Tree-Based Neural Bandits for High-Value Protein Design

Chenyu Wang[1] Tsinghua University

Joseph Kim[1] Princeton University

Le Cong Stanford University

Mengdi Wang[2] Princeton University

Abstract

Protein design involves searching over a large combinatorial sequence space. Evaluating the fitness of new protein sequences often requires wet-lab experiments that are costly and time consuming. In this paper we propose a tree-based neural bandits algorithm that utilizes a modified upper confidence bound algorithm for accelerating the search for optimal designs. The algorithm makes adaptive queries starting from single mutations and generalizes to multiple mutations, as guided by the neural bandit and a Monte Carlo tree search process. The algorithm is tested on two public protein fitness datasets, Protein G B1 and YAP WW domains. For both datasets, our algorithm consistently identifies top fitness protein sequences. Notably, this approach finds a diverse and rich class of high fitness proteins using substantially fewer design queries compared to a range of alternative methods.

1 Introduction

Proteins are the essential building blocks of life. Due to their versatility, proteins have been extremely valuable targets of scrutiny for diverse real-world applications. To this end, protein engineering aims to design protein variants with novel or improved properties.

One approach is Bayesian optimization, which has a long history since the seminal work by Kushner (1964). Gaussian processes (GPs) are commonly used for Bayesian optimization since they provide calibrated uncertainty estimates. Several works have utilized Gaussian processes in conjunction with Bayesian optimization or bandit algorithms for protein design (Romero et al., 2013; Frisby and Langmead, 2021; Hie et al., 2020)

Another very powerful protein engineering strategy is directed evolution (DE) (Arnold, 1998; Romero and Arnold, 2009), which was originally an innovative, purely wet-lab procedure, recognized by the 2018 Nobel Prize in Chemistry. DE involves iterative rounds of mutagenesis at selected or random positions of a protein (generating a library of variants), selective screening for desirable traits, and amplification of top designs to be used as templates for the next round of mutagenesis. Nonetheless, this is often expensive and time-consuming. Various machine learning (ML) platforms have been developed to assist DE to considerable success (Yang et al., 2019; Wu et al., 2019; Romero et al., 2013; Bedbrook et al., 2019). As the search space for protein design scales exponentially with the size of the protein, these methods could still be quite limiting in that the experimental DE procedures do not consider epistatic interactions that are outside the local search space of these algorithms.

In this work, we present a tree-based neural bandits algorithm that identifies rich and diverse classes of proteins with desirable characteristics in a *global* search space. In addition, our method achieves this performance with significantly fewer protein design queries than currently used algorithms. We utilize recent advances in unsupervised protein language models (Rao et al., 2019; Luo et al., 2021; Rao et al., 2021) to generate embeddings for variant sequences and employ Monte Carlo Tree Search (MCTS) to guide exploration in the protein sequence space. We complement this with neural bandits (Zhou et al., 2020) and compute an upper confidence bound (UCB) based exploration bonus with a kernel trick that substantially improves efficiency and performance.

We test our approach on two public datasets, Protein G B1 (Olson et al., 2014) and YAP WW Domains (Araya et al., 2012). The GB1 protein sequence has 56 residues, and the WW protein has 34 residues, where for each residue there are 20 possible amino acids to choose from. Our tree-based neural bandit algorithm consistently identifies top performing proteins after

[1] Both authors contributed equally to this work.
[2] This is the corresponding author. Correspondence to `mengdiw@princeton.edu`

making only 300 design queries. This query efficiency is remarkable when compared to the massive design space (20^{56} for GB1, 20^{34} for WW) and the large embedding dimensions.

2 Related Works

Bayesian optimization (BayesOpt) is a global black-box optimization algorithm that alternates between two steps: (i) using the data collected so far to train a surrogate model for the objective to be optimized, and (ii) selecting queries based on an acquisition function, which enables balancing exploration and exploitation (see this overview by Frazier (2018)). Most BayesOpt schemes use a Gaussian process (GP) surrogate, as GP's provide calibrated uncertainty estimates. Several works have explored Gaussian processes (GPs) for modeling the protein fitness landscape (Romero et al., 2013; Frisby and Langmead, 2021; Hie et al., 2020). Frisby and Langmead (2021) directly used regularized BayesOpt for protein design. Unfortunately, GPs are hard to scale to large, high-dimensional data and are sensitive to the choice of hyperparameters.

In a similar spirit as Bayesian optimization, Liu et al. (2019) used an ensemble of neural networks to optimize the binding affinity of IgG antibodies. They performed gradient-based optimization on a continuous relaxation of the discrete search space, which is vulnerable to poor generalization.

In general, acquisition function optimization is a non-trivial combinatorial optimization problem. Indeed, Hie and Yang (2021) noted that BayesOpt may not be amenable for protein design, as obtaining well-calibrated uncertainty estimates can be difficult or require many queries when the inputs are discrete and/or high-dimensional.

While directed evolution (DE) has had major success in protein engineering, it is a wet-lab procedure that requires extensive, costly experiments. In addition, DE is sample inefficient and relies on greedy hillclimbing, which often results in getting stuck at local optima. To make DE more sample efficient, several works have investigated utilizing machine learning (ML) to find better sequences more quickly. ML-assisted DE uses the sequence and screening data at each round to update an ML model that predicts the effects of mutations on the property being optimized. The next round is then seeded with the most optimal sequences found by the model. In contrast to classical DE, ML-assisted directed evolution can escape from local optima by learning/generalizing the entire functional landscape from data. Wu et al. (2019) developed a general, straightforward machine learning-assisted directed evolution pipeline. However, these approaches still require an extensive amount of wet-lab experimentation. Furthermore, these methods typically restrict themselves to small subsets of residues for optimization, which drastically reduces the combinatorial search space at the cost of missing many potentially significant epistatic interactions.

The most closely related work to ours was developed by Romero et al. (2013). They modeled the protein fitness landscape using a GP and employed a batch selection upper confidence bound (UCB) algorithm (Desautels et al., 2012) for sequential GP optimization. This approach is equivalent to employing stochastic linear bandits with the respective kernel feature space. Our approach differs from this work and the others listed above in several notable ways. First, in order to sequentially search and optimize in a large *global* search space, we utilize a neural multi-armed bandit approach (Zhou et al., 2020). This leverages a much more powerful function approximator in a neural network (NN), which induces a different kernel. Second, we leverage the neural tangent kernel theory and employ a kernel trick to rigorously calculate a UCB-based exploration bonus to achieve provable learning efficiency using the NN. Third, we incorporate Monte Carlo Tree Search (Coulom, 2006) for more efficient exploration of the protein sequence space. Furthermore, we utilize TAPE (Rao et al., 2019) to generate expressive, low-dimensional embeddings for variant sequences. The combination of neural bandits, tree search, and protein embeddings from language models significantly improves generalizability of our model and accelerates the search for high-value proteins.

3 Problem Formulation

Suppose that we want to optimize a protein, whose design can be encoded by an amino-acid sequence $x \in \mathcal{X}$ of length L. As most proteins are characterized by 20 different amino-acids, $|\mathcal{X}| = 20^L$. A typical value of L can range from 20 to several hundreds.

We are interested in the utility of a protein, which is an unknown function $f(x)$. If we experimentally synthesize a protein with design x, we would obtain a noisy estimate of its utility y such that $\mathbb{E}[y] = f(x)$ (we refer to this as a query). Our goal is to find a design with the highest utility:

$$\max_x f(x).$$

This is a combinatorial optimization problem. If we have no structural knowledge at all, we would have to exhaustively search the entire space $\mathcal{X}$, and the worst-case query complexity would be $|\mathcal{X}| = 20^L$. In this project, we aim to exploit known structures to design a fast online learning algorithm. Our goal is to improve

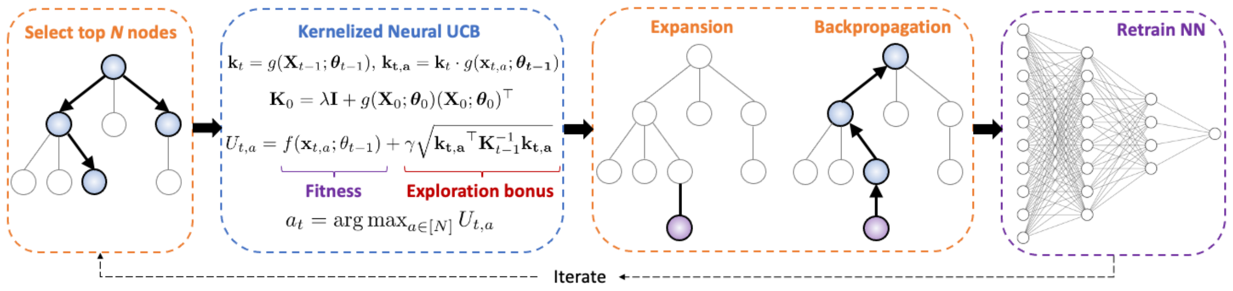

Select top N nodes
Kernelized Neural UCB
$\mathbf{k}_t = g(\mathbf{X}_{t-1};\boldsymbol{\theta}_{t-1})$, $\mathbf{k_{t,a}} = \mathbf{k}_t \cdot g(\mathbf{x}_{t,a};\boldsymbol{\theta_{t-1}})$
$\mathbf{K}_0 = \lambda\mathbf{I} + g(\mathbf{X}_0;\boldsymbol{\theta}_0)(\mathbf{X}_0;\boldsymbol{\theta}_0)^\top$
$U_{t,a} = \underbrace{f(\mathbf{x}_{t,a};\theta_{t-1})}_{\text{Fitness}} + \underbrace{\gamma\sqrt{\mathbf{k_{t,a}}^\top \mathbf{K}_{t-1}^{-1}\mathbf{k_{t,a}}}}_{\text{Exploration bonus}}$
$a_t = \arg\max_{a\in[N]} U_{t,a}$
Expansion
Backpropagation
Retrain NN
Iterate

Figure 1: **Model Architecture.** We first initialize the tree structure and pretrain the NN with a small subset of single mutation variants. Then, for T iterations, we (1) select the top N nodes in the tree by MCTS, (2) compute a kernelized neural UCB score, (3) expand the node corresponding to the best query and backpropagate in the tree, and (4) retrain the NN with updated training data.

the utility via multiple rounds of experiments (each experiment makes a number of queries), and we hope to find the optimal design $x^* = \arg\max_x f(x)$ as fast as possible.

4 Approach

The proposed tree-based neural bandits algorithm is composed of three key components. First, we utilize protein embedding to represent the information underlying the protein sequences, and formulate the high-value protein design problem into multi-armed bandits. Secondly, we apply neural bandits with a novel kernel trick in Upper Confidence Bound calculation. Thirdly, we incorporate Monte Carlo Tree Search into our model to guide exploration in the combinatorial sequence space. The model architecture is illustrated in Figure 1. The full pseudocode for the algorithm is provided in Algorithm 1. Detailed information about the three components are elaborated as follows.

4.1 Protein Embedding and Multi-armed Bandits

Protein modelling acts as the first step for *in silico* protein related research. Various protein embedding methods have been put forward, including TAPE (Rao et al., 2019), ECNet (Luo et al., 2021), MSA Transformer (Rao et al., 2021), etc. We utilize TAPE as the representation of any given amino acid sequences, which is pre-trained with self-supervised learning to extract information from unlabelled sequences.

In contrast to the previous works of Wu et al. (2019) which is under a directed protein evolution framework and Frisby and Langmead (2021) which handled the problem with a regularized Bayesian optimization approach, we formulate the protein design problem into multi-armed bandits in order to find a rich class of high-value proteins with limited experimental queries. To be specific, we utilize the contextual bandits and take the sequences after mutations as the arms with the protein embedding as the context. With regard to the exploration exploitation trade-off, the UCB consisting of both the payoff estimation term and the bonus term is calculated as the query criterion. Compared with directed evolution methods, which rely on greedy optimization, multi-armed bandits can explore the variant sequence space more efficiently.

4.2 Neural Bandits and Kernel Tricks

As illustrated in Section 5, multi-layer perceptrons with TAPE embedding inputs work well in the supervised task to predict fitness value of a given sequence. This supports the use of neural contextual bandits proposed by Zhou et al. (2020) in our protein design problem. In neural contextual bandits, an MLP model $f(\mathbf{x};\boldsymbol{\theta})$ is retrained with the new queries, and the sequences to be queried are selected based on the neural-UCB, $U_{t,a}$, of arm a in the t-th round of query:

$$U_{t,a} = f(\mathbf{x}_{t,a};\boldsymbol{\theta}_{t-1}) + \gamma\sqrt{g(\mathbf{x}_{t,a};\boldsymbol{\theta}_{t-1})^\top \mathbf{Z}_{t-1}^{-1} g(\mathbf{x}_{t,a};\boldsymbol{\theta}_{t-1})/m} \quad (1)$$

where

$$\mathbf{Z}_t = \mathbf{Z}_{t-1} + g(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_{t-1}) g(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_{t-1})^\top/m$$

a_t denotes the arm with largest $U_{t,a}$, m denotes the hidden dimension of the 3-layer MLP, γ is the trade-off parameter, and $g(\mathbf{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})$ is the gradient of the neural network.

Note that the gradient vector $g(\mathbf{x};\boldsymbol{\theta}) \in \mathbb{R}^p$, where $p = m + m^2 + md$ (d denotes the dimension of protein embedding $\mathbf{x}$), can be quite large given a large protein embedding dimension. Thus, $\mathbf{Z}_t \in \mathbb{R}^{p\times p}$ tends to be a large matrix that is intractable to compute and store in memory. In the experimental implementation of Zhou et al. (2020), the authors used the diagonal elements to approximate the matrix to lower the computational and memory cost in the matrix product.

Algorithm 1 Tree-based Neural Contextual Bandits

```
1: Input: Number of rounds $T$, neural bandit exploration parameter $\gamma$, MCTS exploration parameter $C$,
   number of pretraining data $n_{pre}$, number of top sequences selected from MCTS $N$.
2: Network Construction: Define the neural network model $f(\mathbf{x};\boldsymbol{\theta})$ as a 3-layer MLP model with hidden
   dimension size $m$, and is trained by an Adam optimizer with learning rate $\eta$, regularization parameter $\lambda$,
   batch_size $b$ and epoch number $J$.
3: Tree Construction: Construct tree structure $\mathcal{T}$ with the wild type as root node, and the single mutation
   variant of a given node as its children.
4: Initialization: Initialize $\boldsymbol{\theta}_0$ by training the MLP with randomly sampled $n_{pre}$ single mutants embedding
   $\mathbf{X}_0$ and log fitness $\mathbf{R}_0$. Initialize number of visit and value of $\mathcal{T}$ by visiting each node in $\mathbf{X}_0$ once.
5: Initialize $\mathbf{K}_0 = \lambda\mathbf{I} + g(\mathbf{X}_0;\boldsymbol{\theta}_0)g(\mathbf{X}_0;\boldsymbol{\theta}_0)^\top$
6: for $t = 1,\dots,T$ do
7:     Select top $N$ nodes $\{\mathbf{x}_{t,a}\}_{a=1}^N$ from $\mathcal{T}$ based on $U_{t,a}^{inte}$ calculated by (3)
8:     for $a = 1,\dots,N$ do
9:         Compute $U_{t,a} = f(\mathbf{x}_{t,a};\theta_{t-1}) + \gamma\sqrt{(g_{t,a}^\top g_{t,a} - \mathbf{k_{t,a}}^\top \mathbf{K}_{t-1}^{-1}\mathbf{k_{t,a}})/\lambda}$
10:            where $\mathbf{k}_t = (g_1, g_2, \dots, g_n)^\top$, $\mathbf{k_{t,a}} = \mathbf{k}_t \cdot g(\mathbf{x}_{t,a};\theta_{t-1})$
11:        Let $a_t = \arg\max_{a\in[N]} U_{t,a}$
12:    end for
13:    Query sequence $a_t$ and observe log fitness value $r_{t,a_t}$
14:    Conduct node expansion and back propagation with $r_{t,a_t}$ in tree $\mathcal{T}$
15:    Re-train the MLP with the updated training data $\mathbf{X}_t = [\mathbf{X}_{t-1}, \mathbf{x}_{t,a_t}]$, $\mathbf{R}_t = [\mathbf{R}_{t-1}, r_{t,a_t}]$ to get $\boldsymbol{\theta}_t$
16:    Compute $\mathbf{K}_t = \lambda\mathbf{I} + g(\mathbf{X}_t;\boldsymbol{\theta}_t)g(\mathbf{X}_t;\boldsymbol{\theta}_t)^\top$
17: end for
```

However, ignoring all non-diagonal elements leads to severe information loss and poor bonus term calculation. We incorporate kernel tricks into the UCB calculation to solve this problem. To be specific, the gradient vector $g(\mathbf{x};\boldsymbol{\theta})$ is replaced by its kernel representation $\mathbf{k} \in \mathbb{R}^n$ and $\mathbf{Z}_t$ is replaced by a regularized gram matrix $\mathbf{K}_t \in \mathbb{R}^{n\times n}$, where n is the number of data points in model training. Let g_i for $i = 1, 2, \dots, n$ denote the gradient vector of the i-th training input, and define $g_{t,a} := g(\mathbf{x}_{t,a};\theta_{t-1})$. Then the kernelized neural UCB can be formulated as follows:

$$U_{t,a} = f(\mathbf{x}_{t,a};\theta_{t-1}) + \gamma\sqrt{(g_{t,a}^\top g_{t,a} - \mathbf{k_{t,a}}^\top \mathbf{K}_{t-1}^{-1}\mathbf{k_{t,a}})/\lambda} \tag{2}$$

where $\mathbf{K_t}$ is updated as $\mathbf{k_t}\mathbf{k_t}^\top + \lambda\mathbf{I}$, $\mathbf{k}_t = (g_1, g_2, \dots, g_n)^\top$, and $\mathbf{k_{t,a}} = \mathbf{k}_t \cdot g_{t,a}$. We will show in Section 5 that the proposed kernel tricks lead to more informative exploration and better performance compared to the diagonal approximation.

4.3 Incorporate MCTS with Neural Bandits

In each round of query in the neural bandits, the optimal sequence is selected and queried based on the neural UCB. However, it is not only time consuming to calculate UCB for every sequence in the dataset, but also unrealistic in real-life applications where the set of all the possible mutations is exponentially large. One straight-forward method to deal with this problem is to randomly sample a certain number of sequences from the sequence pool, which is sub-optimal and leads to large variance.

A more principled sampling strategy can be achieved by incorporating Monte Carlo Tree Search (Coulom (2006)), which leads to the tree-based neural bandits algorithm. Instead of random sampling, we make use of MCTS to first sample a small number of sequences with top scores, and further select the one with the highest neural UCB. To construct the tree stucture, we take the wild type as the root, and define the children of each node as all of its single mutation variants. This structure is motivated by the fact that we wish to avoid changing the protein too much (otherwise, we are most likely to reduce fitness), which translates to a shallow search in the tree. In contrast to the selection and play-out step in standard MCTS, we select the top N nodes (where N is the number of sampled sequences) instead of just the best node, and get the fitness value of the terminal node directly as the evaluation result V to replace the play-out process.

In order to combine information from both the MCTS and neural bandits, we integrate the two upper confidence bounds by utilizing the tighter bound of the two for the selection of top N nodes:

$$U_{t,a}^{inte} = \min\{U_{t,a}^{neuro}, U_{t,a}^{MCTS}\} \tag{3}$$

where $U_{t,a}^{neuro}$ is derived based on Eq. 2 and $U_{t,a}^{MCTS} = \overline{V_{t,a}} + C\sqrt{\log N(parent(a))/N(a)}$ is the standard

MCTS upper confidence bound, where $N(\cdot)$ denotes the number of visits for a node, C is the exploration-exploitation trade-off parameter for MCTS, and $\overline{V_{t,a}}$ is the average value of node a in round t.

In order to accommodate different types of nodes, e.g. terminal nodes, in the tree structure, we define a case-by-case $U^{inte}_{t,a}$ in our experiments which makes slight modifications to Eq. 3. Detailed methods are included in Appendix A.

5 Experiments

We test the performance of our algorithm on two benchmark protein datasets, Protein G B1 (Olson et al. (2014)) and YAP WW domains (Araya et al. (2012)). We will analyze the properties of the protein fitness data and discuss the effects of each model component in this section.

5.1 Data Description

5.1.1 Protein G B1

Protein G is an antibody-binding protein expressed in groups C and G Streptococcus bacteria. The B1 domain of protein G (GB1 domain), which has 56 residues in the sequence, interacts with the Fc domain of immunoglobulins. Olson et al. (2014) created a library which encodes all single and double mutants in the GB1 protein and profiled the relative binding efficiency to IgG-Fc for each variant. By measuring the number of each variant contained in the input library, n^{in}, and the output "selected" library, n^{sel}, the fitness W_{mut} of a variant is calculated as follows:

$$W_{mut} = \frac{n^{sel}_{mut}/n^{sel}_{wt}}{n^{in}_{mut}/n^{in}_{wt}} \tag{4}$$

where the subscript wt represents the numbers of wild type in the libraries and subscript mut denotes those of the mutant. Thus, $W_{mut} = 1$ indicates a variant with the same fitness as the wild type.

Unlike Frisby and Langmead (2021) and Wu et al. (2019), which used the dataset of the 149,361 out of 160,000 (i.e. 20^4) possible variants of four sites in protein G B1 by Wu et al. (2016), we focus on the dataset with the fitness landscape of almost all single and double mutation variants of all the 56 sites. It contains fitness measurements for all 1,045 single mutants and 509,693 double mutants (95.1% of all $\binom{56}{2} \times 19^2 = 555,940$) with low experimental error.

5.1.2 WW Domain

The hYAP65 WW domain is a modular domain with 34 residues which mediates a variety of protein-protein interactions. Araya et al. (2012) used deep mutational scanning to measure the ability of 47,000 unique variants of the WW domain protein to bind to their polyproline peptide ligand, and the corresponding fitness value relative to the wild type is calculated the same as Eq. 4.

Among the 47,000 variants, 24,665 variants with low experimental standard error were identified. Among these, 478 contain a single mutation, 17,389 contain two mutations, 6,115 contain three mutations, and 681 contain more than 3 mutations.

The kernel density estimation for log fitness values of different levels of mutants are shown in Figure 2. The histograms are normalized by the number of variants. Most of the variants are inferior to the wild type with log fitness smaller than 0. The 95th and 99th percentiles of the log fitness values are marked in the figure. Compared with GB1 protein, although the observations of double mutants are more sparse in the WW domain dataset, it contains higher order mutants with more complex fitness landscapes, which act as a good supplement to the GB1 dataset. Table 1 highlights some statistics of both datasets. Both the maximal fitness value and 99th percentile of high order mutants are much larger than those of single mutants, which indicates the importance of efficient optimization in this high-dimensional combinatorial space.

Table 1: Statistical data of fitness for each level of mutation in the two datasets

dataset	mutation level	variants number	maximal fitness	99% percentile
GB1 Protein	single	1045	5.02	2.68
	double	509,693	14.46	3.15
WW Domain	single	478	7.86	4.66
	double	17,389	38.07	4.60
	triple	6,115	37.51	8.34
	4+	681	20.26	6.42

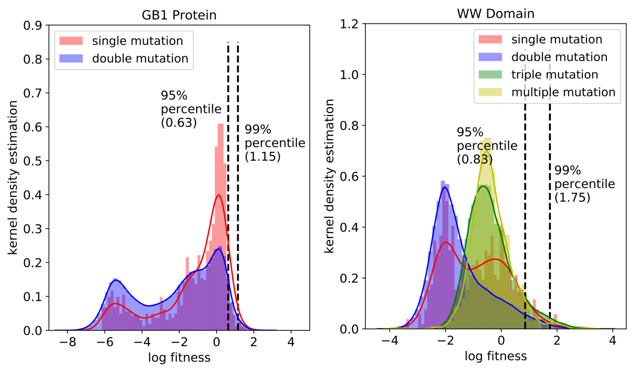


Figure 2: Histograms and kernel density estimations of log fitness values of different level of mutants in the two datasets.

5.2 Supervised Learning with TAPE embedding

Before digging into protein sequence design, it is helpful to analyze the relationship between TAPE embedding and the fitness value for each level mutants, and check the performance of the supervised learning task to make accurate prediction on the fitness value with TAPE embedding, as the ability to accurately map sequence embeddings to fitness is one of the foundations of the neural contextual bandits algorithm.

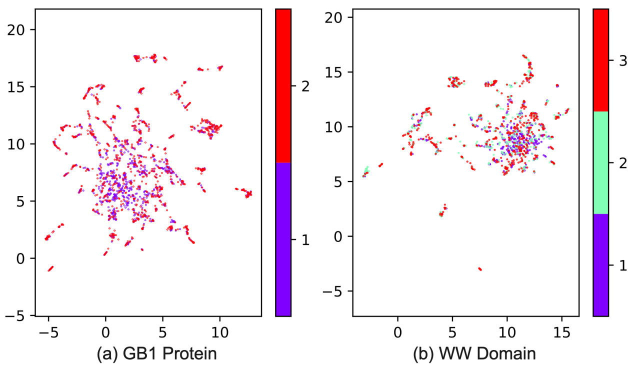


Figure 3: **UMAP projection of TAPE embedding of each level of mutations in GB1 and WW domain datasets.** For ease of visualization, random subsets of double and triple mutants are selected, so that the subsets have the same size as the set of single mutants. Color indicates the level of mutation.

To better visualize the high dimensional embedding, we utilize UMAP (McInnes et al., 2018) to project the embeddings into a two-dimension space and generate a scatter plot, as shown in Figure 3 and Figure 4. Proteins are marked with colors representing their levels of mutation in Figure 3. Both datasets consist of a dense center made up of all orders of mutations surrounded by smaller clusters that are composed of higher-order mutants. Meanwhile, points in the peripheral clusters are more likely to have extreme fitness values, while points in the center cluster tend to have a close-to-average fitness. This indicates that some of the high-order mutants are far away from the wild type and other mutants in the embedding space, and thus potentially possess different properties in terms of binding affinity.

Furthermore, we define the supervised learning task for each dataset with a 3-layer MLP model with the Swish activation function (Ramachandran et al., 2017) and a fixed number of single mutants training data points (600 for GB1 protein and 300 for WW Domain). Then, we change the proportion of double mutants in the whole training data and test the model performance on test data for different levels of mutations. Table 2 shows the R^2 values for prediction.

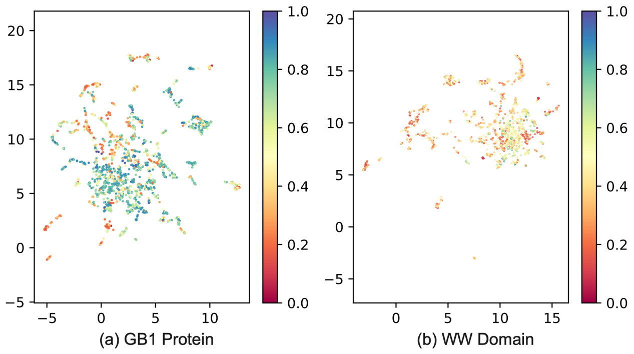


Figure 4: The same UMAP projection as in Figure 3. The color indicates the log fitness value normalized by minimum and maximum.

Table 2: Supervised learning R^2 for each proportion of double mutants in MLP training data of the GB1 and WW domain dataset.

GB1 Protein	train	validation	test - single	test - double
0%	0.991±0.003	0.785±0.022	0.749±0.014	0.634±0.026
50%	0.993±0.002	0.789±0.026	0.746±0.043	0.648±0.026
100%	0.993±0.002	0.724±0.015	0.724±0.015	0.640±0.022

WW Domain	test - single	test - double	test - triple	test - 4+
0%	0.455±0.034	0.501±0.005	0.250±0.006	0.110±0.009
50%	0.470±0.050	0.494±0.025	0.255±0.022	0.144±0.024
100%	0.386±0.062	0.491±0.018	0.274±0.013	0.159±0.025

In both datasets, the model can predict well for single and double mutants. Furthermore, the model is able to generalize well to double mutants when trained with just single mutation data. However, performance drops significantly for higher order mutants, which likely have more epistatic interactions and larger distribution shift.

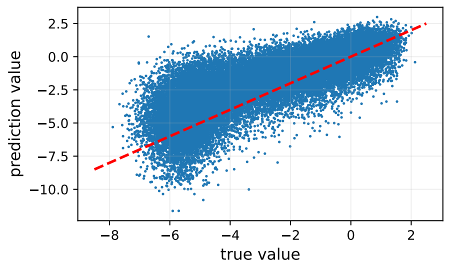
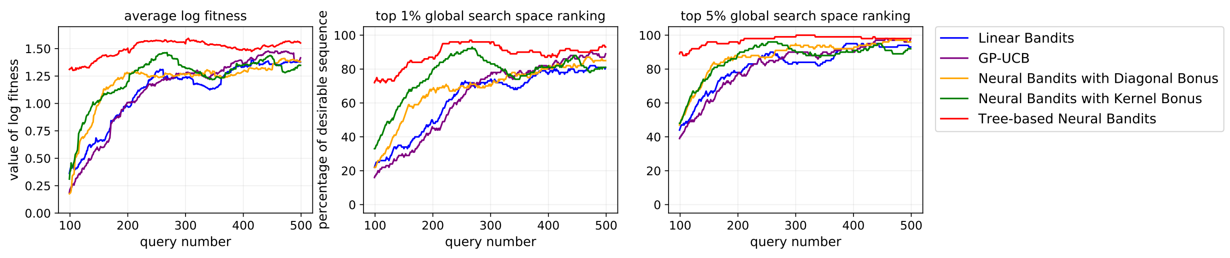
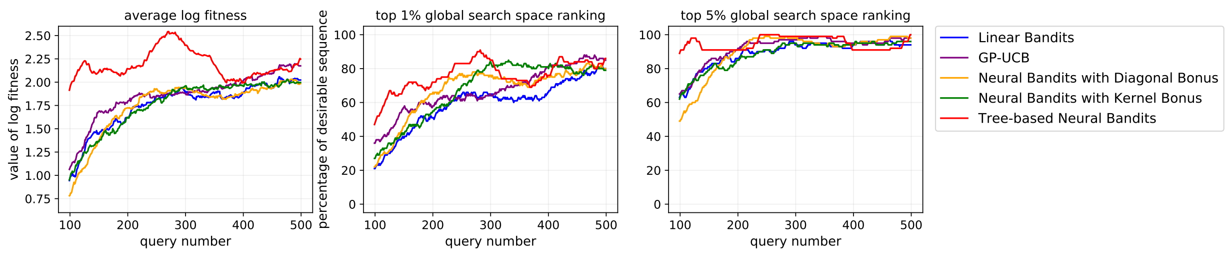


Figure 5: **Models trained by supervised learning tends to predict high-value proteins more accurately than low-value ones.** The figure plots the true log fitness value against MLP predicted value for double mutants in GB1 data when using 100% single mutants training data. The overall R^2 for double mutants test data is 0.634 and the $y = x$ line is plotted in red color.

By plotting the true value versus the MLP prediction value for double mutants in GB1 dataset in Figure



(a) GB1 Protein



(b) WW Domain

Figure 6: **Model performance with regard to the two metrics for each backward rolling 100 sequences.** Five models including the tree-based neural bandits are tested on two benchmark datasets, and evaluated based on two types of metrics calculated on a rolling window basis, the average log fitness and the global search space ranking, as described in Section 5.3.

5, we see that the model predicts high fitness variants with higher accuracy than for low fitness variants. This is typical for such datasets as higher fitness variants are usually more stable and have lower experimental error. This phenomena is amenable to the bandit setting, as we are more interested in the high fitness regions of the fitness landscape than the low fitness regions.

5.3 Results

To verify the efficiency of our model and the effect of each key components in our model, we compare the query efficiency of the tree-based neural bandits model with four other models on the two benchmark datasets, GB1 Protein and WW Domain:

1. **Linear Bandits:** Linear bandits utilize regularized linear regression with TAPE embedding inputs and make queries based on linear UCB.

2. **GP-UCB:** We run the batch GP-UCB algorithm (Desautels et al., 2012) with an RBF kernel.

3. **Neural Bandits with Diagonal Bonus:** Use neural UCB to decide which sequence to query. For the intractable bonus calculation due to the high dimensional gradient vector, as Zhou et al. (2020) did in their experiments, only use the diagonal elements to approximate the Gram matrix.

4. **Neural Bandits with Kernel Bonus:** Instead of a diagonal representation which would lose much information, utilize the kernel tricks for dimension reduction when calculating the bonus term.

We have also tried Bayesian optimization, but these results are omitted as Bayesian optimization performed very poorly compared to the other listed methods. This is most likely due to the high-dimensionality of the TAPE embeddings. To measure model performance, we utilize two metrics on a rolling window basis. For each time of a model's query, the metrics are computed for the 100 most recent queries. We define the two metrics:

1. **Global search space ranking:** the percentage of sequences in the 100 variants that are within the top 1% / 5% of the global fitness space (as a reference, log fitness 99th percentile is 1.17 for GB1 and 1.75 for WW).

2. **Average log fitness:** The previous metric focuses on the ranking perspective, while the average log fitness is more straightforward.

For each model, we use an initialization set of 200 random single mutants and train the MLP model with the same hidden dimension size. We perform a grid search to optimize the model regularization parameter

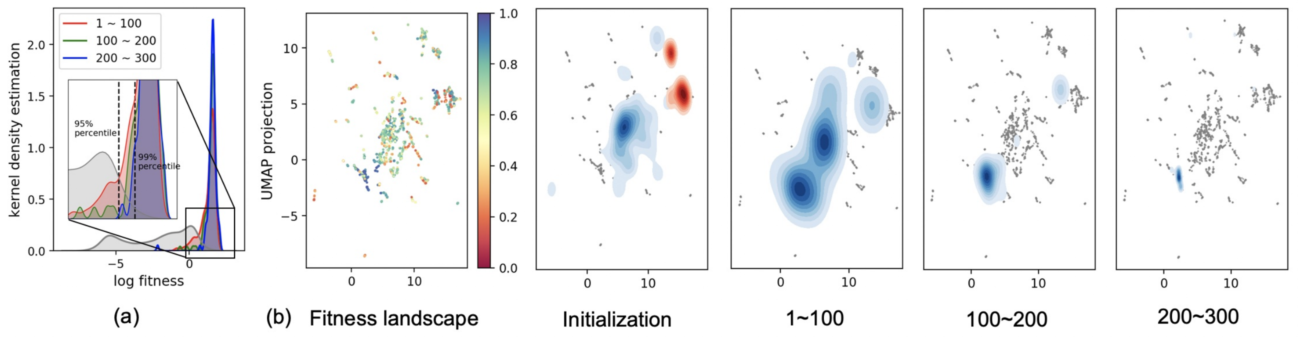


Figure 7: **Sampling distribution shifts as the tree-search bandit algorithm learns to identify top-value proteins.** (a) Fitness distribution shift during rounds of model queries of GB1 dataset; (b) Contour plot on UMAP projection of embedding for the initialization data and every 100 rounds of queries. The contours in blue correspond to queries made by the tree-based neural bandits model, and the red contour in the initialization figure is the distribution of queries made by Wittmann et al. (2021) to train their ML-assisted DE model. More details in section 5.3.

λ and the exploration-exploitation trade-off parameters γ and C. To mitigate the cost to retrain the NN, we retrain every 20 new queries. The detailed experimental settings are described in Appendix B.

The overall results are illustrated in Figure 6. We see that each of the components of our tree-based neural bandits model, including neural bandits, kernel tricks, and MCTS, play an active role to increase search and query efficiency. It is worth noting that for the GB1 dataset, our model is able to achieve more than 80% of the top 1% queries in terms of global space ranking, when pre-trained on 200 single mutation variants and trained with less than 200 additional queries. Furthermore, with these queries, it achieves a rolling average log fitness of over 1.25, which is about 3.5 times of the wild type fitness. This indicates that our model can efficiently identify a rich and diverse class of high fitness sequences.

From Figure 6, we see that our model achieves a much higher average fitness level throughout all query iterations. This indicates how MCTS would guide exploration in neural bandits: at initialization, it guides the bandit algorithm to search for high predicted fitness branches of the tree by the integrated UCB. For branches with very few queries, U^{neuro} will give a tighter bound with the assistance of neural network models; for branches that have been explored to a large extent, U^{MCTS} works by gathering information from previous queries.

To analyze the search trajectory, we plot the fitness distribution shift and the queried sequences UMAP projection shift of the initialized state and every 100 rounds of queries of the tree-based neural bandits on the GB1 dataset in Figure 7. Figure 7(a) demonstrates that the queried sequences concentrate on high fitness variants with increased number of queries. Figure 7(b) highlights the way that exploration and exploitation are conducted. From initialization to 100 queries, the model does sufficient exploration in the global space and finds several local optima. In further steps, the queries concentrate in a small area of UMAP projection space that corresponds with high fitness variants. We plot the distribution of the sequences selected by Wittmann et al. (2021) in red in the initialization subplot. Their research focuses on the four-site mutants dataset by Wu et al. (2016) and 384 sequences are selected as the training data for the MLDE model. We observe that these variants do not cover high fitness variants in global variant sequence space.

6 Conclusions

We propose a tree-based neural bandits algorithm for efficient combinatorial optimization in a large search space, and utilize it for protein design. We demonstrate that our algorithm finds rich and diverse protein sequences with favorable qualities in fewer queries than other models. Our algorithm is also quite flexible as we can replace the TAPE embeddings with more expressive embeddings like MSA based embeddings (Rao et al., 2021), which might allow implicit consideration of high-order epistasis.

References

Araya, C. L., Fowler, D. M., Chen, W., Muniez, I., Kelly, J. W., and Fields, S. (2012). A fundamental protein property, thermodynamic stability, revealed solely from large-scale measurements of protein function. *Proceedings of the National Academy of Sciences*, 109(42):16858–16863.

Arnold, F. H. (1998). Design by directed evolution. *Accounts of Chemical Research*, 31(3):125–131.

Bedbrook, C. N., Yang, K. K., Robinson, J. E., Mackey, E. D., Gradinaru, V., and Arnold, F. H. (2019). Machine learning-guided channelrhodopsin engineering enables minimally invasive optogenetics. *Nature Methods*, 16:1176—1184.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Desautels, T., Krause, A., and Burdick, J. (2012). Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization. In *International Conference on Machine Learning*.

Esposito, D., Weile, J., Shendure, J., Starita, L. M., Papenfuss, A. T., Roth, F. P., Fowler, D. M., and Rubin, A. F. (2019). Mavedb: an open-source platform to distribute and interpret data from multiplexed assays of variant effect. *Genome biology*, 20(1):1–11.

Frazier, P. I. (2018). A tutorial on bayesian optimization.

Frisby, T. S. and Langmead, C. J. (2021). Bayesian optimization with evolutionary and structure-based regularization for directed protein evolution. *Algorithms for Molecular Biology*, 16(1):1–15.

Hie, B., Bryson, B. D., and Berger, B. (2020). Leveraging uncertainty in machine learning accelerates biological discovery and design. *Cell Systems*, 11:461–477.

Hie, B. L. and Yang, K. K. (2021). Adaptive machine learning for protein engineering.

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106.

Liu, G., Zeng, H., Mueller, J., Carter, B., Wang, Z., Schilz, J., Horny, G., Birnbaum, M. E., Ewert, S., and Gifford, D. K. (2019). Antibody complementarity determining region design using high-capacity machine learning. *Bioinformatics*, 36(7):2126–2133.

Luo, Y., Jiang, G., Yu, T., Liu, Y., Vo, L., Ding, H., Su, Y., Qian, W. W., Zhao, H., and Peng, J. (2021). Ecnet is an evolutionary context-integrated deep learning framework for protein engineering. *Nature Communications*, 12(5743).

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Olson, C. A., Wu, N. C., and Sun, R. (2014). A comprehensive biophysical description of pairwise epistasis throughout an entire protein domain. *Current biology*, 24(22):2643–2651.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions.

Rao, R., Bhattacharya, N., Thomas, N., Duan, Y., Chen, X., Canny, J., Abbeel, P., and Song, Y. S. (2019). Evaluating protein transfer learning with tape. *Advances in neural information processing systems*, 32:9689.

Rao, R., Liu, J., Verkuil, R., Meier, J., Canny, J. F., Abbeel, P., Sercu, T., and Rives, A. (2021). Msa transformer. *bioRxiv*.

Romero, P. A. and Arnold, F. H. (2009). Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology*, 10:866–876.

Romero, P. A., Krause, A., and Arnold, F. H. (2013). Navigating the protein fitness landscape with gaussian processes. *Proceedings of the National Academy of Sciences USA*, 110(3):E193–E201.

Wittmann, B. J., Yue, Y., and Arnold, F. H. (2021). Informed training set design enables efficient machine learning-assisted directed protein evolution. *Cell Systems*.

Wu, N. C., Dai, L., Olson, C. A., Lloyd-Smith, J. O., and Sun, R. (2016). Adaptation in protein fitness landscapes is facilitated by indirect paths. *Elife*, 5:e16965.

Wu, Z., Kan, S. B. J., Lewis, R. D., Wittmann, B. J., and Arnold, F. H. (2019). Machine learning-assisted directed protein evolution with combinatorial libraries. *Proceedings of the National Academy of Sciences*, 116(18):8852–8858.

Yang, K. K., Wu, Z., and Arnold, F. H. (2019). Machine-learning-guided directed evolution for protein engineering. *Nature Methods*, 16:687–694.

Zhou, D., Li, L., and Gu, Q. (2020). Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pages 11492–11502. PMLR.

A Approach: Details on the Integrated UCB Calculation

As stated in Section 4.3, we select top N nodes from the tree structure according to the integrated UCB in Eq. 3. Based on the characteristics of the protein mutation tree, we conduct several modifications to handle the different types of nodes in MCTS as follows:

A.1 Avoid getting stuck in local optima.

To make sure that our model does sufficient exploration of the global search space in the tree structure and does not get stuck in local optima, we add an additional term *add* to the original UCB value for the non-terminal nodes set $\mathcal{N}_1$ containing nodes which have not been visited or have been visited only once (i.e. its children nodes have not been visited):

$$U_{n_t \in \mathcal{N}_1}^{inte} = \min\{U_{n_t}^{neuro}, U_{n_t}^{MCTS}\} + add \tag{5}$$

Note that the original bound is the tighter one between MCTS bound and neural UCB, thus it is hard to reach a good balance between exploration and exploitation by merely increase the tradeoff parameters γ and C of the two bounds. Other methods like ε-greedy fail to work well due to the property that high fitness sequences are highly concentrated in several single mutation branches of the tree.

Besides, we impose constraints on visiting frequency of both terminal and non-terminal nodes. Each non-terminal nodes can not be visited more than p in proportion to total query number, and each terminal node can only be visited less than v times.

A.2 Deal with epistasis phenomenon.

Epistasis is a phenomenon in genetics in which the effect of a gene mutation is dependent on the presence or absence of mutations in other genes or other positions. This may lead to high fitness double mutants whose parent single mutant nodes have low fitness value. To handle this problem, we use the MCTS bound itself instead of the integrated bound for the non-terminal nodes set $\mathcal{N}_2$ containing nodes with visited children nodes:

$$U_{n_t \in \mathcal{N}_2}^{inte} = U_{n_t}^{MCTS} \tag{6}$$

With this, the selection criterion depends on the average value of visited nodes in this branch without being bounded by the potentially low value of this node.

A.3 Terminal node bonus discount.

In the protein design scenario, the depth of tree structure is much smaller than other MCTS application scenarios. Since we hope to achieve a rich and diverse class of high fitness mutants, the bonus discount d is included for the nodes set $\mathcal{N}_3$ containing terminal nodes that have been visited (and thus have precise fitness value):

$$U_{n_t \in \mathcal{N}_3}^{inte} = \min\{U_{n_t}^{neuro}, U_{n_t}^{MCTS}\} \tag{7}$$

where

$$U_{n_t}^{MCTS} = \overline{V_{n_t}} + \frac{C}{d}\sqrt{\log N(parent(n_t))/N(n_t)} \tag{8}$$

With hyper-parameter $d > 1$, the bonus terms of visited terminal nodes are discounted, which lower the algorithm's preference for the nodes that we have already gathered full information and thus do not gain better knowledge by querying it again.

B Experiments

B.1 Data Collection and Preprocessing

We utilized two public protein fitness datasets in our experiments, the GB1 protein and the WW Domain. The GB1 dataset is available in the **supplementary file** of Olson et al. (2014). We calculate the fitness value for each mutants based on the number of the input library and the output selected library given in this file.

The WW Domain dataset is available in the **MaveDB database** (Esposito et al., 2019). We utilize the amino acid variant scores file which contains 30487 unique protein sequences. Furthermore, we drop the sequences with mutations to the initiator codon or the terminator codon, and drop the low confidence sequences with fitness standard error greater than 0.25.

For both datasets, we input the protein sequences and get their TAPE embedding via the **tape library**.

B.2 Experimental Settings

We evaluate the performance of the tree-based neural bandits model and the other four models as stated in Section 5.3 of the paper on the two datasets described above. We implemented our model with the Pytorch architecture on a computer with an Intel Core i5 CPU and a 8 GB RAM.

In our experiments, the dimension of TAPE embedding is 768, and we set the number of rounds T as 500, number of sampled sequences from MCTS (for the tree-based neural bandits) or random sampling (for other four models) N as 1000, number of pretraining data n_{pre} as 200, neural network hidden dimension m as 128, MLP batch size b as 20, and epoch number J as 50. Besides, for the GB1 dataset, we set the additional term add in Section A.1 to be 2, the visiting frequency constraint p to be 0.3, v to be 3, and the bonus discount d to be 10. For the WW Domain dataset, we set add to be 1, p to be 0.1, v to be 10, and d to be 10. We use grid search to tune the MCTS tradeoff parameter C over $\{0.5, 1, 1.5\}$, neural bandits exploration parameter γ over $\{10^{-3}, 10^{-2}, 10^{-1}\}$, the regularization parameter λ over $\{10^{-4}, 10^{-3}, 10^{-2}\}$ and the MLP learning rate η over $\{10^{-4}, 5 \times 10^{-4}, 10^{-3}\}$. We select $C = 1, \gamma = 0.01, \lambda = 10^{-3}, \eta = 5 \times 10^{-4}$ for both datasets.

For the other four benchmark models, we conduct similar grid search for hyper-parameters. For linear bandits, we select the exploration parameter γ as 0.1 over $\{0.01, 0.1, 0.5\}$ and the regularization parameter λ as 0.1 from $\{0.01, 0.1, 1, 3\}$ for both datasets. For neural bandits with diagonal bonus, we select γ as 0.01 over $\{10^{-3}, 10^{-2}, 10^{-1}\}$, and λ as 0.01 for GB1 and 10^{-3} for WW over $\{10^{-3}, 10^{-2}\}$. For neural bandits with kernel bonus, we select γ as 0.01 over $\{10^{-3}, 10^{-2}\}$ and λ as 10^{-3} over $\{10^{-3}, 10^{-2}\}$ for both datasets. For GP-UCB, we select γ as 0.01, λ as 10^{-4}, and the RBF kernel parameter ω as 10^{-3} over $\{10^{-3}, 10^{-2}\}$.