GLSO: Grammar-guided Latent Space Optimization for Sample-efficient Robot Design Automation
Supplementary Material

A Variational Autoencoder Implementation Details

Network Dimensions: In this work, we set the latent vector $z \in \mathbb{R}^{28}$, hidden representation of nodes $h_i \in \mathbb{R}^{450}$, and embedding of node attributes $x_{\text{emb}} \in \mathbb{R}^{450}$. The dimensions of the weights in the encoder and decoder are set accordingly. The property prediction network is implemented as a MLP with two hidden layers of size 128, with ReLU activation after each hidden layer. We used PyTorch [1] for all network implementation.

Training: Our VAE uses a learning rate of 1e-3, which is decayed every 40000 training steps at a rate of 0.9. We optimize the trainable weights using Adam [2], while applying a gradient clipping of magnitude 50. We trained the model for 400000 steps with a batch size of 32 to obtain the reported results. We anneal the weight of KL loss from zero to one during training, which helps ensure that the encoder does not start by pushing the KL loss to zero, as noted in [3].

B Bayesian Optimization Implementation Details

Our Bayesian Optimization (BO) implementation, using [4], begins with 50 steps of random exploration, which helps diversify the exploration space. Subsequent sampling points are determined by the Expected Improvement (EI) acquisition function. Every 10 steps, a random point is sampled to explore. We used an optimization bound of $[-3, 3]$ across all latent dimensions. Our EI acquisition function has $\xi = 0.01$, which controls the exploration rate. Our Gaussian Process uses a Matern kernel with $\nu = 2.5$. We additionally apply domain reduction, with shrinkage parameter $\gamma_{\text{osc}} = 0.7$, panning parameter $\gamma_{\text{pan}} = 1.0$, zoom parameter $\eta = 0.9$.

C GRU equations

The following equations are used to calculate the messages passed between nodes in the message passing graph neural networks in the VAE at each iteration. Notation follows from the main text, where W and U refer to trainable weights, s, z and r are internal variables, and m refers to the messages.

$$s_{ij} = \sum_{k \in N(i)/j} m_{ki} \tag{1}$$

$$z_{ij} = \sigma(W^z x_i + U^z s_{ij} + b^z) \tag{2}$$

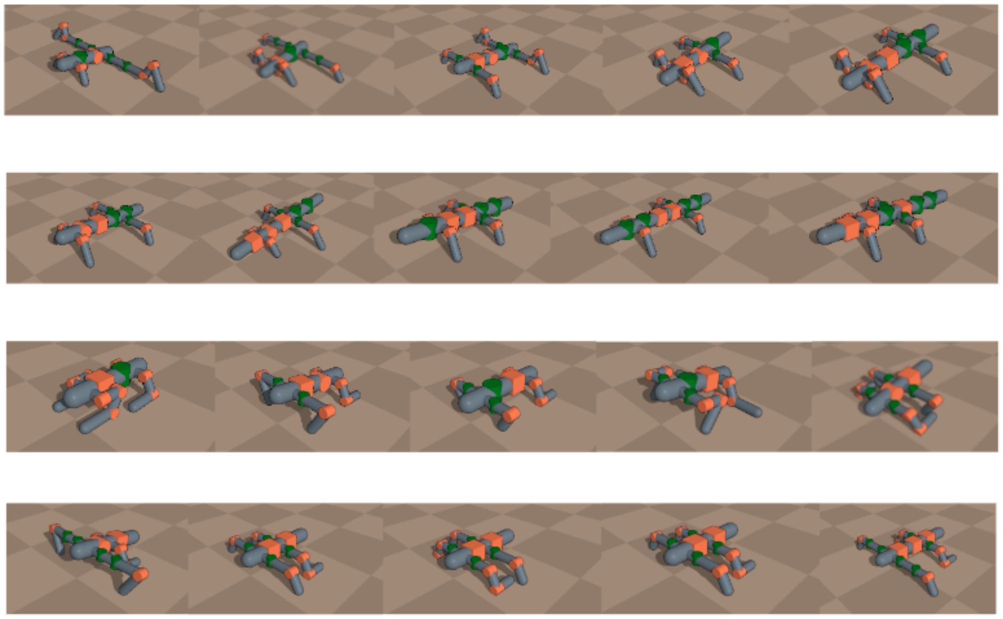
$$r_{ki} = \sigma(W^r x_i + U^r m_{ki} + b^r) \tag{3}$$

$$\tilde{m}_{ij} = tanh(W x_i + U \sum_{k \in N(i)/j} r_{ki} \bigodot m_{ki}) \tag{4}$$

$$m_{ij} = (1 - z_{ij}) \bigodot s_{ij} + z_{ij} \bigodot \tilde{m}_{ij} \tag{5}$$

D Latent Space Interpolation

To gain an intuition behind how the latent space maps a combinatorial design space to a continuous one, we created visualizations of how the designs vary as the latent vector is linearly interpolated between two points:



The left and right-most designs in each row are obtained from two points in latent space, and the designs between them are created by linearly interpolating the latent vector at equal intervals, then decoding those variables.

References

[1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[2] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[4] F. Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014. URL `https://github.com/fmfn/BayesianOptimization`.