - \epsilon_\theta(\mathbf{x}_t(\mathbf{x}_0,\epsilon,t),t,\mathbf{c})||^2] \tag{5}$$

Note the three major distinctions from (3): (i) the optimization variable is the embedding $\mathbf{c}$ not θ (ii) the denoiser is conditioned on the embeddings $\epsilon_\theta(\ldots,\mathbf{c})$ and (iii) the data distribution is based on the diffusion model p_θ not the inaccessible real data distribution p and is conditioned on the embeddings $\mathbf{c}$. This adopts an online learning scheme as the sampling distribution is dependent on the changing $\mathbf{c}$. The procedure is briefly summarized in Algorithm 2, where *Filter* is an operation to drop the oldest data when exceeding the buffer limit. In addition, during this stage, we use fixed prescribed controllers since we found empirically that a randomly initialized controller may not be sufficiently informative to drive the convergence toward reasonably good solutions; also, enabling the controller to be trainable makes the optimization prohibitively slow and extremely unstable, potentially due to the difficulty of the controller required to be universal to a diverse set of robot designs. After the embedding optimization, we can perform conditional generation that synthesizes samples corresponding to robot designs with improved physical utility via classifier-free guidance as in (4). **Online learning in embedding optimization.** In Section B.1, we discuss how to online collect a dataset to optimize the embedding toward improved physical utility. Given a simplified version of (5)

$$\min_{\mathbf{c}} \mathbb{E}_{p_\theta(\mathbf{x}_0|\mathbf{c})}[g(\mathbf{x}_0,\mathbf{c})] \tag{6}$$

where, for notation simplicity, we drop $t \sim [1,T]$, $\mathcal{N}(\epsilon;\mathbf{0},\mathbf{I})$ in the sampling distribution, and summarize $[||\epsilon - \epsilon_\theta(\mathbf{x}_t(\mathbf{x}_0,\epsilon,t),t,\mathbf{c})||^2]$ as $g(\mathbf{x},\mathbf{c})$. We can rewrite the expectation term as,

$$\int p_\theta(\mathbf{x}_0)\frac{p_\theta(\mathbf{c}|\mathbf{x}_0)}{p_\theta(\mathbf{c})}g(\mathbf{x}_0,\mathbf{c})dx \tag{7}$$

which allows to sample from $p_\theta(\mathbf{x}_0)$ (i.e., generating samples from the diffusion model) and reweight the loss with $\frac{p_\theta(\mathbf{c}|\mathbf{x}_0)}{p_\theta(\mathbf{c})}$; the latter scaling term is essentially proportional to a normalized task performance. Empirically, we can maintain a buffer for the online dataset and train the embedding with the sampling distribution biased toward higher task performance; we use a list to store samples with top-k performance in our implementation (we also tried reshaping the sampling distribution like prioritized experience replay in reinforcement learning but we found less stability and more hyperparameters required in training compared to our simpler top-k approach).

Connection to MCMC. In diffusion sampling, the simplest way to perform reverse denoising process as in main text follows [6],

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\left(\frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t,t)), \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\mathbf{I}\right) \tag{8}$$

Here, the denoising term can either be unconditional $\epsilon_\theta(\mathbf{x}_t,t)$ or conditional via classifier-free guidance $\hat{\epsilon}_{\theta,\text{classifier-free}}(\mathbf{x}_t,t,\mathbf{c})$ as in (4). We use the latter to incorporate the optimized embedding. To further leverage physics-based simulation, we aim to introduce physical utility during diffusion sampling process. One possibility is to utilize classifier-based guidance [2],

$$\hat{\epsilon}_{\theta,\text{classifier-based}} := \epsilon_\theta(\mathbf{x}_t,t) - s\cdot\sqrt{1-\bar{\alpha}_t}\nabla_{\mathbf{x}_t}\log p(\mathbf{c}|\mathbf{x}_t) \tag{9}$$

where $p(\mathbf{c}|\mathbf{x}_t)$ can be conceptually viewed as improving physical utility and $\nabla_{\mathbf{x}_t}\log p(\mathbf{c}|\mathbf{x}_t)$ can be obtained using differentiable physics and the unconditional score. Note that we slightly abuse the notation here by overloading $\mathbf{c}$ with conditioning from differentiable simulation during sampling other than the classifier-free guidance using the optimized embedding. However, combining (8) and (9) much less resembles any gradient-based optimization techniques, which are shown to be effective in soft robot co-design with differentiable simulation [8, 1]. Fortunately, drawing a connection to energy-based models [15, 4, 3], yet another alternative to incorporate conditioning in diffusion models is Markov Chain Monte Carlo (MCMC) sampling [3], where we use Unadjusted Langevin Dynamics,

$$\mathbf{x}_t = \mathbf{x}_t^{(K)}, \quad \text{where } \mathbf{x}_t^{(k)} \sim \mathcal{N}\left(\mathbf{x}_t^{(k)}; \mathbf{x}_t^{(k-1)} + \frac{\sigma^2}{2}\nabla_{\mathbf{x}}\log p(\mathbf{c}|\mathbf{x}_t^{(k-1)}), \sigma^2\mathbf{I}\right) \tag{10}$$

Table 2: Configuration of embedding optimization.

	Balancing	Landing	Crawling	Hurdling	Gripping	Moving a Box
Buffer Size	600	600	60	600	60	60
Min. Buffer Size	60	60	60	60	60	60
Num. Samples / Epoch	60	60	60	60	60	60
Train Iter. / Epoch	1	1	1	1	1	1
Buffer Top-K	12	12	6	6	6	6
Batch Size	6	6	6	6	6	6

Table 3: Configuration of diffusion as co-design.

	Balancing	Landing	Crawling	Hurdling	Gripping	Moving a Box
$t_{\max}$	400	150	400	400	400	400
$t_{\min}$	0	0	0	0	0	0
Δt	50	25	60	60	60	60
K	3	3	5	5	5	5
σ	$10^{-4} \cdot \beta$	$10^{-4} \cdot \beta$	$10^{-4} \cdot \beta$	$10^{-4} \cdot \beta$	$10^{-4} \cdot \beta$	$10^{-4} \cdot \beta$
κ	10^4	10^4	10^4	10^4	10^4	10^4
γ	-	-	0.01	0.001	0.001	0.001
Renorm Scale	10	10	10	10	10	10

where K is the number of samples in the current MCMC with k as indexing, σ^2 is a pre-defined variance, and $\mathbf{x}_t^{(0)} = \mathbf{x}_{t-1}$. In the context of diffusion models, this procedure is commonly performed within a single diffusion step to drive the sample toward higher-density regime under the intermediate distribution $p(\mathbf{x}_t) = \int q(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)d\mathbf{x}_0$ at diffusion time t. Inspired by its resemblance to gradient ascent with stochasticity from the added Gaussian noise of variance σ^2, we establish a connection to design optimization, reformulating diffusion process as co-design optimization as in (1) and (2). Specifically, we can apply Bayes rule to decompose the score of $p(\mathbf{c}|\mathbf{x}_t^{(k-1)})$,

$$\nabla_{\mathbf{x}} \log p(\mathbf{c}|\mathbf{x}_t) = \nabla_{\mathbf{x}} \log p(\mathbf{x}_t|\mathbf{c}) - \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) \tag{11}$$

where $\nabla_{\mathbf{x}} \log p(\mathbf{x}_t)$ is simply the denoiser output $\epsilon_\theta(\mathbf{x}_t, t)$ and $\nabla_{\mathbf{x}} \log p(\mathbf{x}_t|\mathbf{c})$ is the gradient of task performance with respect to the intermediate sample of the diffusion model from differentiable physical simulation and robotizing process. Overall, this leads to (1).

C Implementation Details In Algorithm

In this section, we provide more implementation details and experimental configurations of DiffuseBot and other baselines. In Table 2, we list the configurations of the embedding optimization. With respect to Algorithm 2, "buffer" refers to the online dataset $\mathcal{D}$.

- *Buffer Size* is the capacity of the dataset.
- *Min. Buffer Size* is the minimum of data filled in the buffer before training starts.
- *Num. Samples / Epoch* is number of new samples collected in each epoch, where epoch here refers to a new round of data collection in online learning.
- *Train Iter. / Epoch* is number of training iterations per epoch.
- *Buffer Top-K* is the number of datapoints with top-k performance being retained in the *Filter* step atop the most up-to-date data.
- *Batch Size* is the batch size in the embedding optimization.

In Table 3, we list the configurations of diffusion as co-design. We follow (1)(2) for:

- K is number of MCMC sampling steps at the current diffusion time.
- σ is the standard deviation related to the MCMC step size.
- κ is the ratio between two types of design gradients.

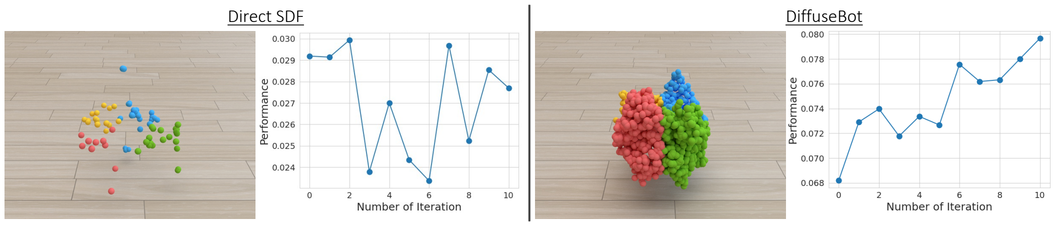

Figure 2: Comparison between *Direct SDF* and DiffuseBot on conversion to solid geometry for robotizing.

- γ is the weight for trading off design and control optimization.
- t_{max} and t_{min} are the maximal and minimal diffusion time to perform diffusion as co-design, respectively.
- Δt is the diffusion time interval to perform diffusion as co-design.

For baselines, we use learning rates for control optimization following γ in Table 3; for particle-based and voxel-based approaches, we use learning rate 0.01 for design optimization; for implicit function and diff-CPPN, we use learning rate 0.001 for design optimization. For the inputs of implicit function and diff-CPPN, we use x, y, z coordinates in the local workspace, the distance to the workspace center on the xy, xz, yz planes, and the radius from the center. For the network architecture of implicit function, we use a 2-layer multilayer perceptron with hidden size 32 and Tanh activation. For the network architecture of diff-CPPN, we use Sin, Sigmoid, Tanh, Gaussian, SELU, Softplus, Clamped activations with 5 hidden layers and 28 graph nodes in each layer.

D Details In Task Setup

In this section, we provide more details of all tasks setup including environment configuration and prescribed actuator and controller if any (as mentioned in Section B.1; it is fixed during embedding optimization and will be co-optimize during diffusion as co-design). We build our environments on top of SoftZoo [16] and employ the Material Point Method for simulation. Each environment is composed of boundary conditions that include impenetrable ground, and, in the case of fixed objects or body parts, glued particles. All tasks use units without direct real-world physical correspondence and last for 100 steps with each step consisting of 17 simulation substeps. In the following, when describing the size of a 3D shape without explicitly mentioning the axis, we follow the order: length (x, in the direction when we talk about left and right), height (y, parallel with the gravity direction), and width (z). All tasks are demonstrated in Figure 1 in the main paper.

Balancing. The robot is initialized atop a stick-like platform of shape 0.02-unit $\times$ 0.05-unit $\times$ 0.02-unit. The robot is givenan initial upward velocity of a 0.5-unit/second and allowed to free fall under gravity. The goal is for the robot to passively balance itself after dropping again on the platform; the performance is measured as the intersection over union (IoU) between the space occupied by the robot during the first simulation step and the space occupied by the robot during the last simulation step. The robot geometry is confined to a 0.08-unit $\times$ 0.08-unit $\times$ 0.08-unit workspace. There is no prescribed actuator placement or controller (passive dynamics).

Landing. The robot is initialized to be 0.08-unit to the right and 0.045-unit above the landing target with size of 0.02-unit $\times$ 0.05-unit $\times$ 0.02-unit. The robot is given an initial velocity of 0.5-unit/second to the right. The goal of the robot is to land at the target; the performance is measured as the exponential to the power of the negative distance between the target and the robot in the last frame $e^{-||p_H^{\text{object}} - p_H^{\text{robot}}||}$, where $p_H^{\cdot}$ is the position of the robot or object at the last frame with horizon H. The robot geometry is confined to a 0.08-unit $\times$ 0.08-unit $\times$ 0.08-unit workspace. There is no prescribed actuator placement or controller (passive dynamics).

Crawling. The robot is initialized at rest on the ground. The goal of the robot is to actuate its body to move as far away as possible from the starting position; the performance is measured as the distance traveled $||p_H^{x,\text{robot}} - p_0^{x,\text{robot}}||$, where $p_\cdot^{x,\text{robot}}$ is the position of the robot in the x axis at a certain frame with horizon H. The robot geometry is confined to a 0.08-unit $\times$ 0.08-unit $\times$ 0.08-unit workspace. The actuator placement is computed by clustering the local coordinates of the robot centered at its average position in the xz (non-vertical) plane into 4 groups. Each group contains an actuator with its direction parallel to gravity. The prescribed controller is a composition of four sine waves with frequency as 30hz, amplitude as 0.3, and phases as 0.5π, 1.5π, 0, and π for the four actuators.

Hurdling. An obstacle of shape 0.01-unit $\times$ 0.03-unit $\times$ 0.12-unit is placed in 0.07-unit front of the robot. The goal of the robot is to jump as far as possible, with high distances achieved only by bounding over the obstacle. The performance is measured as the distance traveled. The robot geometry is confined to a 0.08-unit $\times$ 0.08-unit $\times$ 0.08-unit workspace. The actuator placement is computed by clustering the local coordinates of the robot centered at its average position in the length direction into 2 groups. Each group contains an actuator aligned parallel to gravity. The prescribed controller takes the form of open-loop, per-step actuation sequences, set to linearly-increasing values from (0.0, 0.0) to (1.0, 0.3) between the first and the thirtieth frames and zeros afterward for the two actuators (the first value of the aforementioned actuation corresponds to the actuator closer to the obstacle) respectively.

Gripping. An object of shape 0.03-unit $\times$ 0.03-unit $\times$ 0.03-unit is placed 0.08-unit underneath the robot. The goal of the robot is to vertically lift the object; the performance is measured as the vertical distance of the object being lifted $||p_H^{y,\text{object}} - p_0^{y,\text{object}}||$, where $p_\cdot^{y,\text{object}}$ is the position of the object in the y axis at a certain frame with horizon H. Within a 0.06-unit $\times$ 0.08-unit $\times$ 0.12-unit workspace, we decompose the robot into a base and two attached submodules, and we set the output of DiffuseBot or other robot design algorithms as one of the submodules and make constrain the other submodule to be a mirror copy; conceptually we design "the finger of a parallel gripper." The base is clamped/glued to the upper boundary in z and given a large polyhedral volume to serve as the attachment point of the gripper finger submodules. The actuator placement is computed by clustering the local coordinates of the robot centered at its average position in the length direction into 2 groups; each submodule has one pair of actuators. Each actuator is aligned parallel to gravity. Overall, one pair of actuators comprises the "inner" part of the gripper and the other comprises the "outer" part. Suppose the actuation of the two pairs of actuators is denoted in the format of (actuation of the outer part, actuation of the inner part), the prescribed controller is per-frame actuation, set to (i) linearly-increasing values from (0.0, 0.0) to (0.0, 1.0) between the first and the fiftieth frames, and (ii) then linearly-decreasing values from (1.0, 0.0) to (0.0, 0.0) between the fiftieth and the last frames.

Moving a box. A 0.03-unit $\times$ 0.03-unit $\times$ 0.03-unit cube is placed on the right end of the robot (half a body length of the robot to the right from the robot center). The goal of the robot is to move the box to the left; the performance is measured as the distance that the box is moved to the right with respect to its initial position $||p_H^{x,\text{object}} - p_0^{x,\text{object}}||$, where $p_\cdot^{x,\text{object}}$ is the position of the object in the x axis at a certain frame with horizon H. The robot geometry is confined to a 0.16-unit $\times$ 0.06-unit $\times$ 0.06-unit workspace. The actuator placement is computed by clustering the local coordinates of the robot centered at its average position in the height direction into 2 groups. Each group contains an actuator aligned parallel to the ground. The prescribed controller is per-frame actuation, initialized to linearly-increasing values from (0.0, 0.0) to (1.0, 0.0) between the first and last frame for the lower and the upper actuator respectively.

E Analysis On Robotizing

As mentioned in main text, Material Point Method simulation requires solid geometry for simulation; thus, we need to convert the surface point cloud from Point-E [12] to a volume. The most direct means of converting the point cloud to a solid geometry is to compute the signed distance function (SDF), and populate the interior of the SDF using rejection sampling. We refer to this baseline as *Direct SDF*. Here, we use a pretrained transformer-based model provided by Point-E as the SDF. In Figure 2, we compare Direct SDF with our approach described in main text Solid Geometry. We perform robotizing on intermediate samples at t=300. We observe that Direct SDF fails to produce well-structured solid geometry since it is trained with watertight on geometry, and thus the conversion cannot gracefully handle generated point clouds that do not exhibit such watergith structure. This is specifically common in the intermediate diffusion sample $\mathbf{x}_t$ as $\mathbf{x}_t$ is essentially a Gaussian-noise-corrupted version of the clean surface point cloud. In contrast, the robotizing of DiffuseBot produces a much well-structured solid geometry since it explicitly handles the noisy interior 3D points by introducing a tailored loss as in Shape As Points optimization [13] (see main text). In addition, a better-structured robot geometry is critical to obtain not only a more accurate evaluation of a robot design at the forward pass of the simulation but also the gradients from differentiable physics at

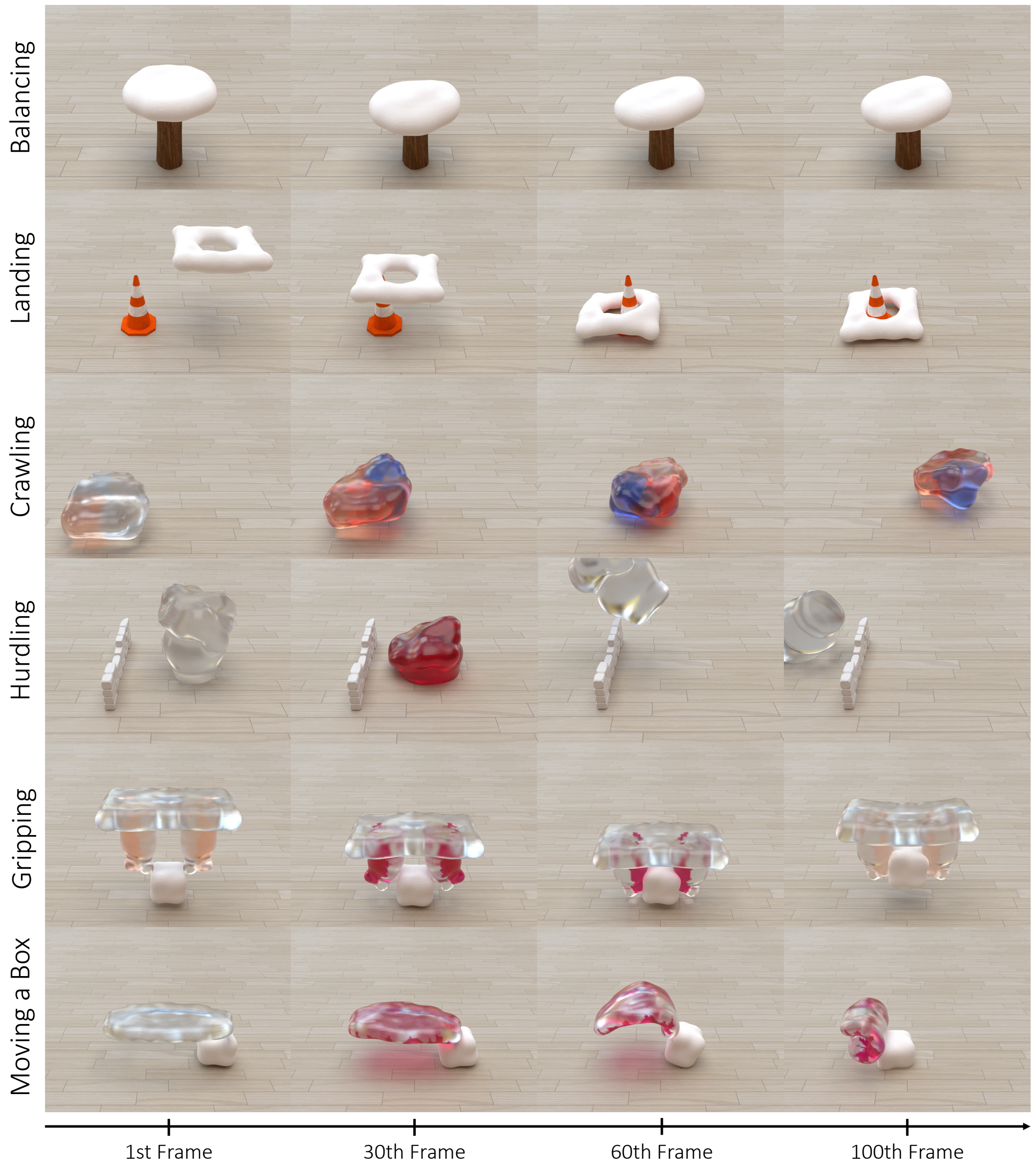

Figure 3: Examples of robots bred by DiffuseBot to achieve the all presented tasks.

the backward pass. In Figure 2, we further perform co-optimization on the two robots obtained by Direct SDF and DiffuseBot; we observe a more stably increasing trend of task performance in our approach, demonstrating that the gradient is usually more informative with a better-structured robot. Empirically, while Direct SDF sometimes still provides improving trend with unstructured robots in co-design optimization, the performance gain is not as stable as DiffuseBot.

F Additional Visualizations Of Experiments

In this section, we show more visualization for a diverse set of the experimental analysis. Please visit our project page (`https://sites.google.com/view/diffusebot`) for animated videos.

Generated robots performing various tasks. In Figure 3, we display a montage of a generated robot's motion for each task.

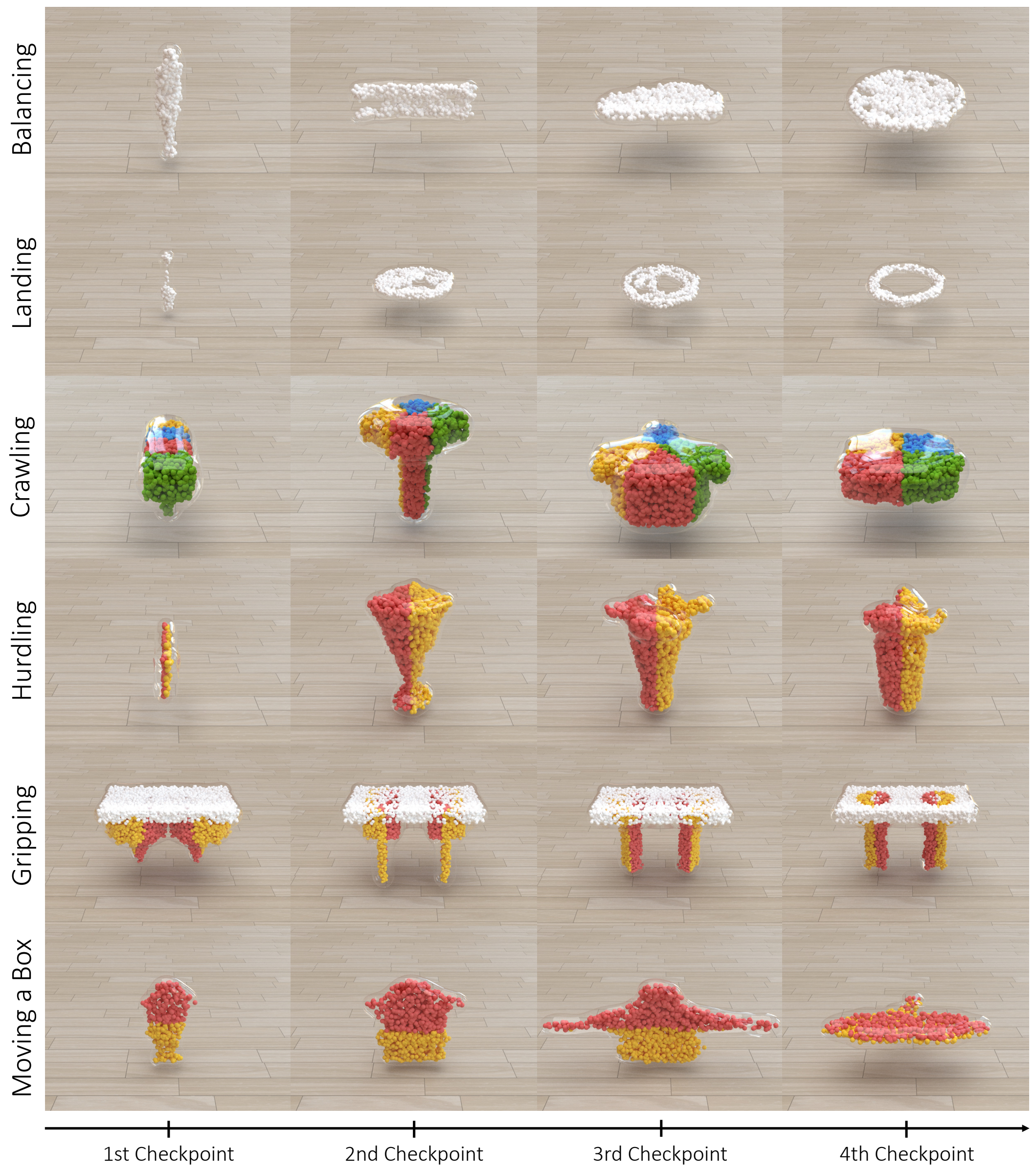

Figure 4: Examples of DiffuseBot evolving robots to solve all presented tasks.

Physics-guided robot generation In Figure 4, we show how DiffuseBot evolves robots throughout the embedding optimization process. Note that DiffuseBot is a generative algorithm and thus results presented are for demonstration purposes and reflect only a single fixed random seed.

Diffusion process of robot generation. In Figure 5, we show how robots evolve throughout the diffusion process at different diffusion time via $\mathbf{x}_t$ with t from T to 0; not to be confused by Figure 4 where all generations are obtained *via* a full diffusion sampling $\mathbf{x}_0$ and different conditioning embeddings $\mathbf{c}$. This figure demonstrates the robotized samples across diffusion times, gradually converging to the final functioning robots as t approaches 0.

Comparison with baselines. In Figure 6, we present robots generated from different baselines. Note that Point-E is essentially DiffuseBot without embedding optimization and diffusion as co-design.

Incorporating human feedback. In Figure 8, we showcase more examples of incorporating human textual feedback to the generation process of DiffuseBot. We leverage the compositionality of diffusion-based generative models [10, 3]. Specifically, this is done by combining two types of

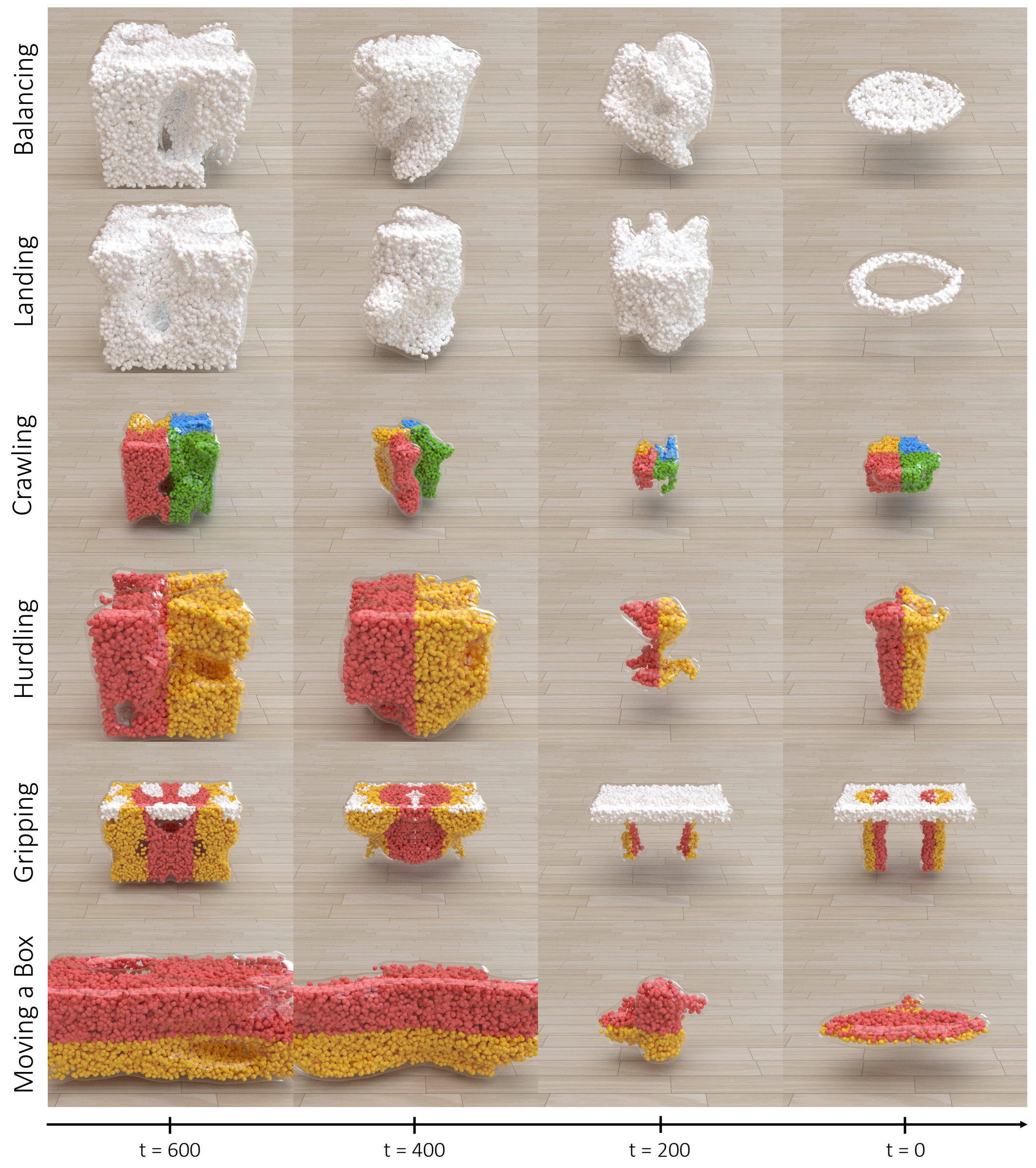

Figure 5: Demonstrations of generations changing from noises to robots with physical utility throughout the diffusion sampling process. We present snapshots at diffusion times $t = 600, 400, 200, 0$ for all presented tasks.

classifier-free guidance as in (4); one source of guidance is derived from the task-driven embedding optimization while the other is derived from embeddings from human-provided textual descriptions. These two conditional scores, namely $\epsilon_\theta(\mathbf{x}_t, t, \mathbf{c}_{\text{physics}})$ and $\epsilon_\theta(\mathbf{x}_t, t, \mathbf{c}_{\text{human}})$, can then be integrated into the diffusion as co-design framework as in (1). We refer the reader to the original papers for more details and theoretical justification. We demonstrate examples of: the balancing robot with additional text prompt *"a ball"*, the crawling robot with additional text prompt *"a star"*, the hurdling robot with additional text prompt *"a pair of wings"*, and the moving-a-box robot with additional text prompt *"thick"*. Interestingly, human feedback is introduced as an augmented "trait" to the original robot. For example, while the hurdling robot keeps the tall, lengthy geometry that is beneficial for storing energy for jumping, a wing-like structure appears at the top of the robot body instead of overwriting the entire geometry. This allows users to design specific parts for composition while also embedding fabrication and aesthetic priorities and constraints through text. We leave future explorations of these ideas for future work.

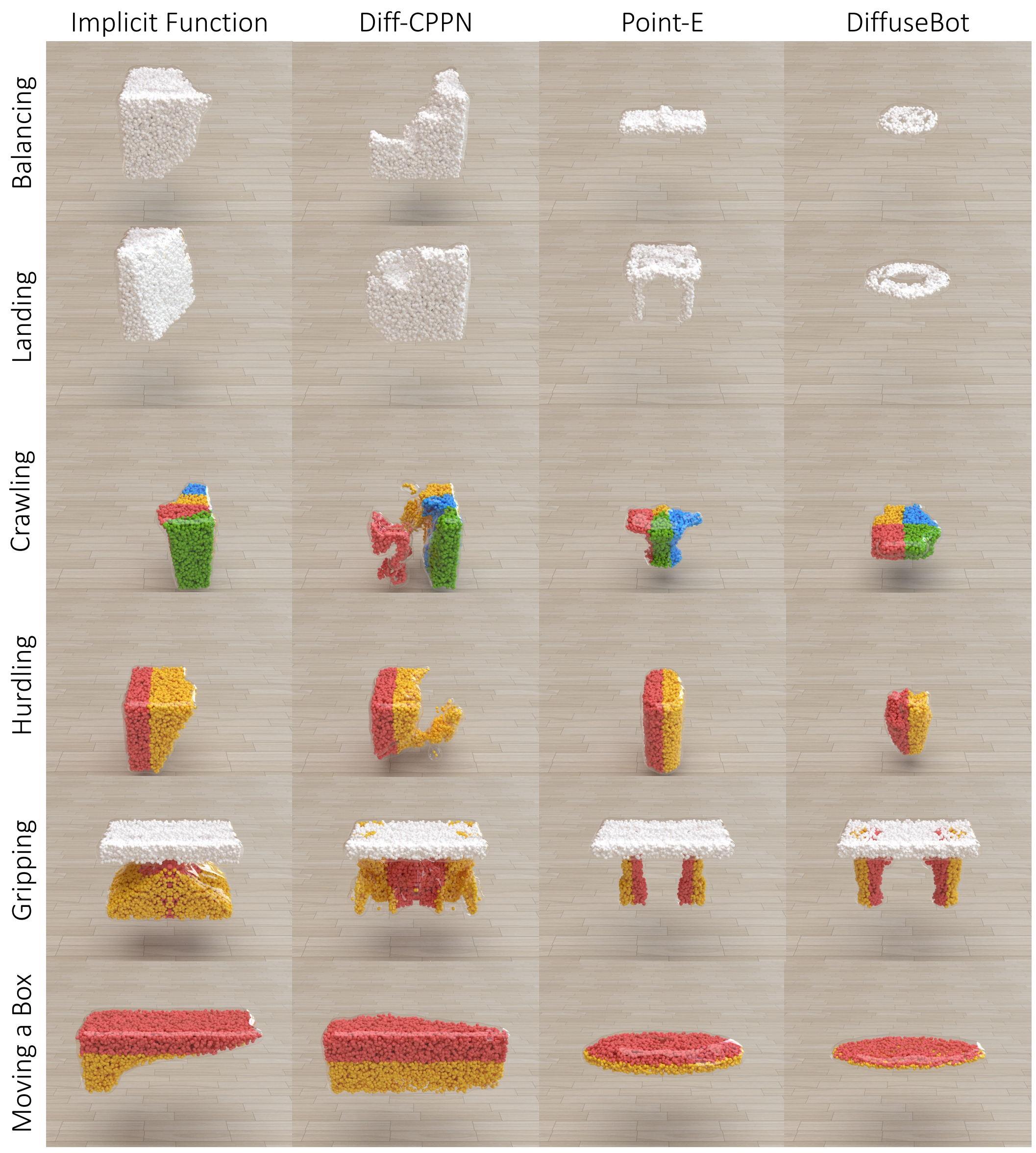

Figure 6: Comparison of robots generated by different baselines.

G Hardware Design and Fabrication

To create a real-world counterpart of the DiffuseBot, we fabricated a proof-of-concept physical robot gripper, as illustrated in Figure 9(a). The gripper was designed to match the shape of the digital gripper and successfully demonstrated its ability to grasp objects, as shown in Figure 10. A video demonstrating the grasping process can be found on our project page (`https://sites.google.com/view/diffusebot`).

During the translation from the simulated design to a physical robot, we aimed to minimize differences between the simulated and physical designs. However, due to hardware and fabrication limitations, certain modifications were necessary.

One such challenge involved replicating the arbitrary contraction force at the particles from the simulation in the physical world. To address this, we employed tendon transmission as the actuation method for the real-world gripper, which was found to be suitable for emulating interaction forces from the digital world without introducing significant discrepancies. To enhance the stability of the

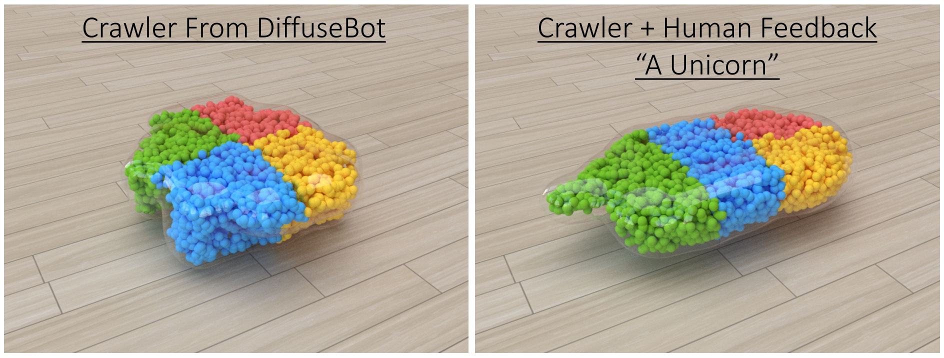

Figure 7: Incorporating human textual feedback.

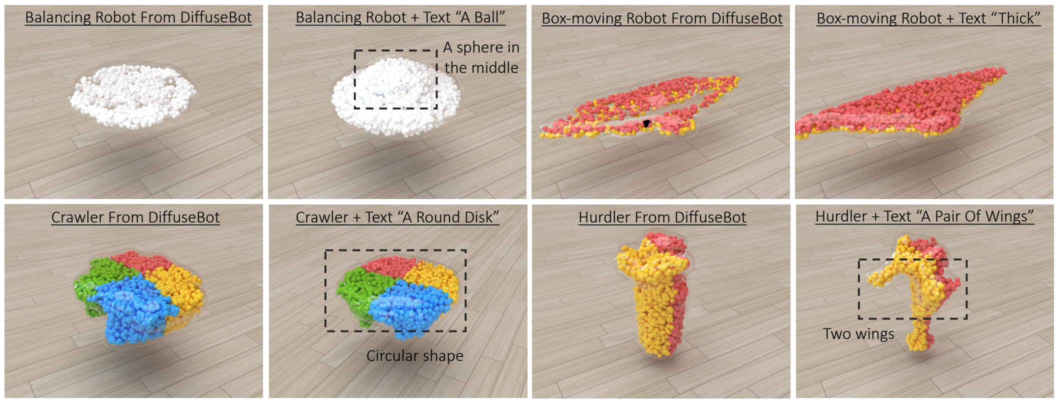
Balancing Robot From DiffuseBot
Balancing Robot + Text "A Ball"
A sphere in the middle
Box-moving Robot From DiffuseBot
Box-moving Robot + Text "Thick"
Crawler From DiffuseBot
Crawler + Text "A Round Disk"
Circular shape
Hurdler From DiffuseBot
Hurdler + Text "A Pair Of Wings"
Two wings

Figure 8: More examples of incorporating human textual feedback into robot generation by DiffuseBot.

tendon-driven actuation, we incorporated a rigid base ("Base" in Figure 9(a)) above the gripper itself ("Gripper" in Figure 9(a)). This base was designed to withstand the reaction force generated by the tendon-driven actuation. Furthermore, we added four tendon routing points (represented by small dotted circles in Figure 9(b)) on each finger to securely fix the tendon path. By utilizing four Teflon tubes (shown in Figure 9(b)), we were able to position complex components such as motors, batteries, and controllers away from the gripper, reducing its complexity.

The actuation strategy of the simulated gripper applies equal contraction force to both fingers simultaneously. To replicate this strategy, we employed underactuated tendon routing in the development of the gripper. This approach eliminates the need for four separate actuators, thereby reducing the complexity of the robot. We used tendon-driven actuators specifically designed for underactuated tendon-driven robots as they solve practical issues such as size and friction issues that commonly arise in such systems [9].

The soft gripper was 3D-printed using a digital light projection (DLP) type 3D printer (Carbon M1 printer, Carbon Inc.) and commercially available elastomeric polyurethane (EPU 40, Carbon Inc.). To enhance the softness of the robot body beyond the inherent softness of the material itself, we infilled the finger body with a Voronoi lattice structure, a metamaterial useful for creating a soft structure with tunable effective isotropic stiffness [5, 11]. We generated a Voronoi lattice foam with point spacing of 2.5mm, and a beam thickness of 0.4 mm as shown in the red boundary of Fig. 9(c). Finally, we tested the soft gripper, designed and fabricated as described above, to verify its ability to grasp an object, as shown in Figure 10.

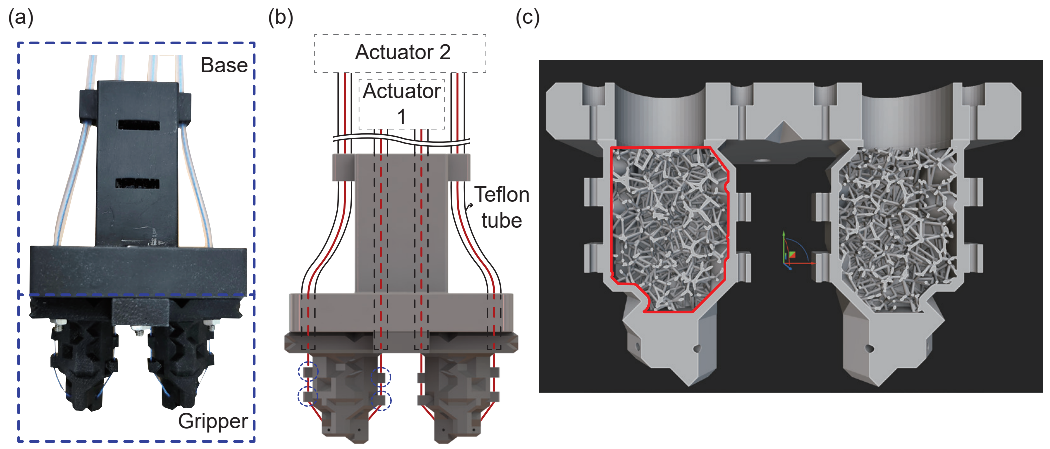

Figure 9: **(a)** shows a fabricated proof-of-concept physical robot gripper; **(b)** describes detailed robot configuration; **(c)** represents interior structure design used to soften the robot body.

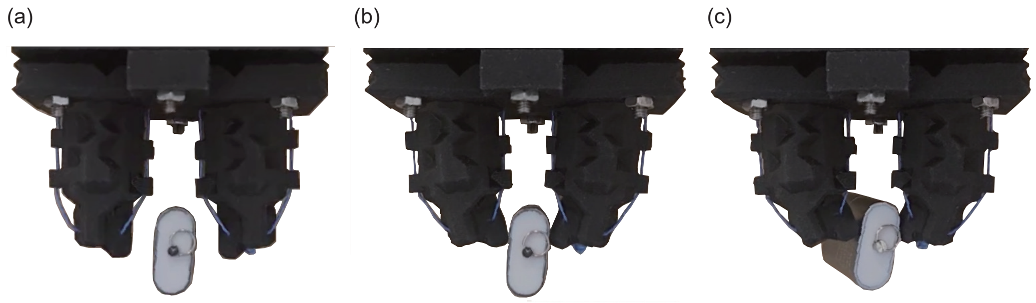

Figure 10: Grasping motion of the real-world soft gripper from DiffuseBot. Time order from **(a)** to **(c)**. The demo video can be seen at our project page.