

APPENDIX

A THEORETICAL DETAILS

The full statement of the following theorem considers an MBO algorithm with function class $\mathcal{F} = \mathcal{F}_{C_1} \oplus \dots \oplus \mathcal{F}_{C_{N_{\text{clique}}}}$, so that every of its element has form

$$f(\mathbf{x}) = \sum_{i=1}^{N_{\text{clique}}} f_{C_i}(\mathbf{x}_{C_i}).$$

As described in Section 3, Cliqueformer’s architecture forms such a function class on top of the learned latent space. We define the statistical constant as

$$C_{\text{stat}} = \sqrt{\frac{1}{1-\sigma}}, \quad \text{where } \sigma = \max_{C_i \neq C_j, \hat{f}_{C_i}, \hat{f}_{C_j}} \mathbb{E}_{\mathbf{x} \sim p} [\hat{f}_{C_i}(\mathbf{x}_{C_i}), \hat{f}_{C_j}]$$

and the function approximation complexity constant as

$$C_{\text{cpx}} = \sqrt{\frac{N_{\text{clique}} \sum_{i=1}^{N_{\text{clique}}} \log(|\mathcal{F}_{C_i}|/\delta)}{N}},$$

where δ is the PAC error probability (Shalev-Shwartz & Ben-David, 2014).

Theorem 1 (Grudzien et al. (2024)). *Let $f(\mathbf{x})$ be a real-valued function, \mathcal{C} be the set of maximal cliques of its FGM, and Π be a policy class. Let C_{stat} and C_{cpx} be constants that depend on the probability distribution of \mathbf{x} and function approximator class’s complexity, respectively, defined in Appendix A. Then, the regret of MBO with the FGM information is given by,*

$$\eta(\pi^*) - \eta(\hat{\pi}_{\text{FGM}}) \leq C_{\text{stat}} C_{\text{cpx}} \max_{\pi \in \Pi, \mathbf{x} \in \mathcal{X}, C \in \mathcal{C}} \frac{\pi(\mathbf{x}_C)}{p_C(\mathbf{x}_C)}.$$

Theorem 2. *Let $d \geq 2$ be an integer and $\mathbf{x} \in \mathbb{R}^d$ be a random variable with positive density in \mathbb{R}^d . There exists a function $f(\mathbf{x})$ and two different reparameterizations, $\mathbf{z} = z(\mathbf{x})$ and $\mathbf{v} = v(\mathbf{x})$, of \mathbf{x} , that both follow a standard-normal distribution, but the FGM of f with respect to \mathbf{z} is a complete graph (has all possible edges), and with respect to \mathbf{v} it is an empty graph (has no edges).*

Proof. Since the density of \mathbf{x} is positive and continuous, we can form a bijection that maps \mathbf{x} to another random variable $\mathbf{z} \in \mathbb{R}^l$, where $l \leq d$, that follows the standard-normal distribution (Dai & Wipf, 2019, Appendix E). We denote this bijection as $Z(\mathbf{x})$. Let us define

$$\mathbf{y} = f^z(\mathbf{z}) = \exp\left(\frac{1}{\sqrt{l}} \sum_{i=1}^l z_i\right).$$

Then, the FGM of f^z has an edge between every two variables since each variable’s partial derivative

$$\frac{\partial f^z}{\partial z^i} = \frac{1}{\sqrt{l}} \exp\left(\frac{1}{\sqrt{l}} \sum_{i=1}^l z_i\right)$$

is also a function of all others (Grudzien et al., 2024, Lemma 1). Consider now a rotation $\rho : \mathbf{z} \mapsto \mathbf{v} = (v_1, \dots, v_l)$ such that $v_1 = \frac{1}{\sqrt{l}} \sum_{i=1}^l z_i$. Then, $\mathbf{v} \sim N(0_l, I_l)$, and \mathbf{y} can be expressed in terms of \mathbf{v} as $\mathbf{y} = f^v(\mathbf{v}) = \exp(v_1)$. Then, the FGM of f^v has no edges, since it depends on only one variable, inducing no interactions between any two variables. Recall that $\mathbf{x} = Z^{-1}(\mathbf{z})$. Then, \mathbf{x} be represented by standard-normal \mathbf{z} and \mathbf{v} , obtainable by

$$\mathbf{z} = Z(\mathbf{x}) \text{ and } \mathbf{v} = \rho(\mathbf{z}) = \rho(Z(\mathbf{x})).$$

Furthermore, we can define

$$f(\mathbf{x}) = f^z(Z(\mathbf{x}))$$

which is identically equal to $f^z(\mathbf{z})$ and $f^v(\mathbf{v})$, which have a complete and an empty FGM, respectively, thus fulfilling the theorem’s claim. \square

B EXPERIMENTAL DETAILS

Datasets. We use the implementation of Grudzien et al. (2024) to generate data with latent radial-basis functions. Also, we initially wanted to use Design Bench (Trabucco et al., 2022) for experiments with practical tasks. However, at the time of this writing, the benchmark suite was suffering a data loss and was not readily available. To overcome it, we manually found the data and implemented dataset classes. TFBind-8 (Trabucco et al., 2022) could be fully downloaded since the number of possible pairs (\mathbf{x}, \mathbf{y}) is quite small. Hence, a design can be evaluated by looking up its score in the dataset. For Superconductor (Hamidieh, 2018), we pre-trained an XGBoost oracle on the full dataset, and trained our model and the baselines to predict the labels produced by the oracle. The proposed designs of the tested models are evaluated by calling the oracle as well. We obtained DNA Enhancers dataset from the code of Uehara et al. (2024), available at

`https://github.com/masa-ue/RLfinetuning_Diffusion_Bioseq/tree/master`.

Following the procedure in

`https://github.com/masa-ue/RLfinetuning_Diffusion_Bioseq/blob/master/tutorials/Human-enhancer/1-Enhancer_data.ipynb`.

we additionally filter the dataset to keep only sequences featured by chromosomes from 1 to 4. We use their pre-trained oracle for generating labels and evaluation of proposed designs. Following Fannjiang & Listgarten (2020) and Trabucco et al. (2022), we train our models on the portions of the datasets with values below their corresponding 80th. Upon evaluation, we obtain the ground-truth/oracle value of the proposed design y , and normalize it as

$$\bar{y} = \frac{y - y_{\min}}{y_{\max} - y_{\min}},$$

and report \bar{y} . y_{\min} and y_{\max} are the minimum and the maximum of the training data. This normalization scheme is different than, for example, the one in the work by Trabucco et al. (2022). We choose this scheme due to its easy interpretability—a score of $\bar{y} > 1$ implies improvement over the given dataset, which is the ultimate objective of MBO methods. However, we note that a score of less than 1 does not imply failure of the algorithm, since we initialize our designs at a random sample from the dataset, which can be arbitrarily low-value or far from the optimum. For some functions, like in latent RBFs, the optima are very narrow spikes in a very high-dimensional space, being nearly impossible to find (see Figure 1a). We choose such an evaluation scheme due to its robustness that allows us to see how good at improving any design our algorithms are overall.

Hyper-parameters. For baselines, we use hyper-parameters suggested by Trabucco et al. (2021). We decreased the hidden layer sizes (at no harm to performance) for LRBF 31 and DNA Enhancers tasks where the performance was unstable with larger sizes. Also, we haven’t tuned most of the Cliquesformer’s hyper-parameters *per-task*. We found, however, a set of hyper-parameters that works reasonably well on all tasks.

On all tasks, we use **2 transformer blocks** in both the encoder and the decoder, with **transformer dimension of 64**, and **2-head attention**. The predictive model $f_{\theta}(\mathbf{z})$ is a multi-layer perceptron with **2 hidden layers of dimension 256**. We change it to 512 only for DNA Enhancers. The best activation function we tested was **GELU** (Hendrycks & Gimpel, 2016), and LeakyReLU(0.3) gives similar results. We use **dropout of rate 0.5** (Srivastava et al., 2014). In all tasks, **weight of the MSE term to $\tau = 10$** (recall Equation (5)). Additionally, we warm up our VIB term linearly for 1000 steps (with maximal coefficient of 1). We train the model with AdamW (Loshchilov et al., 2017) with the default weight decay of Pytorch (Paszke et al., 2019). We set the **model learning rate to 1e-4** and the **design learning rate to 3e-4** in all tasks. We train the design with AdamW with high rates of weight decay (ranging from 0.1 to 0.5).

In all tasks, we wanted to keep the dimension of the latent variable \mathbf{z} more-less similar to the dimension of the input variable \mathbf{x} , and would decrease it, if possible without harming performance, to limit the computational cost of the experiments. The dimension of \mathbf{z} can be calculated from the clique and knot sizes as

$$\dim(\mathbf{z}) = d_{\text{knot}} + N_{\text{clique}} \cdot (d_{\text{clique}} - d_{\text{knot}}).$$

In most tasks, we used the clique dimension $d_{\text{clique}} = 3$ with knot size of $d_{\text{knot}} = 1$. We made an exception for Superconductor, where we found a great improvement by setting $d_{\text{clique}} = 21$ and $N_{\text{clique}} = 4$ (setting $d_{\text{clique}} = 3$ and $N_{\text{clique}} = 40$ gives score of 0.99); and DNA Enhancers, where we doubled the clique size (to 6) and halved the number of cliques to (40), to lower the computational cost of attention. In DNA Enhancers tasks, we additionally increased the MLP hidden dimension to 512 due to greater difficulty of modeling high-dimensional tasks. We summarize the task-specific hyper-parameters in Table 2.

We want to note that these hyper-parameters are not optimal per-task. Rather, we chose schemes that work uniformly *well enough* on all tasks. However, each task can benefit from further alteration of hyper-parameters. For example, we observed that LRBF tasks benefit from different numbers of design steps; for LRBF 41, we found the optimal number to be 400; for Superconductor, it seems to be 200. Due to time constraints, we have not exploited scalability of Cliqueformer in DNA Enhancers tasks, but observed pre-training losses to decrease more with increased parameter count and training duration.

Task	N_clique	d_clique	MLP dim	design steps	Weight decay
LRBF 11	10	3	256	50	0.5
LRBF 31	18	3	256	50	0.5
LRBF 41	20	3	256	50	0.5
LRBF 61	28	3	256	50	0.5
TFBind-8	4	3	256	1000	0.5
Superconductor	4	21	256	1000	0.5
Dna Enhancers	40	6	512	1000	0.1

Table 2: Hyper-parameter configuration for different benchmark tasks.